КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет радіофізики, електроніки та комп'ютерних систем

Кафедра комп'ютерної інженерії

«Розробка інтерактивної карти факультету»

Дипломна робота бакалавра

студента 4 року навчання

Спеціальність: 123 «Комп'ютерна інженерія»

Даниїла Мандрики

Науковий керівник

Кандидат фізико-математичних наук

доцент Олександр БАРАБАНОВ

Рецензент

Кандидат фізико-математичних наук

доцент А. ШКАВРО

До захисту допускаю Завідувач кафедри Юрій БОЙКО

Протокол засідання кафедри від
“___” ___________2022 р., протокол №_______

Київ 2022

Реферат

Кваліфікаційна робота бакалавра складається з :

- 78 сторінок;
- 87 рисунків;
- 7 таблиць;

В кваліфікаційні роботі розглянуто розробку інтерактивної карти з використання програмного забезпечення Unity Engine, Rider IDE та 3D DCC редактору Blender.

Розглянуто :

- 8 підмодулів Інструменту Unity Engine Particle.Systems.

Реалізовано :

- 9 алгоритмів, що реалізують важливий функіонал під час роботи додатку.

Зміст

Реферат....2
Зміст....3
Список скорочень....5
Вступ....6
Мета кваліфікаційної роботи бакалавра....8
Розділ I. Обраний стек технологій....9
§1. Unity....9
§2. Blender....9
§3. Rider....10
Розділ II . Компоненти Unity Engine за допомогою яких реалізована побудова траєкторію маршруту....11
§4. Позиціювання об`єктів(Transform)....11
§5. Генератор частинок(Модуль Particle System)....14
Пункт 5.1. Головні налаштування(ParticleSystem.MainModule)....15
Пункт 5.2. Випромінення (ParticleSystem.emission)....16
Пункт 5.3. Форма(ParticleSystem.shape)....17
Пункт 5.4. Колір, який змінюється з рухом(Color by Speed module)....17
Пункт 5.5. Розмір протягом усього часу існування (Size over Lifetime module)....18
Пункт 5.6. Тип відображення(ParticleSystem.Renderer)....18
§6. MeshRenderer component....20
Розділ III. Алгоритми, що були розроблені для відображення траєкторії маршруту.....21
§7. Алгоритм відображення маршруту між двома опорними точками....21
§8. Конструкція поверхів....24
Пункт 8.1. Шостий поверх лаб-корпусу(_Floor_6_LAB_WithConnector(1)):....25
Пункт 8.1.1. Об’єкт _WrapperFlour_6 (Обкладинка)....26
Пункт 8.1.2. Алгоритм знаходження геометричного центру фігури.....27
Пункт 8.1.3. Видимі об’єкти (visible components)....31
Пункт 8.1.4. Дерево опорних точок (F_6_path)....33
Пункт 8.2. Третій поверх аудиторного корпусу (_Floor_3_AUD_WithConnector(8))....36
§9. Декомпозиція пошуку маршруту між початковою та кінцевою аудиторіями....38
Пункт 9.1 FindStartRoomOnTheLF6AndPathToAnotherFloorWithStairsR()....41
Пункт 9.1 Загальна схема роботи Алгоритму з пошуку маршруту з використанням розбиття поверхів на підмножини....50
§10 Відображення підписів аудиторій на поверхах....59
Пункт 10.1 Алгоритм побудови підпису аудиторій....60
§11 Алгоритм знаходження необхідної кількості створених генераторів частинок, при побудові маршруту....63
Розділ IV. UI з яким взаємодіє користувач....64

§12 Обробка даних введених користувачем....66
§13 Початок відображення маршруту та його припинення....67
§14 Обертання інтерактивної карти....69
§14 Збільшення та зменшення масштабу відображення карти....72
Пункт 14.1 Camera component....73
§15 Повертання видимих частин карти в початкове положення....75
Розділ V. Проміжні результати практичної частини. 76
Виконана робота....77
Висновки....77
Перелік посилань....77
Додаток А....78

Список скорочень

3D DCC (**Digital Content Creation**) - програмне забезпечення, яке використовується для створення 3D - моделей, анімацій.

Приклади **ПО**:

- **Blender** (використовувався під час виконання практичної частини);
- **ZBrush;**
- **Maya;**
- **Autodesk 3dsMax;**

IDE (Integrated Development Environment) – набір програмних рішень, для полегшення процесу програмування.

Приклади **ПО**:

- **Visual Studio;**
- **Visual Studio Code;**
- **Rider** (використовувався під час виконання практичної частини)

P(particle**) –** частинка, об’єкт створений інструментом **Particle Systems**.

PObj (**ParentObject**) – батьківський об’єкт. Від його позиції, яку ми можемо побачити завдяки компоненту **transform**, вираховується позиція всіх дочірніх об’єктів. При його переміщенні всі дочірні об’єкти рухаються разом з ним, зберігаючи встановлену позицію відносно позиції батьківського об’єкту.

ChObj (**ChildObject**) – дочірній об’єкт, позиція якого вираховується відносно позиції **PObj.**

Вступ

Проблема навігації людей у нових офісних будівлях при переїзді чи при першому робочому дні. Чи пошук потрібного магазину в великих торговельних центрах. Чи пошук маршруту до найближчого бомбо-сховища. Всі ці проблеми будуть постійно з`являтися в житі людей.

Шляхи вирішення цієї проблеми мають різні види реалізації.

Найчастіше люди для навігації використовують плани евакуації при пожежі, які за умови сумлінних працівників знаходяться на кожному поверху. Найчастіша проблема, з даним методом, що заданий маршрут по якому має рухатись людина, вже не відповідає дійсності. Додаткова проблема, яка з`являється не у робочого персоналу будівлі, не кожна людина може здогадатися де знаходяться ці плани на кожному поверсі.

Для пошуку потрібного магазину у великих торговельних центрах використовують інформаційні стенди. Людина може підійти і обрати поверх який її цікавить, та шукати на ньому необхідний магазин. Ввести потрібний магазин і знайти шлях до нього або побачити поверх, на якому знаходиться потрібний магазин. Однак недолік в тому, що дані стенди найчастіше зайняті іншими людьми, і іноді потрібно чекати близько 15-ти хвилин, щоб отримати доступ до інформаційного стенду. Це не є завжди раціональне витрачання часу. Іноді ця система, з міркувань безпеки доступна лише на території торговельного центру. В людини відсутня можливість спланувати свій маршрут знаходячись в комфортних умовах.

Для навігації в паркових зонах використовуються великі білборди, на яких знаходиться карта всієї паркової зони та виділено місце на карті, яке відображає позицію даного стенду. Гарний приклад даної реалізації є територія Виставкового центру.

За період повномасштабної війни людьми було розроблені інтерактивні web-карти. В кожного в кого був вихід до мережі інтернет, міг скористатися нею та дійти до потрібного сховища. Однак виникали нюанси з тим, як саме проникнути в приміщення бомбосховища, яке іноді знаходилось в досить незрозумілих місцях, без потрібного виділення. Або навіть заблоковане. З плином часу ці проблеми вирішились. Однак ніхто не забезпечений від подібних проблем. І якщо людина не могла знайти час на знаходження найкоротшого маршруту, під час екстреної ситуації подібні помилки можуть вартувати життя.

Вирішення подібної проблеми головною цілю даної кваліфікаційної роботи.

Прикладом дослідження було обрано територію рідного факультету радіофізики та комп`ютерних систем.

Також подібна задача є досить важливою на інших факультетах нашого університету, в яких інфраструктура набагато більша.

Так як більшість студентів мають мобільні телефони на базі операційної системи андроїд, мною було вирішено створити додаток саме під цю операційну систему. В майбутньому планується розробка і під інші ОС.

Вирішенням подібних задач займаються різні компанії з розробки програмного забезпечення та іноді за це беруться студенти вищих навчальних закладів.

Гарним прикладом компанії з широкими можливостями є компанія MapBox.

На їхньому інформаційному порталі, ви знайдете безліч API, які допоможуть вирішити поставленні задачі навігаці та побудови маршруту.

Подібною проблемою займалися студенти НДУ(Новосибірського державного університету). Ними було розроблено додаток “мой НГУ” для мобільних пристроїв на базі операційної системи IOS.

Посилання на додаток знаходиться в переліку посилань.

Мета кваліфікаційної роботи бакалавра

Метою даної роботи розробка мобільного застосунку, який представляє з себе інтерактивну карту нашого факультету, з можливістю побудови маршруту між початковою та кінцевою аудиторіями.

Дану мету більш детально розділено на такі підзадачі:

- Знаходження потрібного інструментарію(від його вибору залежить можливості розробника, та майбутня підтримка застосунку).
- Детальний розбір технологій та принципу свторення об'єктів, що використовувались при розробці інтерактивної карти.
- Написання зрозумілого алгоритму пошуку маршруту та розміщення проекту на доступному для всіх репозиторію.
- Розробка мінімального UI та описання логіки взаємодії з ним.
- Показати проміжні результати виконаної роботи.

Розділ I. Обраний стек технологій

§1. Unity

Хоча Unity і починав своє існування як доступний ігровий движок для інді-розробників, з плином часу даний движок розвився. На сьогоднішній день крім розробки ігор можливості розробки движку включають в себе такі напрямки:

- Мобільна розробка;
- Робота з різними платформами(реалізація кросплатформи);
- Відеомонтаж. рендеринг відео та створення відеорядів;
- AR та VR;
- Створення архітектурних рішень;
- Навчання учнів загально освітніх шкіл у сфері точних наук, так як движок має гарно прописані властивості фізичного світу.
- Робота зі музикальними рядами та ефектами.

Також однією з переваг є відкритість документації, що є важливим для розробника. Дружнє та широке ком'юніті, яке існує вже не перший рік. Тому при пошуку проблеми є ймовірність знайти вирішення вашої проблеми.

Підтримка зі сторони розробників движку. ПО постійно підтримується та розвивається.

Низька ціна асетів, які використовуються при побудові проектів.

Можливість створювати свої додаткові модулі для движку, імпортувати їх та ділитися з іншими розробниками в внутрішньому магазині "Asset Store". У компанії MapBox також є свої модулі, які постійно підтримуються розробниками.

Чітко прописана політика використання движку та можливість займатися створенням некомерційних проектів.

Підтримка неймовірно великої кількості форматів файлів. Особливо зручно при роботі з 3D об'єктами.

Більшість студентів нашого факультету навчали програмуванню на мові C#, і як раз саме ця мова використовується при написнаі скриптів для Unity Engine.

§2. Blender

3D DCC редактор з відкритою документацією та взаємодією з більшістю програмних рішень для роботи з 3D моделями.

Достатньо широкий спектр інструментів для реалізації моделювання та створення анімацій.

Також повністю безкоштовна платформа. Ідеально підходить для розробки пед-проектів та навчальної практики.

§3. Rider

Ide від компанії **JetBrains –** спеціально створена для роботи з движком Unity.

Невисока затратність апаратних ресурсів, що дозволяє використовувати навіть студентам з досить бюджетним апартним устаткуванням, при цьому зникає проблема з вибором операційної системи, так як Rider підтримується на більшості ОС.

Розділ ІІ . Компоненти Unity Engine за допомогою яких реалізована побудова траєкторію маршруту

Для того щоб зрозуміти, як реалізована ідея малювання шляху розберемо такі частини движку Unity як :

- Позиціювання об`єктів.
- Інструмент, або як визначено в документації движку – модуль **Particle System.**

§4. Позиціювання об`єктів(Transform)

За позиціювання об'єктів в 3D-просторі в движку відповідає компонент **Transform**.

Для того щоб була можливість взаємодії з даним компонентом необхідно додати його на бажаний об'єкт. За умовчанням при створені об'єкту даний компонент додається движком.

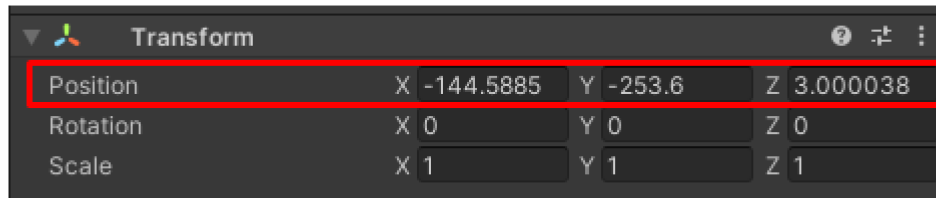

Рис. 1 Демонстрація полів компонента Transform.

Position – положення об'єкту в кординатах **X, Y, Z**;

Rotation – значення поворотів навколо осей **X, Y** і **Z**, вимірюється в градусах.

Scale – масштаб об'єкту. "1" – значення за умовчанням, що являє собою початковий розмір об'єкту.

Корисні властивості ваємодії **ParentObject(PObj)** між **ChildObject(ChObj)**:

1. При зміні положення **PObj, ChObj** рухаються разом за положенням **PObj**. Дана властивість використовується найчастіше при роботі з великою кількості, сполучених між собою об'єктів.
2. Значення положення(**position**), обертання(**rotation**) та масштабу(**scale**) трансформації(**Transform**) обраного об'єкту вимірюються відносно батьківського об'єкту.

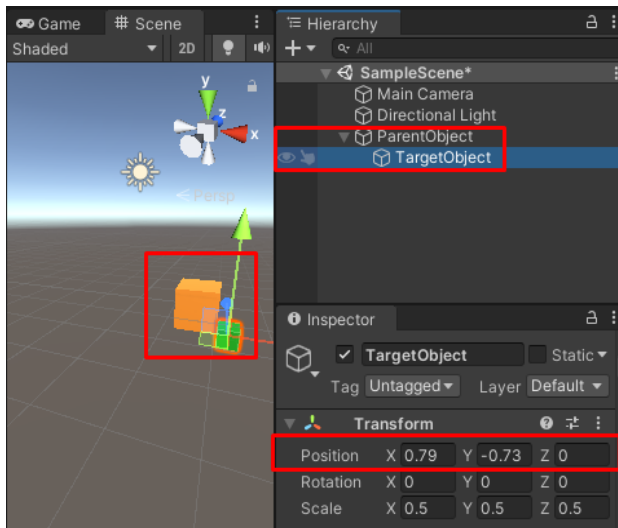

Рис. 2 Демонстрація початкової позиції **PObj,** та координати цільового об'єкту

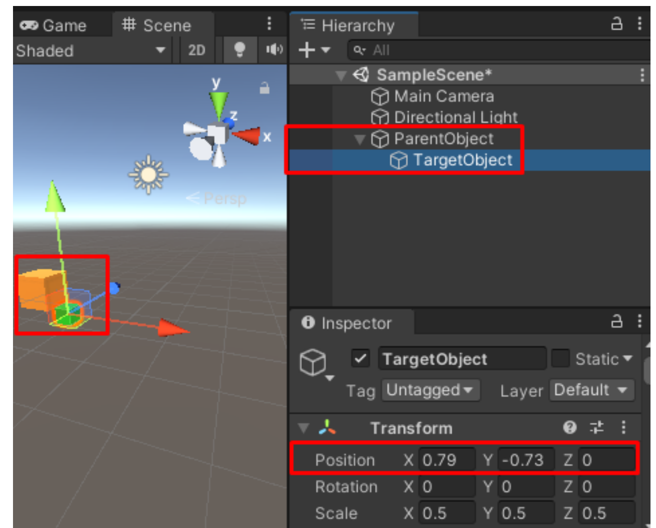

Рис. 3 Демонстрація кінцевої позиції **PObj**, та координати цільового об'єкту

При відсутності батьківського об'єкту розрахунки відбувається відносно світового простору:

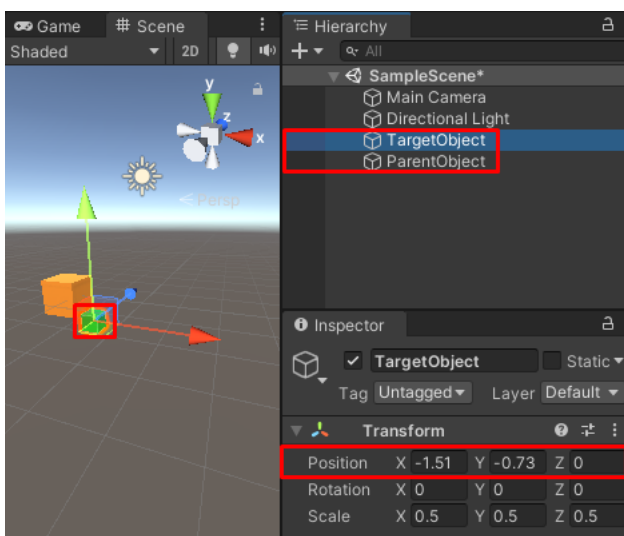

Рис. 4 Демонстрація початкової позиції цільового об'єкту у якого відстутній **PObj.**

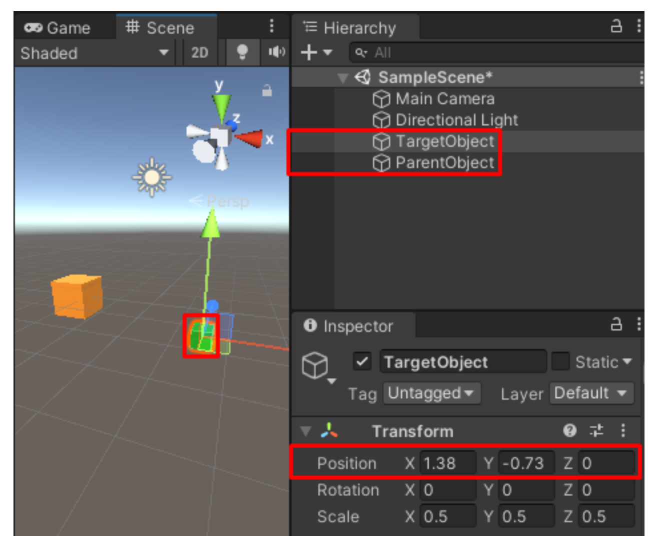

Рис. 5 Демонстрація кінцевої позиції цільового об'єкту, у якого відстутній **PObj.**

3. Для того, щоб дізнатися положення об'єкту досить просто звернутися до компонента **transform**, який знаходиться на обраному об'єкті.

Лістининг коду, написаний на мові програмування C#

```
using UnityEngine;

public class PositionExample : MonoBehaviour
{
    public GameObject TargetObject;

    // Start is called before the first frame update
    void Start()
    {
        Debug.Log(
            $" Target x: {TargetObject.transform.position.x}
                      y: {TargetObject.transform.position.y}
                      z: {TargetObject.transform.position.z}"
                );
    }
}
```

На рис.6 зображено виконання коду, який демонструє можливість отримати значення координат вибраного об’єкта.

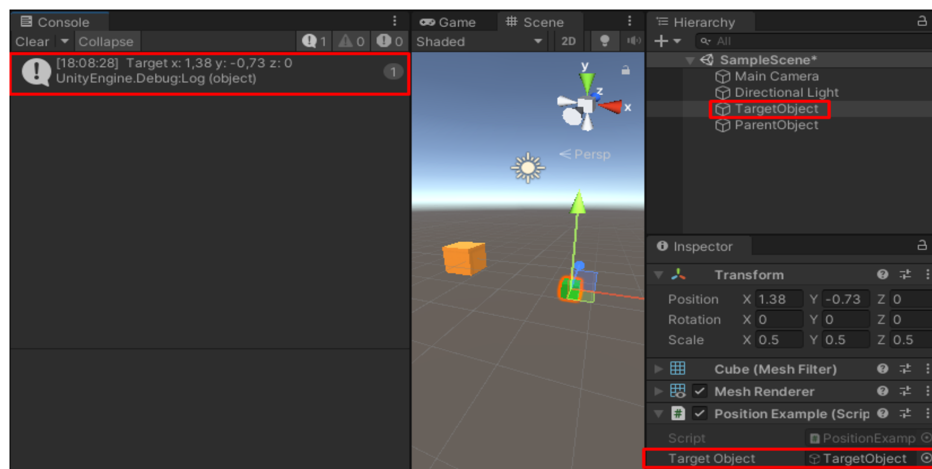

Рис. 6 Демонстрація можливості взаємодії з координатами об`єкту

§5. Генератор частинок(Модуль Particle System)

Модуль **Particle System** дозволяє моделювати рідкі об’єкти, такі як рідини, хмари та полум’я, створюючи та анімуючи визначену кількість як **2D** так і **3D** об’єктів.

На Рис.7 зображено стандартні варіації того, як можуть виглядати частинки, які створюються за допомогою **Particle System.**

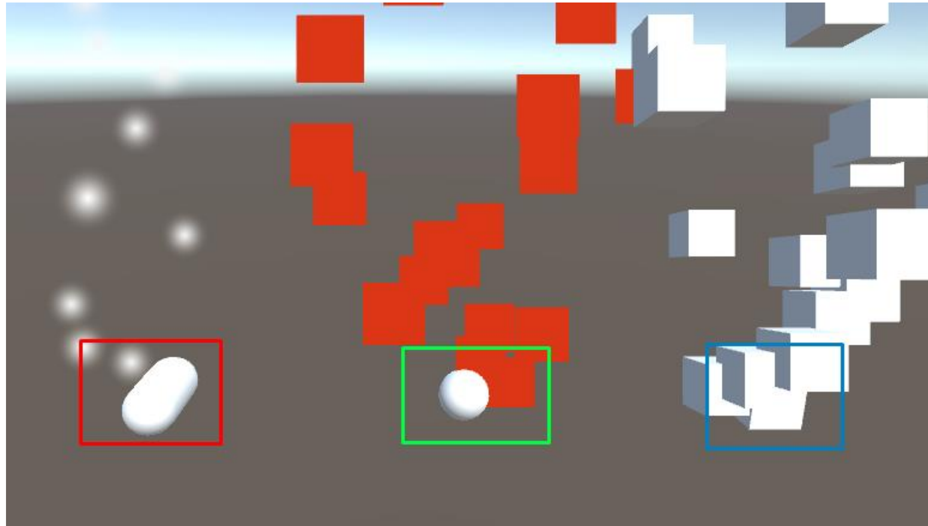

Рис. 7. Генерація стандартних частинок

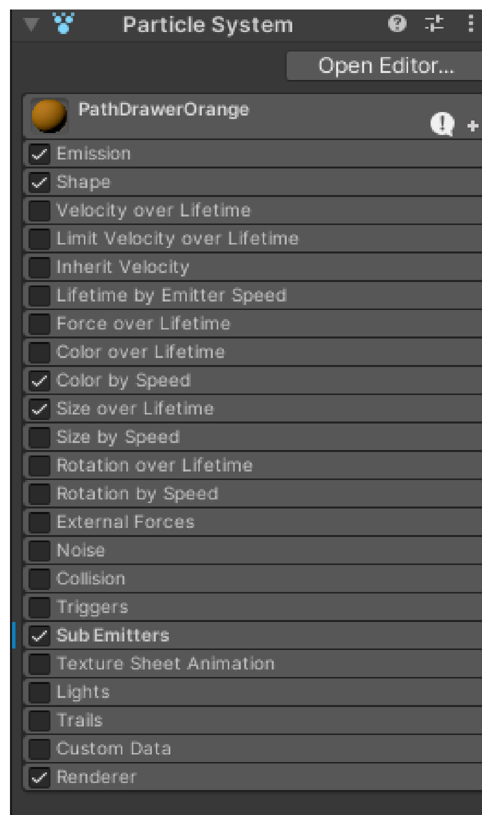

Рис. 8. Обрані найголовніші підмодулі генератору частинок, які були налаштовані для можливості створювати траєкторію маршруту.

Даний інструмент використовується для малювання траєкторії шляху, по якому має пройти людина, шукаючи шлях від початкової до кінцевої аудиторій.

Пункт 5.1. Головні налаштування(ParticleSystem.MainModule)

Отже, розглянемо потрібні налаштування, які має набути генератор частинок для вирішення поставленої задачі.

Рис. 9. Демонстрація полів, що використовувались при налаштування молуля – **MainModule**.

Duration (**тривалість**) – проміжок часу, протягом якого відбувається генерація частинок. Характеристика вимірюється в секундах.

Looping (**циклічність**) – чи є генерація частинок циклічна, чи ні. Після проходження визначного часу, який встановлений за допомогою властивості **Duration**, чи буде поновлюватись генерація частинок.

Start Lifetime (**початок життя**) – проміжок часу, протягом якого створена частинка існує. По закінченню даного часу, створена частинка знищується зі сцени

Start Speed (**початкова швидкість**) – з якою швидкістю буде рухатись частинка в вказаному напрямку, після її створення.

Start Size (**початковий розмір**) **-** початковий розмір частинок які вони отримують, в момент створення.

Scaling Mode (**режим масштабування**) **-** як система частинок застосовує свій компонент **Transform** до частинок, які вона випромінює.

- **Hierarchy** (**Ієрархія**) - масштабування відбувається відповідно до його трансформації та всіх її батьків.
- **Local** (**Локальний**) - масштабування з використанням лише власного компонента **Transform**, ігноруючи всіх батьків.
- **Shape** (**Форма**) – масштабування відбувається лише на вихідні положення частинок, але не їх розмір.

Play on Awake (**програш при події Awake**) **-** якщо встановлено значення true**, Particle System** автоматично почне відтворюватися під час запуску**.**

Emitter Velocity (**швидкість випромінення**) - як система частинок обчислює швидкість, яку використовують модулі Inherit Velocity та Emission. Система може обчислити швидкість за допомогою компонента **Rigidbody** (твердого тіла), якщо він існує, або шляхом відстеження руху компонента **Transform.**

Max Particles (**Максимальна кількість частинок**) - максимальна кількість частинок в системі які можуть знаходитись на сцені одночасно. Якщо межа досягнута, деякі частинки видаляються.

Auto Random Seed (**Автоматичне випадкове завантаження**) - якщо ввімкнено, **Particle System** буде виглядати по-різному під час кожного відтворення. Якщо встановлено значення **false**, система буде абсолютно однаковою під час кожного відтворення.

Culling Mode (**Режим відбракування**) – властивість допомагає налаштувати відображення за камерою. Можна обрати чи призупиняти моделювання системи частинок, коли частинки знаходяться поза екраном.

- **Automatic (Автоматичний) -** зациклені системи використовують паузу, а всі інші системи використовують **Always Simulate**.
- **Pause And Catch-up (Пауза і наздоганяння) -** система припиняє симуляцію в позакадровому режимі. Під час повторного входу до подання моделювання виконує великий крок, щоб досягти точки, де вона була б, якби її не призупинили. У складних системах цей параметр може довести до небажаних стрибків продуктивності.
- **Pause (Пауза) -** Система припиняє симуляцію в позакадровому режимі.
- **Always Simulate (Симуляція виконується завжди) -** Система обробляє його моделювання на кожному кадрі, незалежно від того, знаходиться він на екрані чи ні.

Пункт 5.2. Випромінення (ParticleSystem.emission)

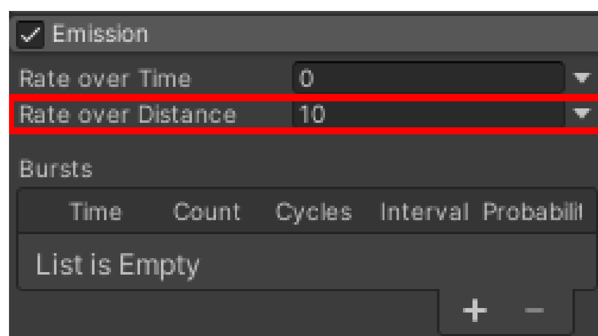

Рис. 10. Демонстрація полів, що використовувались при налаштування модуля – **Emision**.

Rate over Time (**кількість з плином часу**) - швидкість, з якою емітер породжує нові частинки з часом.

Rate over Distance (**кількість в залежності**) - швидкість, з якою емітер породжує нові частинки під часу руху. (Саме ця властивість буде найголовнішою, так як ми будемо переміщати генератор частинок, по заданій траєкторії, тим самим малювати користувачу шлях, який його/її цікавить)

Пункт 5.3. Форма(ParticleSystem.shape)

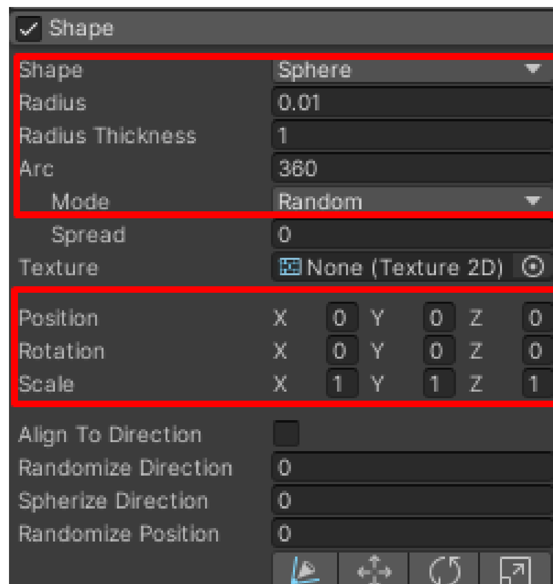

Рис. 11. Демонстрація полів, що використовувались при налаштування модуля – **Shape**.

Shape (форма) - тип форми, з якої виділяються частинки.

Radius (радіус) - радіус форми, з якої виділяються частинки.

Radius Thickness (товщина радіусу) - Радіус товщини краю фігури, з якого виділяються частинки.

ARC (дуга) - кут дуги кола для викиду частинок.

Пункт 5.4. Колір, який змінюється з рухом(Color by Speed module)

Рис. 12. Демонстрація полів, що використовувались при налаштування модуля – **Color by Speed**.

Color (колір) – градієнт для кольору частинки, визначений у діапазоні швидкостей.

Speed Range (діапазон швидкості) - Нижня і верхня межі діапазону швидкостей, на які зіставляється колірний градієнт (швидкості за межами діапазону відображатимуться до кінцевих точок градієнта).

Пункт 5.5. Розмір протягом усього часу існування (Size over Lifetime module)

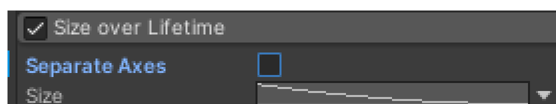

Рис. 13. Демонстрація полів, що використовувались при налаштування модуля – **Size over LifeTime**.

Separate Axes (окремі осі) – можливість задати різній розмір часток незалежно на кожній осі.

Size (**розмір**) – крива, яка визначає, як змінюється розмір частинки протягом її життя. (В нащому випадку, розмір частинок буде зменшуватись під кінець їхнього існування)

Пункт 5.6. Тип відображення(ParticleSystem.Renderer)

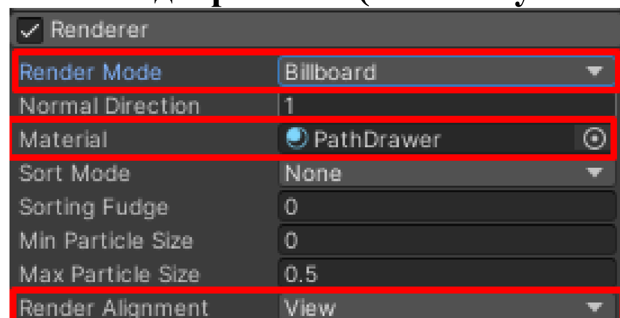

Рис. 14. Демонстрація полів, що використовувались при налаштування модуля – **Renderer**.

Даний модуль відповідає за вигляд частинок, які будуть генеруватися системою.

Render Mode (**режим відображення**) - вказує, як саме система створює частинки.

- **Billboard("рекламний шит") - Unity** відтворює частинки як рекламні щити, і вони спрямовані в напрямку, який ви вкажете в **Render Alignment**.
- **Stretched Billboard (розтягнутий білборд) -** частинки повернені до камери із застосованими різними можливими параметрами масштабування.
- **Horizontal Billboard (горизонтальний білборд) -** площина частинок паралельна площині «підлоги» **XZ**.
- **Vertical Billboard (вертикальний білборд) -** частинка знаходиться вертикально на світовій осі Y, але повертається обличчям до камери.
- **Mesh (сітка) -** Unity відтворює частинку з компонента **3D Mesh** замість використання **білборда**.

Material (матеріал) - матеріали які ви додаєте, автоматично створюються екземпляри матеріалів і система робить їх унікальними для цього засобу візуалізації. Можете налаштувати потрібний матеріал для відображення різних важливих шляхів.

Render Alignment (вирівнювання відображення) - властивість визначає напрямок, до якого спрямовані білборди з частинками.

- **View (перегляд) –** частинки завжди будуть звернені до площини камери.
- **World (загальний/світовий простір) -** частинки вирівнюються за напрямком світових осей**.**
- **Local (локальний/батьківський простір) -** частинки вирівнюються за компонентом **Transform** , який знаходиться на самих частинках.

- **Facing (лицьовий) –** частинки направлені в напрямок видимої частини об’єкту камери.
- **Velocity (швидкість) -** Частинки спрямовані в тому ж напрямку, що й їхній вектор швидкості.

Після налаштування потрібних властивостей, ми отримали генератор частинок який:

1. Створює частинки лише при русі самого генератора частинок.
2. Частинки завжди направлені в коректному напряму для користувача, тобто повернені до площини камери.
3. Також є можливість зміни кольорів, через додавання різних матеріалів до генератору частинок. Тим самим з’являється можливість коректно показати користвуачу альтернативні шляхи, не заплутавши при цьому користувача.

§6. MeshRenderer component

Компонент відтворює сітку полігонів, з яких складається об’єкт. Сітка позначенв 1 на рис. При використані компоненту Mesh Renderer(на рис позначено 2) є можливість відображення об’єкту

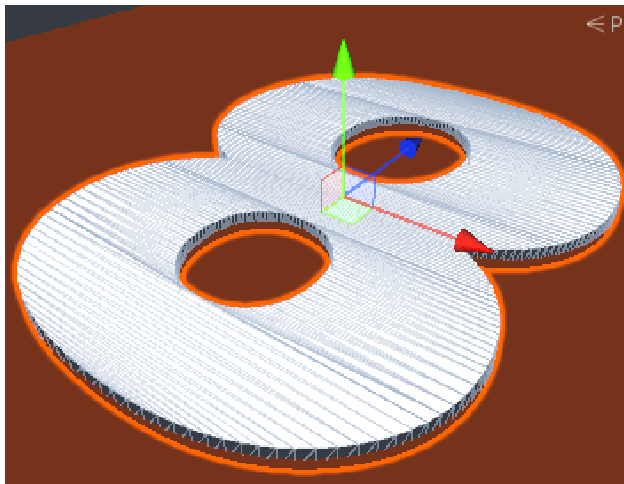

Рис.15

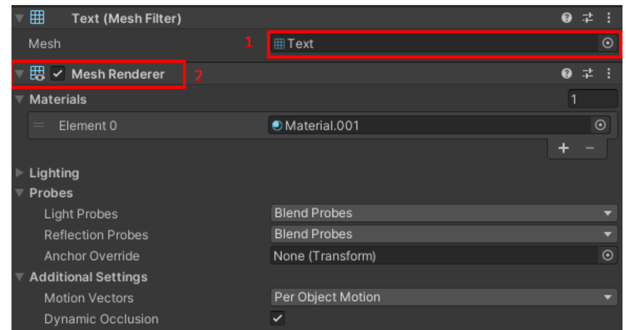

Рис.16

При відключенні компоненту Mesh Renderer, Unity Engine не відображає обраний об’єкт.

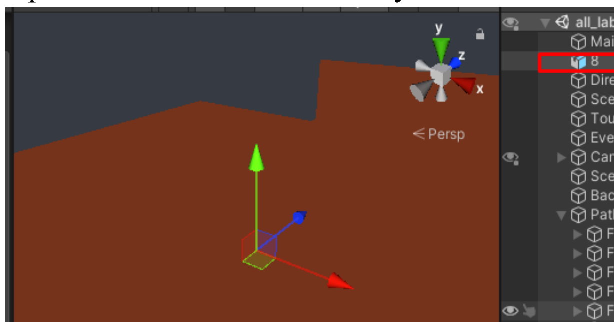

Рис.17

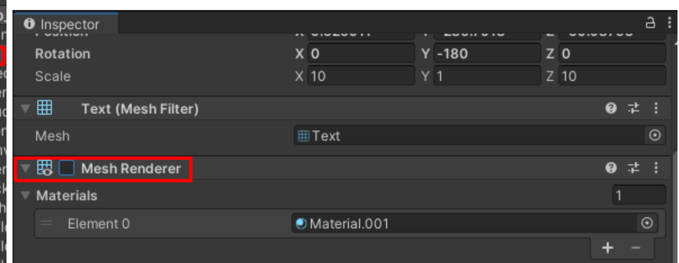

Рис.18

Найважливішу можливість, що має розробник використовуючи даний компонент - доступ до всіх вершин з яких складається полігональна сітка. А отже і до їх позицій в просторі.

Для реалізації доступу до важливих точок, з яких складається сітка в движку використовується клас **Bounds**

Bounds – преставляє з себе обмежувальну рамку вирівняну по осям . Обмежувальний квадрат, вирівняний по осі, або скорочено AABB, — це прямокутник, вирівняний за осями координат і повністю охоплює деякий об’єкт.

Поля, які будемо використовувати в алгоритмі:

Bounds.size – загальний розмір коробки.

- **size.x** – ширина;
- **size.y** – висота;
- **size.z** – глибина коробки;

Розмір вказаюється у розмірі світу, а не відносно батьківського об’єкту.

Розділ III. Алгоритми, що були розроблені для відображення траєкторії маршруту.

§7. Алгоритм відображення маршруту між двома опорними точками

Отже нам нічого не заважає дізнаватися позиції об'єктів завдяки компоненту **Transform**. Розберемо алгоритм малювання лінії лише між двома точками/об'єктами з повного шляху.

```
private IEnumerator DrawIndividualPath(GameObject PathDrawer, List<Transform>
PathListOfCoordinates)
{
    PathDrawer.SetActive(false);
    PathDrawer.transform.position = PathListOfCoordinates[0].position;
    yield return null;

    PathDrawer.SetActive(true);
    yield return null;

    var waitTime = 0.5f;

    for (int i = 0; i < PathListOfCoordinates.Count - 1; i++)
    {
        var elapsedTime = 0.0f;
        while (elapsedTime < waitTime)
        {
         PathDrawer.transform.position =

         Vector3.Lerp(
                      PathListOfCoordinates[i].position,
                      PathListOfCoordinates[i + 1].position,
                     (elapsedTime / waitTime)
                     );
         elapsedTime += Time.deltaTime;

         yield return null;
        }

        PathDrawer.transform.position = PathListOfCoordinates[i + 1].position;
        yield return null;
    }

    yield return new WaitForSeconds(PathDelay);

    StartCoroutine(DrawIndividualPath(PathDrawer, PathListOfCoordinates));
}
```

Ми знаємо, якщо генератор частинок(***в скрипті PathDrawer***) почне рухатись, він буде створювати частинки під час руху, які будуть нерухомо стояти на створеній позиції та знищуватись коли їх період життя дійде до кінця.

Залишилось лише зрозуміти куди рухати генератор частинок.

Уявимо, що ми записали в список(***в скрипті PathListOfCoordinates***) компоненти **Transform,** наших декількох точок.

Розберемо алгоритм на проході між першими двома точками А, Б. Так як дії алгоритму повторюються від однієї опорної і до наступної опорної точки. Тому немає сенсу розбирати кожну ітерацію циклу, який використовується в алгоритмі.

Так як швидкість оновлення кадрів, може бути набагато швидша ніж, швидкість з якою буде рухатись наш генератор частинок, використовуємо сопрограму, або як їх називають в документації unity - корутину.

За її допомогою ми можемо вказати кількість часу, яку ми маємо почекати після виконання попередньої інструкції перед виконанням наступної. Для цього використовується ключове слово **yield return,** далі йде вже вказаний проміжок часу**.**

Спочатку задаємо положенню генератора частинок координати положення першої опорної точки - точки А.

Якщо користувач змінить номер початкової аудиторії, йому не хотілось би бачити лінію, яку буде малювати генератор частинок при моментальній зміні початкового положення. Для уникнення малювання перескоків між початковими точками виконається команда, яка деактивує відображення генератора частинок. Після виконання інструкції переміщення в початкову позицію ми і чекаємо і не переходимо до активації генератора частинок, поки все ж таки генератор частинок не перемітиться на початкову позицію точки А :

```
PathDrawer.SetActive(false);
PathDrawer.transform.position = PathListOfCoordinates[0].position;
yield return null;
```

Після цього, активуємо відображення генератора частинок.

Далі, поки позиція **PS** не дорівнюватиме положенню точки Б, ми будемо лінійно інтерполювати відстань між ними. Для цього будемо використовувати функцію движка **Vector3.Lerp().**

```
Vector3.Lerp(Vector3 a, Vector3 b, float t)
```

a – поточна позиція **PS**;

b – кінцева позиція (***позиція точки Б***);

t - значення, яке використовується для інтерполяції між точками **a** та **b**.

Для більш гнучкого і плавного відображення руху, значення **t** – на кожній ітерації циклу дорівнює затраченому часові з початку виконання циклу, який ділиться на часову константу движка **Time.waitTime,** яку можна змінити в движку змінивши значення поля **Fixed Timestep.**

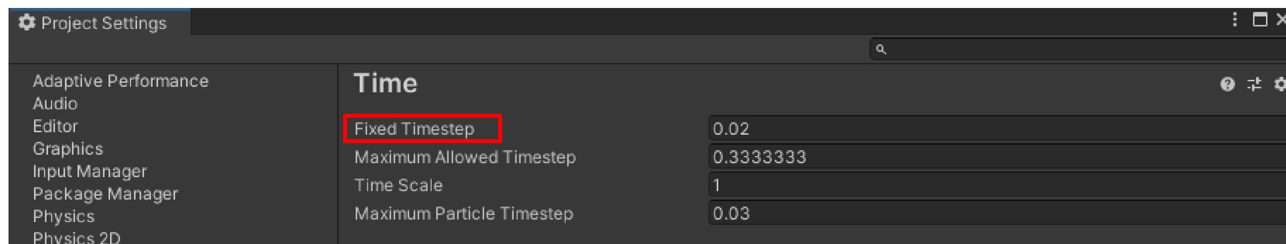

Рис. 19. Демонстрація значення поля **Fixed TimeStep.**

Позицію, яку ми отримуємо після лінійної інтерполяції ми присвоюємо положенню **PS**, за рахунок цього і моделюється рух **PS**, між точкою **А** та **Б**. Це все продовжується поки позиція **PS** не дорівнюватиме позиції **В**.

А далі ми змінюємо кінцеву точку **В** на наступну в списку. І цикл повторюється. І таким способом ми рухаємо наш генератор частинок від початкової аудиторії до кінцевої.

§8. Конструкція поверхів

Для того щоб розуміти як саме будується маршрут, де беруться координати опорних точок, між якими рухається генератор частинок – розберемо загальну будову об'єкта, який відіграє роль поверху на сцені де відбувається пошук маршруту між аудиторіями.

Так як за основу розробки інтерактивної карти був вибраний наш рідний факультет електроніки та комп'ютерних систем, розберемо типи поверхів які є в нашому рідному факультеті.

Ціллю дослідження обрано **10 поверхів:**

- **7-поверхів** лаб-корпусу ;
- **3-поверхи** аудиторної частини .

Тому на сцені, де відбувається пошук шляху, знаходиться 10 об'єктів, які відграють роль поверхів:

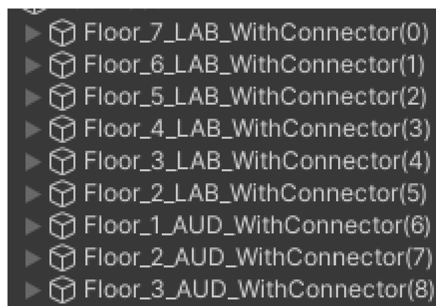

Рис. 20. Демонстрація об'єктів, які відграють роль поверхів.

Ці об'єкти представляють з себе моделі кожного з обраних поверхів.

Між кожними поверхами є сходи або ліфти, частини будівлі роль яких з'єднувати між собою ці поверхи.

Також більшість поверхів відрізняються своєю будовою.

Також, як не дивно, на кожному поверсі є свої аудиторії, які не повторюються на інших поверхах(***об`єкти імена яких є унікальними***).

Тому розберемо більш детальну конструкцію кожного поверху, а саме об'єкт, який відіграє роль поверху як взаємодію його **ChObj** об'єктів, з яких він і складається.

Отже, об'єкт який відіграє роль поверху(***надалі FloorObject***) являє собою **PObj**, якому підпорядковуються об'єкти, з яких складається поверх. Кожен **ChObj** об'єкта **FloorObject** має свою задачу, яка виконується при взаємодії з обраним поверхом якому вони належать.

Головною задачею **FloorObject** є відігравання ролі об'єкта, через який є можливість взаємодіяти з його **ChObj** за допомгою коду. Також при потребі перемістити чи повернути поверх, повертатись та переміщуватись буде саме **FloorObject**.

Далі піде детальний розбір одного поверху з частини лаб-корпусу та одного з аудиторної частини.

Пункт 8.1. Шостий поверх лаб-корпусу(_Floor_6_LAB_WithConnector(1)):

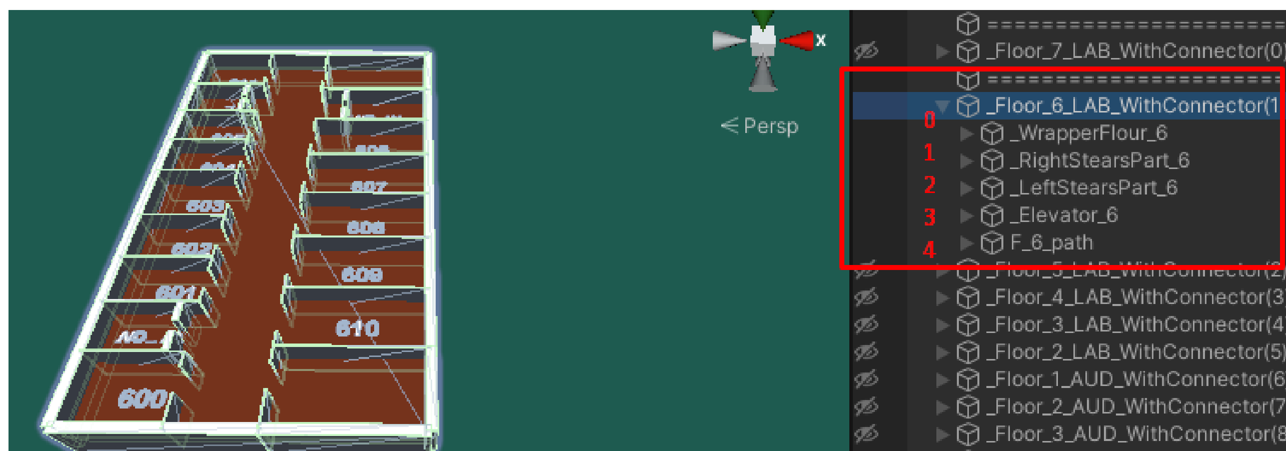

Рис. 21. Демонстрація ChObjects, з яких складається об'єкт шостого поверху.

Він складається з **5-ти** дочірніх об'єктів, які відіграють кожен свою важливу роль:

Назва дочірнього об\`єкту	Індекс який має **ChObj (child index)**
_WrapperFlour_6	0
_RightStearsPart_6	1
_LeftStearsPart_6	2
_Elevator_6	3
F_6_path	4

Таблиця 1. Список всіх дочірніх об'єктів яких складається об'єкт поверх.

Пункт 8.1.1. Об’єкт _WrapperFlour_6 (Обкладинка)

Обкладинка(wrapper) шостого поверху, фактично сама 3D модель поверху, створена з стандартних 3D компонентів движка. Мнемонічна назву об’єкт отримав, через властивість взаємодії **PObj** та **ChildObj.**

Всі моделі, які необхідно об’єднати та переміщувати, при цьому не втрачавши значення відстаней одне від одного. Все це дочірні об’єкти, які немов потрібно покрити обкладинкою, яка зафіксує їх положення. Тому PObj і був названий обкладенкою.

Сам цей дочірній об’єкт складається з купи стандартних моделей, і іноді щоб не допуститися помилки, варто перевіряти чи насправді позиція обкладинки знаходиться в геометричному центрі відносно всіх дочірніх об’єктів з яких складається

На Рис.18 було виділено декілька **ChObjects**, щоб показати, що обкладинка і дійсно складається з багатьох об’єктів в яких є своя позиція. Тому було прийнято рішення написати функцію, яка знаходить центр геометричної фігури, яка являє собою модель поверху.

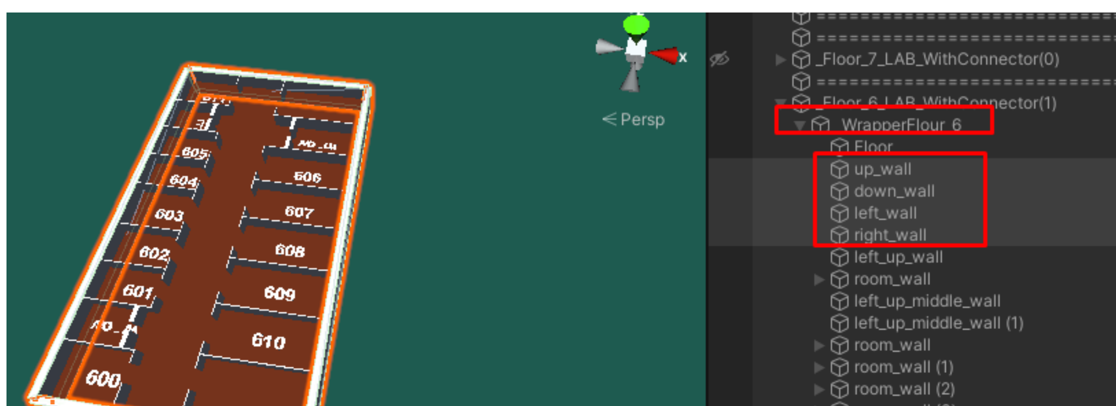

Рис. 22. Демонстрація виділених об’єктів, з яких складається об’єкт **_WrapperFlour_6_.**

Пункт 8.1.2. Алгоритм знаходження геометричного центру фігури.

```
private void FindCenreOfFloor()
{
    Transform tempWrapper = transform.GetChild(0);

    float maxXPosition = tempWrapper.GetChild(0).transform.position.x;
    float minXPosition = tempWrapper.GetChild(0).transform.position.x;
    float maxYPosition = tempWrapper.GetChild(0).transform.position.y;
    float minYPosition = tempWrapper.GetChild(0).transform.position.y;
    float maxZPosition = tempWrapper.GetChild(0).transform.position.z;
    float minZPosition = tempWrapper.GetChild(0).transform.position.z;

    for (
         int childDetailIndex = 0;
         childDetailIndex < tempWrapper.childCount;
         childDetailIndex++
        )
        {
          Vector3 tempFloorWrapper =
          tempWrapper.GetChild(childDetailIndex).transform.position;

          if (tempFloorWrapper.x > maxXPosition) // find Max X Position
          {
            maxXPosition = tempFloorWrapper.x;
          }

          if (tempFloorWrapper.x < minXPosition) // find Min X Position
          {
            minXPosition = tempFloorWrapper.x;
          }

          if (tempFloorWrapper.y > maxYPosition) // find Max Y Position
          {
            maxYPosition = tempFloorWrapper.y;
          }

          if (tempFloorWrapper.y < minYPosition) // find Min Y Position
          {
            minYPosition = tempFloorWrapper.y;
          }


         if (tempFloorWrapper.z > maxZPosition) // find Max Z Position
          {
            maxZPosition = tempFloorWrapper.z;
          }

          if (tempFloorWrapper.z < minZPosition) // find Min Z Position
          {
             minZPosition = tempFloorWrapper.z;
          }
    }


    float midPositionX = (minXPosition + maxXPosition) / 2.0f;
    float midPositionY = (maxYPosition + minYPosition) / 2.0f;
    float midPositionZ = (maxZPosition + minZPosition) / 2.0f;

    // X Red
    Debug.DrawLine(new Vector3(minXPosition, midPositionY, midPositionZ),
        new Vector3(maxXPosition, midPositionY, midPositionZ), Color.red);

    // Y Green
    Debug.DrawLine(new Vector3(midPositionX, minYPosition, midPositionZ),
        new Vector3(midPositionX, maxYPosition, midPositionZ), Color.green);

    // Z Blue
    Debug.DrawLine(new Vector3(midPositionX, midPositionY, minZPosition),
        new Vector3(midPositionX, midPositionY, maxZPosition), Color.blue);

    pointerObjetc1.transform.position =
    new Vector3(midPositionX, midPositionY, midPositionZ);
}
```

Для постійної наглядності ліній, які будуть перетинатись в геометричному центрі, ми будемо визивати цей метод кожний кадр:

```
private void Update()
{
    FindCenreOfFloor();
}
```

Даний скрипт ми переміщуємо на **PObj** для шостого поверху. В нашому випадку ім'я **PObj _Floor_6_LAB_WithConnector(1).**
Щоб отримати значення позицій з дочірніх компонентів **transform**, існує метод для компоненту transform :

```
GetChild(childIndex);
```

childIndex - індекс дочірнього компонента **transform** на повернення. Обов`язково має бути меншим за Transform.childCount(***кількість ChObjects для обраного вами об'єкту***).

Номер початкового індексу дорінює нулю.

Отже, скрипт знаходиться на об'єкті **_Floor_6_LAB_WithConnector(1).** Нам потрібно отримати доступ до компоненту **transform** об'єкту "обкладинки" (**_WrapperFlour_6),** який в свою чергу є **ChObj** з індексом нуль, для об'єку **_Floor_6_LAB_WithConnector(1).**

Тому ми зберігає посилання на об'єкт **_WrapperFlour_6**, за допомогою наступної команди.

```
Transform tempWarpper = transform.GetChild(0);
```

А далі нам необхідно лише пройтись по всім **ChObjects,** які присутні в обкладинці та для кожної з головних осей(X,Y,Z) – знайти об'єкт з мінімальним та максимальним положенням координат на кожній із осей.

Маючі ці дані досить просто дізнатися координати центру геометричної фігури, якою являє собою змодельований поверх.

Шукана позиція буде знаходитись за координатами **Xmid, Ymid, Zmid**, де:

$$\mathrm{Xmid} = \frac{\mathrm{Xmin} + \mathrm{Xmax}}{2}$$

$$\mathrm{Ymid} = \frac{\mathrm{Ymin} + \mathrm{Ymax}}{2}$$

$$\mathrm{Zmid} = \frac{\mathrm{Zmin} + \mathrm{Zmax}}{2}$$

Саме в цій позиції має знаходитись **PObj** для шостого поверху. Для того щоб показати, що іноді **PObj** не знаходиться в бажаній позиції, в потрібну координату я переміщуватиму позицію сфери. В скрипті за неї відповідає об'єкт **pointerObjetc1.** Іноді подібні помилки виникають через людський фактор розробника.

На Рис. 19 **червоним** показано де знаходиться **PObj** для всіх моделей, що використовуються в створенні 6 поверху. **Зеленим** положення сфери, координати якої відповідають координатам геометричного центру поверху в просторі.

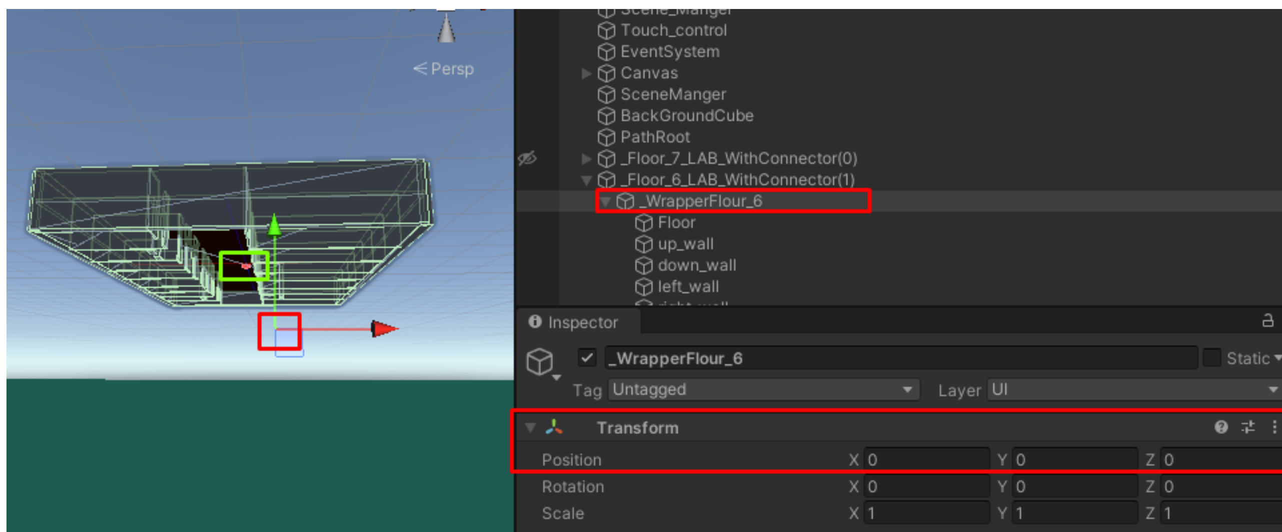

Рис. 23. Демонстрація невірної позиції об'єкта **_WrapperFlour_6**

У сфери немає **PObj.** Тому її координати вимірюються незалежно в світовому просторі. На Рис.19 вказуються відносні координати об'єкту обкладинки. Для порівняння перемістимо обкладинку на початковий рівень ієрархії, щоб в обкладинки був відсутній **PObj.** Тоді її координати також будуть вираховуватись в світовому просторі, а не відносно позиції **PObj.**

Для наглядності прибрав відображення підлоги(***об'єкт Floor***), щоб краще бачити перетин **синьої**(***з`єднує мінімальну та максимальну позицію об'єктів по осі Z*** ) та **червоної** (***з`єднує мінімальну та максимальну позицію об'єктів по осі X***) ліній.

На Рис.20 зображено координати сфери, яка вказує на геометричний центр моделі 6-ого поверху.

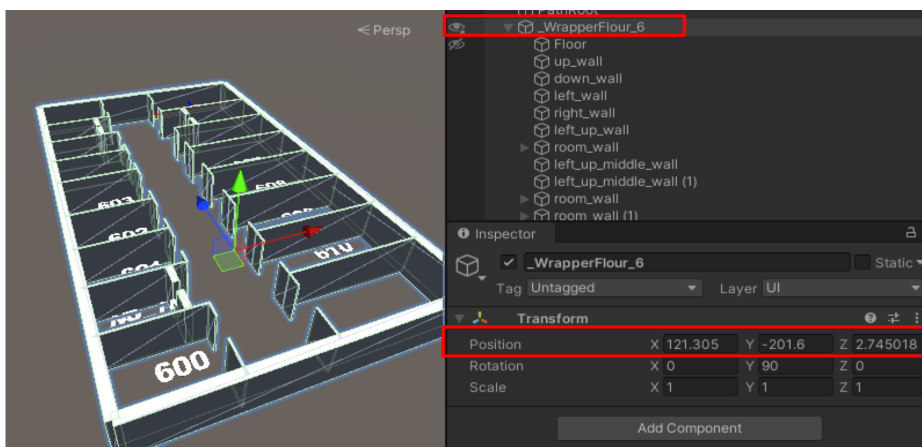

Рис. 24. Позиція в якій знаходиться **_WrapperFlour_6_.**

На Рис.21 зображено координати обкладинки, яка являє собою **PObj** для всіх елементів, що використовувались в моделюванні поверху.

Як видно з представлених рисунків, позиція обкладинки не збігається з потрібною позицією. Дану помилку досить легко виправити:

1. Необхідно винести всі **ChObjects,** з під об’єкту обкладинки.
2. Змінити позицію обкладинки на потрібну, вказавши дані у відповідні поля.
3. Повернути всі **ChObjects** в під-ієрархію об’єкту обкладинки, зробивши знову для всіх цих об’єктів обкладинку **PObj.**

Даний скрипт підходить для перевірки кожного з поверхів, якщо розробник продовжує слідкувати за послідовністю **ChObjects** в під-ієрархії **PObj,** який являє собою об’єкт поверху.

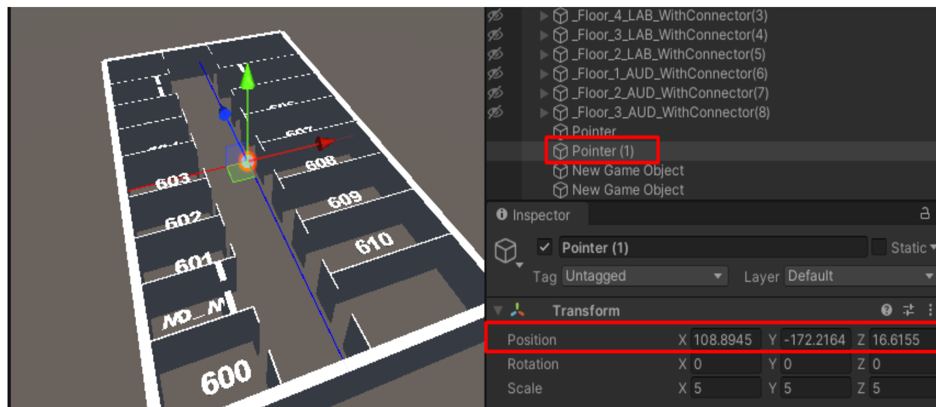

Рис. 25. Позиція в якій мав би знаходитись об’єкт **_WrapperFlour_6_.**

Пункт 8.1.3. Видимі об’єкти (visible components)

_RightStearsPart_6,_LeftStearsPart_6,_Elevator_6

(child index = 1) (child index = 2) (child index = 3)

Об’єкти які будуть відображатися для переходу з поверху на поверх , які відображаються, якщо поверх через який проходить маршрут не є кінцевим. Немає сенсу відображати весь 6 поверх, якщо маршрут починається на **7-ому** поверсі і закінчується на **5-ому.**

Достатньо лише відобразити правильний об’єкт переходу в залежності від обраного користувачем виду переходу :

- Найближчими сходами (**_RightStearsPart_6**, **_LeftStearsPart_6**)
- Ліфтом (**_Elevator_6**)

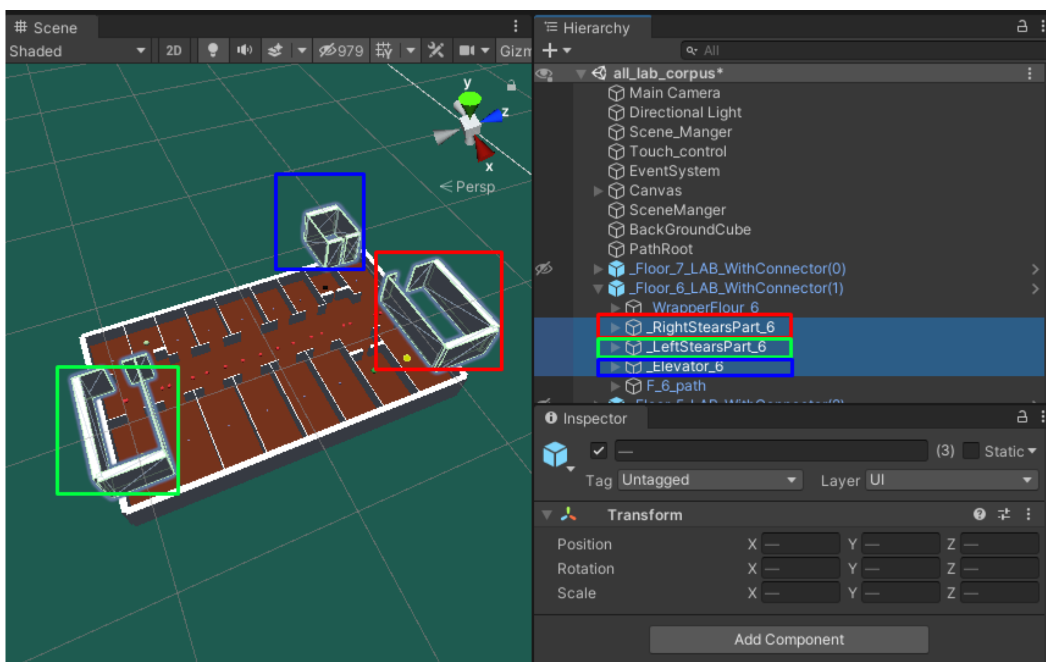

Рис.26. Демонстрація видимих об’єктів.

Для відображення видимих частин поверху в скрипті створений список з об’єктів, в який будуть додаватися потрібні частини поверху, які потрібно відображати.

```
public List<GameObject> visibleObjectList = new List<GameObject>();
```

Перед тим як буде викликатися функція, яка буде малювати траєкторію маршруту, буде виконуватись функція яка перевіряє всі **ChObjects** кожного **FloorObject** на потрібність відображення при побудові маршруту.

Немає сенсу відображати всі поверхи якщо маршрут покриває лише один поверх. Детальний алгоритм виконання всіх функцій буде розглянуто в главі … На даний момент потрібно лише зрозуміти навіщо ці об’єкти були створенні.

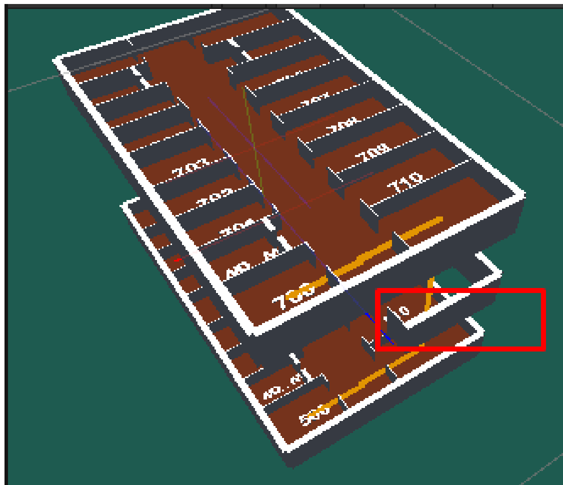

Рис. 27. Використовуємо **visible objects.**

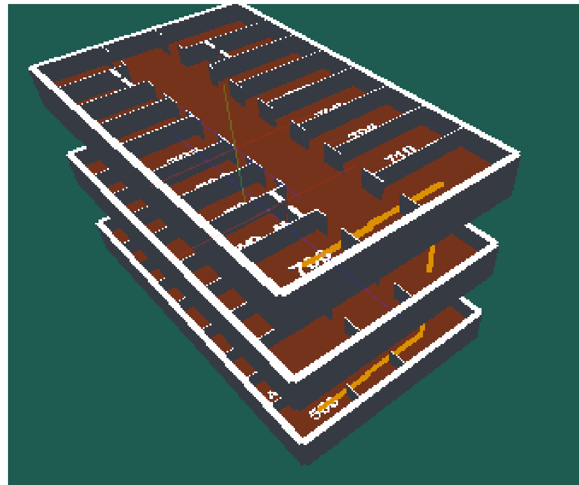

Рис. 28. Відсутнє використання **visible objects.**

Для порівняння два рисунки Рис.23 та Рис.24.

На Рис.27 маршрут будується із використанням **visible components.**

На Рис.28 маршрут будується без використанням **visible components.**

З наведених рисунків робимо висновок: використання таких компонентів як _RightStearsPart_6,_LeftStearsPart_6,_Elevator_6 є доцільним.

Приклад коду де демонструється додавання видимих компонентів поверху:

```
private string BuildPathPassesThroughTheLF6(string inputStairsOrElevator)
{
    Transform pathObjectOfLF6 =
  pathRoot.transform.GetChild(1).GetChild(pathRoot.transform.GetChild(1).childCount - 1);

    string[] checkPartInput = inputStairsOrElevator.Split('_');

    if (checkPartInput[0] == "LeftStairs")
    {
        pathListOfCoordinates.Add(pathObjectOfLF6.GetChild(0).GetChild(0));
        //================Add Visible Part================
        visibleObjectList.Add(pathRoot.transform.GetChild(1).GetChild(2).gameObject);
        //================Add Visible Part================
    }
    else if (checkPartInput[0] == "RightStairs")
    {
        pathListOfCoordinates.Add(pathObjectOfLF6.GetChild(16).GetChild(0));
        //================Add Visible Part================
        visibleObjectList.Add(pathRoot.transform.GetChild(1).GetChild(1).gameObject);
        //================Add Visible Part================
    }
    else if (checkPartInput[0] == "E")
    {
        pathListOfCoordinates.Add(pathObjectOfLF6.GetChild(14).GetChild(0));
        //================Add Visible Part================
        visibleObjectList.Add(pathRoot.transform.GetChild(1).GetChild(3).gameObject);
        //================Add Visible Part================
    }

    return pathListOfCoordinates.Last().name + "_L6";
}
```

Пункт 8.1.4. Дерево опорних точок (F_6_path)

Дерево опорних точок – PObj в якому містяться всі опорні точки, через які і будується маршрут яким буде проходити користувач.

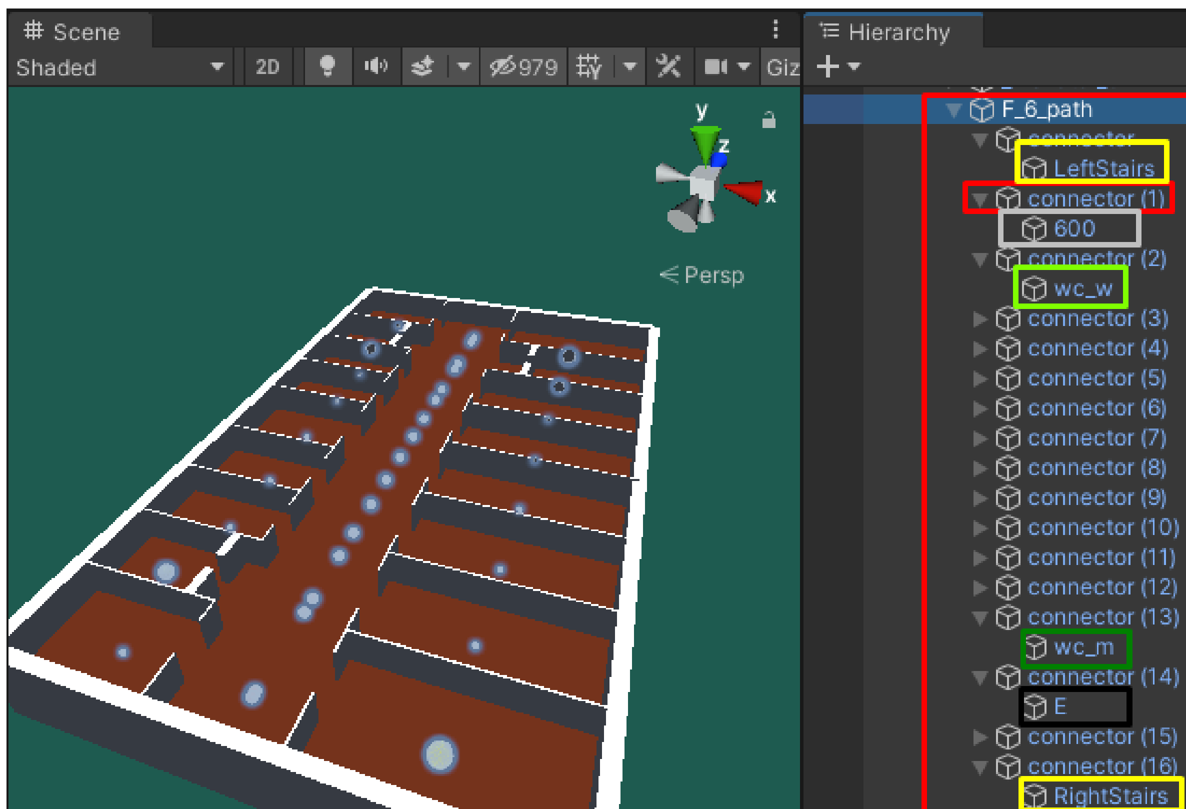

Рис. 29. Демонстрація ієрархії дерева опорних точок та його **ChObjects.**

Розберемо детально, кожен об`єкт, який знаходиться в даному дереві.

Для великої кількості об'єктів, які потрібно розбити на під сімейства об'єктів які відіграть свої задачі існує такий інструмент як тег(**Tag**).

Тег – це опорне слово, яке ви можете призначити одному або кільком об'єктам гри. Наприклад, ви можете визначити теги «**Player**» для персонажів, керованих гравцем, і тег «**Enemy**» для персонажів, які не керуються гравцем. Ви можете визначити предмети, які гравець може збирати в сцені з тегом «**Колекційний**».

Таким способом можна полегшити написання коду. Є досить корисний метод для пошуку об'єктів за окремим тегом. Так якщо вам знадобиться знайти позиції об'єктів з окремим тегом, вам потрібно лише використати команду:

GameObject.FindGameObjectsWithTag(**tag**)

Пункт 8.1.4.1. connector (1), connector (4) (tag:connector)

опорні точки, роль яких бути проміжними точками при побудові маршруту на поверсі.

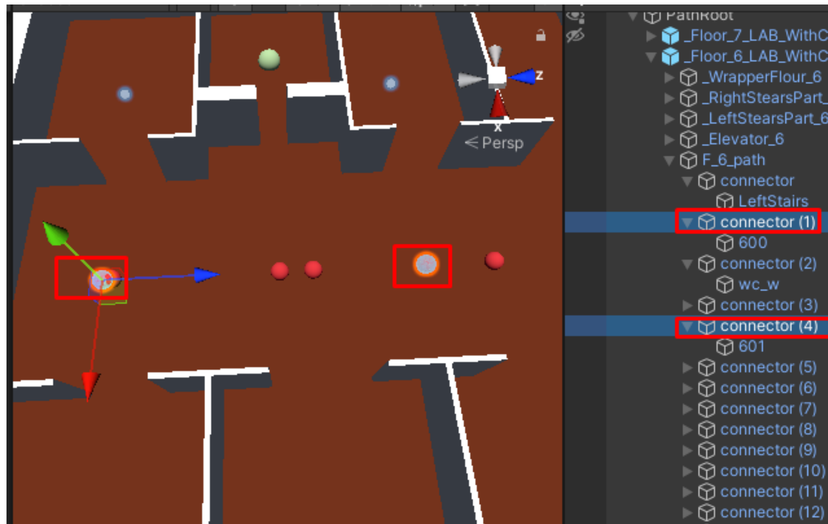

Рис.30

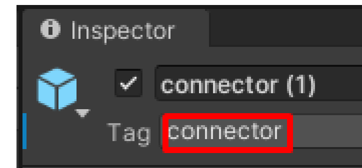

Рис.31

Також використовуються для реалізації повороту маршруту.

Являються **PObj** для таких опорних точок як:

- Аудиторії **(tag:room)**
- Ліфти **(tag:elevator)**
- Жіночі туалети **(tag:wc_w)**
- Чоловічі туалети **(tag:wc_m)**
- Сходи **(tag:Stairs)**

Завдяки ним, також досить зручно звертатись то компонентів transform усіх кімнат на поверсі.

Пункт 8.1.4.2. LeftStairs, RightStairs (tag:Stairs)

Опорні точки, роль яких є з'єднувати маршрут між поверхами, якщо користувач обрав спосіб переміщення сходи.

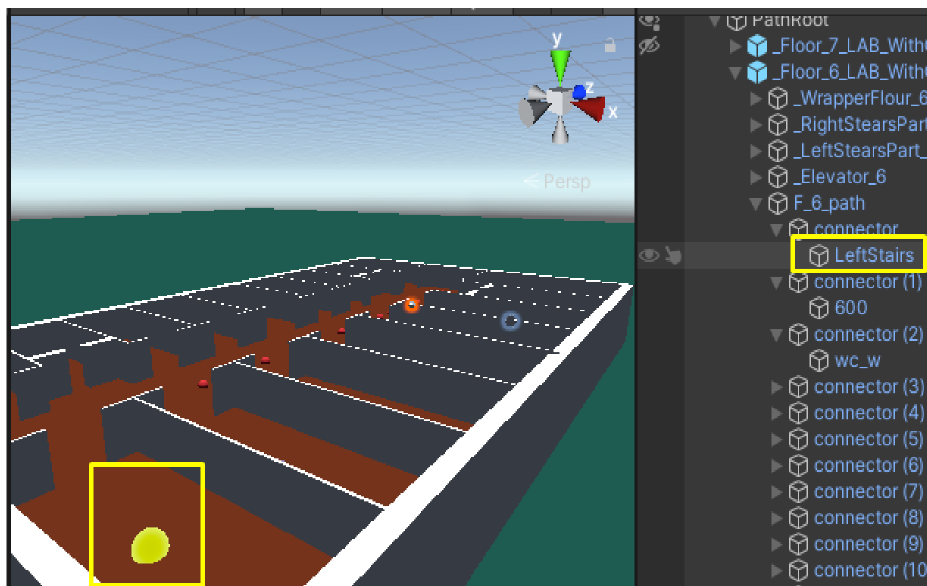

Рис.32

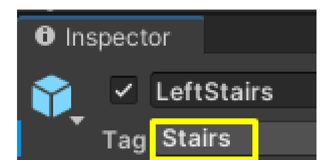

Рис.33

Існують як у підмножині поверхів аудиторних та поверхів лаб-корпусу.

Саме з цих точок буде будуватися шлях на проміжних поверхах та кінцевих поверхах.(***Про ролі поверхів більш детально розглянута інформація в частині***)

Пункт 8.1.4.3. 600 (tag:room)

опорні точки, роль яких бути початковими та кінцевими опорними точками при побудові маршруту.

Рис.34

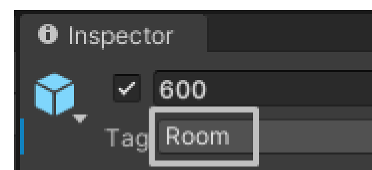

Рис.35

Саме пошук позиції аудиторій і полягає основна мета додатку, яка спростить життя як і студенту так і викладачу.

Пункт 8.1.4.4. wc_w (tag:wc_w)

Опорні точки, роль яких **бути лише кінцевими** опорними точками при побудові маршруту. Подібні до аудиторій, однак дані кімнати є на поверсі в єдиному екземплярі.

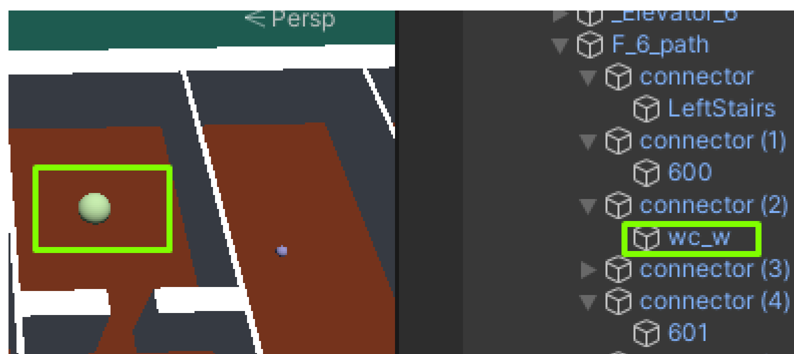

Рис.36

Рис.37

Пункт 8.1.4.5. wc_m (tag:wc_m)

опорні точки, роль яких **бути лише кінцевими** опорними точками при побудові маршруту. Подібні до аудиторій, однак дані кімнати є на поверсі в єдиному екземплярі.

Рис.38

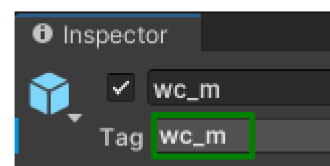

Рис.39

Пункт 8.1.4.6. E (tag:Elevator)

опорні точки, роль яких є з'єднувати маршрут між поверхами, якщо користувач обрав спосіб переміщення ліфт.

Рис.40

Рис.41

Пункт 8.2. Третій поверх аудиторного корпусу (_Floor_3_AUD_WithConnector(8))

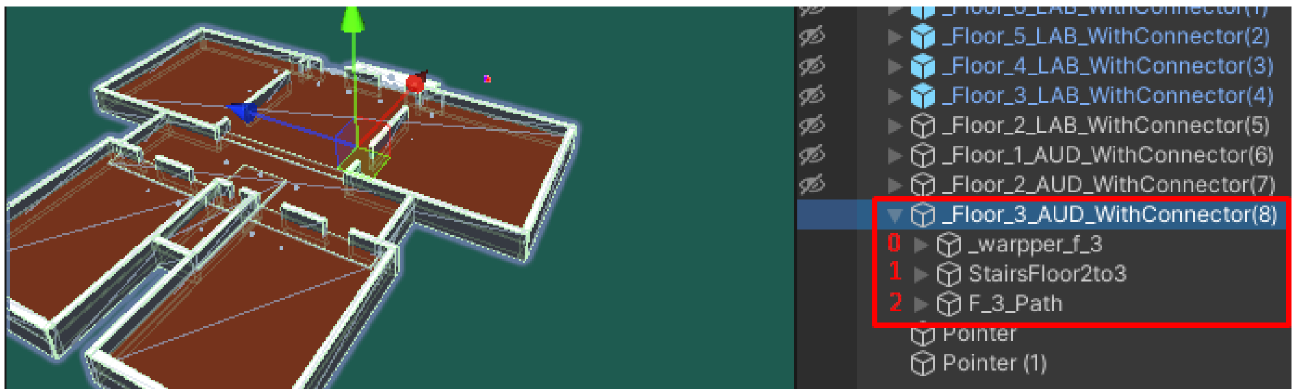

Рис.42

Він складається з **3-ох** дочірніх об`єктів. Їх ролі подібні до ролей дочірних об`єктів поверхів з лаб-корпусу. Однак єдина відмінність в тому, що менше видимих компонентів.

Назва дочірнього об`єкту	Індекс який має **ChObj (child index)**
_warpper_f_3	0
StairsFloor2to3	1
F_3_Path	2

Wrapper_f_3 – обкладинка, по суті модель самого поверху

StairsFloor2to3 – видимий компонент, який буде відображатися при переході між 2 та 3 поверхами.

F_3_Path – дерево з опорними точками.

Дякуючи такій структурі дочірніх об'єктів у всіх поверхах, задача з відображенням та поворотами вже не є такою складною.

Отже на далі розберемо алгоритм, за яким будується маршрут з опорних точок. Для раціонального пошуку створено ще один об`єкт, який являє собою роль **PObj**, для всіх об`єктів, що являють собою поверхи. Дякую чому ми з легкістю зможемо отримувати потрібні нам компоненти з яких складається поверх для того щоб відображати потрібну частину поверху. Також це дозволить спростити пошук опорних точок, через який буде проходити маршрут.

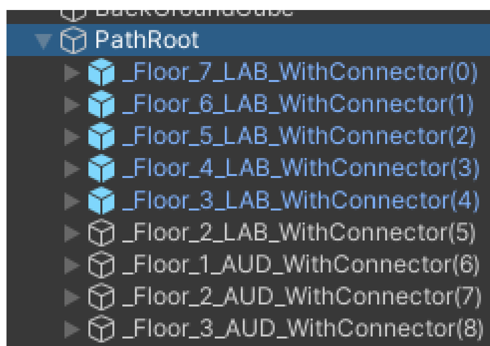

Рис.43

§9. Декомпозиція пошуку маршруту між початковою та кінцевою аудиторіями

Для раціонального пошуку аудиторій розберемо задачу пошуку на окремі підзадачі.

1. Пошук початкового поверху.
2. Пошук кінцевого поверху.
3. Пошук проміжних ц поверхів.

Розглянемо результат виконання 1-ої та 3-ої підзадачі;

Замість того, щоб кожного разу пробігатися по всім поверхам, з метою пошуку кінцевого поверху, використовуючи даний алгоритм дій, можна зменшити непотрібну кількість пошуків. Опираючись на те, що ми можемо розбити поверхи на дві підмножини, і в результаті в залеєності до яких підмножин відноситься початковий та кінцевий поверхи, зможемо первіряти лише потрібну нам частину поверхів.

Множину всіх поверхів можна розбити на дві підмножини:

- Підмножина аудиторних поверхів (AUD).
- Підмножина лабораторних поверхів (LAB).

Таблиця істинності для всіх можливих ситуацій при пошуку початкової та кінцевої аудиторій:

Підмножина початкової аудиторії	Підмножина кінцевої аудиторії	Висновок
AUD	AUD	непотрібно відображати лабораторні поверхи
AUD	LAB	присутній перехід між підмножинами
LAB	AUD	присутній перехід між підмножинами
LAB	LAB	непотрібно відображати аудиторні поверхи

Маючи ці дані, можна стоврити граф переходів між підмножинами. А вже виходячи з графу побудувати правильні умови при пошуку маршруту між початковим та кінцевиповерхами.

Таке розбиття економить час для пошуку шляху між аудиторіями. Викорстання розбиття на підмножини в функції з побудови маршруту.

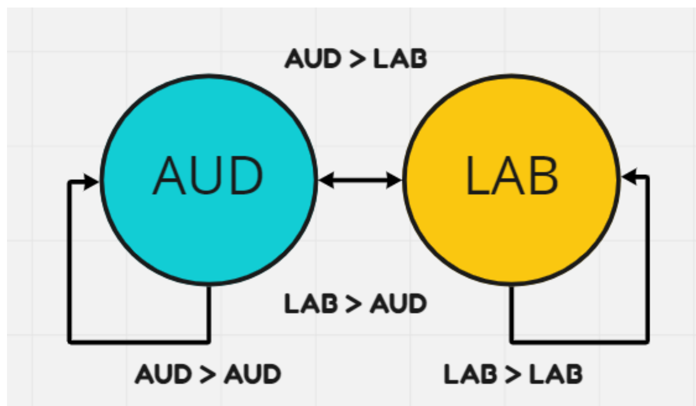

Рис. 44. Граф переходів між множинами під час пошуку початкової та кінцевої аудиторії

```
Frequently called   1 usage   Xergofian *
private List<Transform> FindPathRoomToRoom(string startRoomInputText, string endRoomInputText)
{
    bool sIntFlag = int.TryParse(startRoomInputText, out int startRoomNumberInt);
    bool eIntFlag = int.TryParse(endRoomInputText, out int endRoomNumberInt);

    char[] numbersInStartRoomNumber;
    char[] numbersInEndRoomNumber;

    if (startRoomInputText.Length == 1){...}
    else{...}

    if (endRoomInputText.Length == 1){...}
    else{...}

    int startRoomNumber = Convert.ToInt32(numbersInStartRoomNumber[0].ToString());
    int endRoomNumber = Convert.ToInt32(numbersInEndRoomNumber[0].ToString());

    string inputToAF3 = "";
    string inputToAF2 = "";
    string inputToAF1 = "";
    string inputToLF2 = "";
    string inputToLF3 = "";
    string inputToLF4 = "";
    string inputToLF5 = "";
    string inputToLF6 = "";
    string inputToLF7 = "";

    if (sIntFlag && eIntFlag)
    {
        if (startRoomInputText.Length <= 2 && endRoomInputText.Length <= 2) // Start in AUD <==> End in AUD{...}
        else if (startRoomInputText.Length <= 2 &&
                 endRoomInputText.Length > 2) // Start Room in AUD <==> End Room in LAB {...}
        else if (startRoomInputText.Length > 2 &&
                 endRoomInputText.Length <= 2) //Start Room in LAB <==> End Room in AUD  {...}
        else if (startRoomInputText.Length > 2 &&
                 endRoomInputText.Length > 2) //Start Room in LAB <==> End Room in LAB {...}
    }
    return pathListOfCoordinates;
}
```

Тепер, розберемо більш детально всі можливі випадки побудови шляху через кожний поверх.

1. Кожен поверх може бути як початковим так і кінцевим .
2. Через кожен поверх може проходити маршрут, і при цьому поверх не є початковим так і кінцевим. Це твердження не є вірним для 3 поверху аудиторної підмножини та 7 поверху лабораторної підмножини. Дані поверхи можуть бути лише початковими або кінцевими.
3. Маршрут може починатися і закінчуватися не виходячи за межі одного поверху.

Щоб відповідати всім цим твердженням для поверхів лабораторної підмножини було створенно такі методи(***розбір іде знову лише на прикладі 6-того поверху, так як лабораторні поверхи ідентичні в плані компонента – дерева опорних точок***)

Кожен з наступних методів, створений для кожного поверху відповідно:

FindStartRoomOnTheLF6AndPathToAnotherFloorWithStairsR() - для пошуку початкового поверху та пошуку маршруту з початкового поверху до наступного поверху, використовуючи сходи.

FindStartRoomOnTheLF6AndPathToAnotherFloorWithElevatorR() – для пошуку початкового поверху та пошуку маршруту з початкового поверху до наступного поверху, використовуючи ліфт.

FindEndRoomOnTheLF6() – для пошуку кінцевої кімнати на шостому поверху.

FindStartAndEndRoomOnTheLF6() – для пошуку маршруту між початковою та кінцевою кімнатами, що знаходяться на шостому поверсі.

BuildPathPassesThroughTheLF6() – для побудови маршруту через видимі компоненти, коли шостий поверх є проміжний. Даний метод відсутній для сьомого поверху із підмножини **LAB** та третього поверху підмножини **AUD**.

Щоб відповідати всім цим твердженням для поверхів аудиторної підмножини було створенно такі методи(***розбір іде лише на прикладі 2-ого поверху, так як аудиторні поверхи ідентичні в плані компонента – дерева опорних точок***)

FindStartRoomOnTheAF2AndPathToAnotherFloor() - для пошуку початкового поверху та пошуку маршруту з початкового поверху до наступного поверху, використовуючи сходи так як ліфти відсутні в даній підмножині.

FindStartAndEndRoomOnTheAF2() - для пошуку маршруту між початковою та кінцевою кімнатами, що знаходяться на 2-гому поверсі.

FindEndRoomOnTheAF2() - для пошуку кінцевої кімнати на другому поверсі.

BuildPathPassesThroughTheAF2() - для побудови маршруту через видимі компоненти, коли другий поверх є проміжний.

Перед розглядом загального алгоритму з пошуку маршруту розглянемо більш детально кожен з методів.

Пункт 9.1 FindStartRoomOnTheLF6AndPathToAnotherFloorWithStairsR()

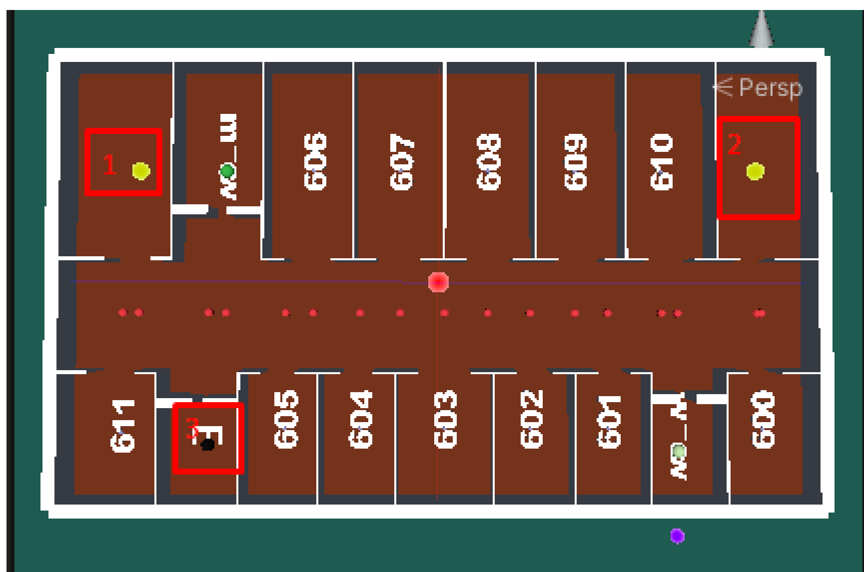

Рис.45

В залежності, від позиції початкової кімнати ми мусимо знайти шлях до найближчих сходів. Для даної реалізації використовувались плани пожежної безпеки, які знаходяться на факультеті.

Рис.46

Тобто, якщо номер початкової аудиорії знаходиться серед списку :

[**600, 601, 602, 608, 609, 610**]

Будуть додаватися координати сходів, які позначені одиницею на Рис.42

[**603,604,605, 606,607,611]**

Будуть додаватися координати сходів, які позначені двійкою на Рис.42

Демонстрація коду:

```
private string FindStartRoomOnTheLF6AndPathToAnotherFloorWithStairsR(string
startRoomName)
{
    Transform pathObjectOnL6F =

pathRoot.transform.GetChild(1).GetChild(pathRoot.transform.GetChild(1).childCoun
t - 1);

    for (int i = 0; i < pathObjectOnL6F.transform.childCount; i++)
    {
        // проходимся по всем конекторам.
        if (pathObjectOnL6F.GetChild(i).childCount != 0)
        {
            for (int child = 0; child <
pathObjectOnL6F.transform.GetChild(i).childCount; child++)
            {
                string tempRoomName =
pathObjectOnL6F.transform.GetChild(i).GetChild(child).name;

                if (tempRoomName == startRoomName)
                {

pathListOfCoordinates.Add(pathObjectOnL6F.transform.GetChild(i).GetChild(child))
;

pathListOfCoordinates.Add(pathObjectOnL6F.transform.GetChild(i));
                }
            }
        }
    }

    if (CheckLeftOrRightStairs("6", startRoomName) == "Left")
    {
        pathListOfCoordinates.Add(pathObjectOnL6F.GetChild(0));
        pathListOfCoordinates.Add(pathObjectOnL6F.GetChild(0).GetChild(0));
    }
    else
    {
        pathListOfCoordinates.Add(pathObjectOnL6F.GetChild(16));
        pathListOfCoordinates.Add(pathObjectOnL6F.GetChild(16).GetChild(0));
    }

    //================Add Visible Part================

visibleObjectList.Add(pathRoot.transform.GetChild(1).GetChild(0).gameObject);

visibleObjectList.Add(pathRoot.transform.GetChild(1).GetChild(pathRoot.transform
.GetChild(1).childCount - 1)
        .gameObject);
```

```
    //================Add Visible Part================

    return pathListOfCoordinates.Last().name + "_L6";
}
```

Перевірка чи потрібно іти до лівих сходів чи правих:

```
private string CheckLeftOrRightStairs(string floorLabNumber, string
startRoomName)
{
    const string startRangeLeftPartOfUpRoom = "00";
    const string endRangeLeftPartOfUpRoom = "02";
    const string startRangeLeftPartOfDownRoom = "08";
    const string endRangeLeftPartOfDownRoom = "10";

    int startRoomInt = Convert.ToInt32(startRoomName);

    int startLeftRangeUpRoom = Convert.ToInt32(floorLabNumber +
startRangeLeftPartOfUpRoom);
    int endLeftRangeUpRoom = Convert.ToInt32(floorLabNumber +
endRangeLeftPartOfUpRoom);
    int startLeftRangeDownRoom = Convert.ToInt32(floorLabNumber +
startRangeLeftPartOfDownRoom);
    int endLeftRangeDownRoom = Convert.ToInt32(floorLabNumber +
endRangeLeftPartOfDownRoom);


    if ((startLeftRangeUpRoom <= startRoomInt && startRoomInt <=
endLeftRangeUpRoom) ||
        (startLeftRangeDownRoom <= startRoomInt && startRoomInt <=
endLeftRangeDownRoom))
    {
        return "Left";
    }
    else
    {
        return "Right";
    }
}
```

Для пошуку на інших поверхах, потрібно робити персональні методи. Для цього розберемо детально метод **FindStartAndEndRoomOnTheAF2()** та будову 2 аудиторного поверху**.**

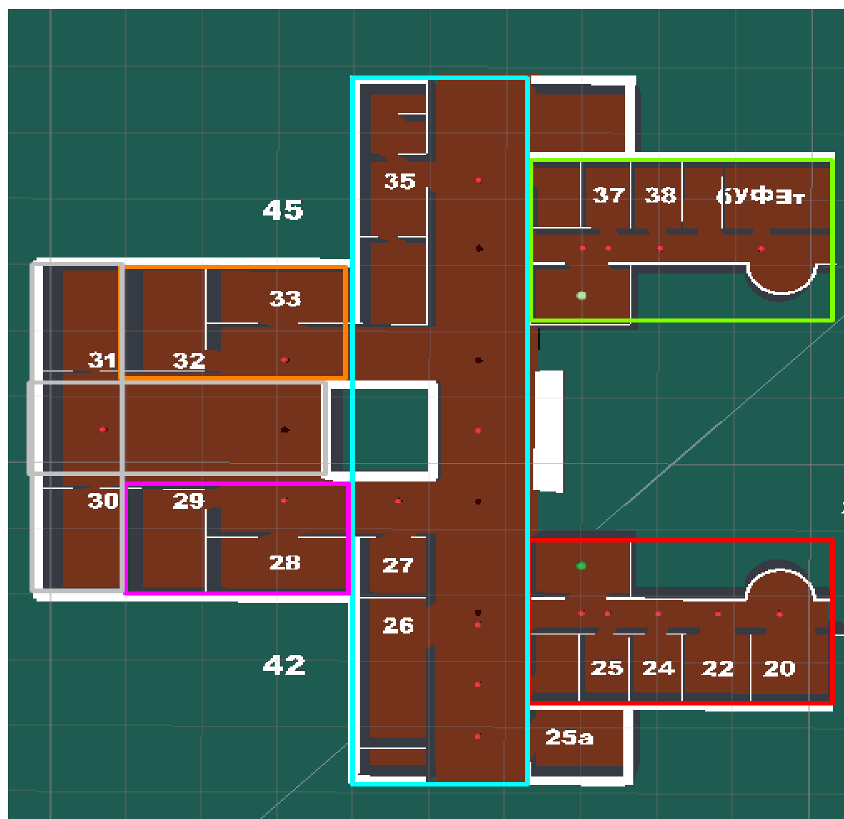

Рис.47

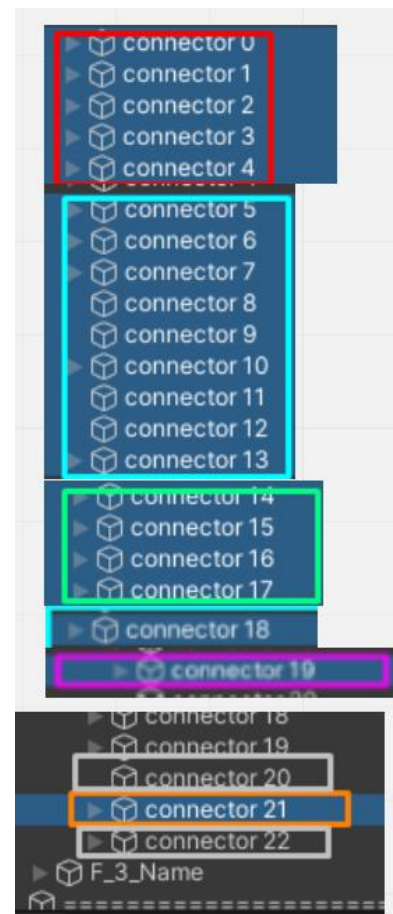

Рис.48

Розбиваємо поверх на секції з кімнатами для яких потрібно будувати подібні маршрути для їх сполучення до основного шляху.

Тому основна задача метода полягає в трьох етапах:

1. Знайти координати початкової кімнати, та конектора, що його з'єднує з деревом конекторів.
2. Додати проміжні точки між секціями з яких складається поверх до маршруту.
3. Додати конектор кінцевого поверху та кінцеву кімнату.

Демонстрація виконання даного алгоритму:

```
private void FindStartAndEndRoomOnTheAF2(string startRoomName, string endRoomName)
{
    int indexOfConnectorStartRoom = 0;
    int indexOfConnectorEndRoom = 0;
    int indexOfChildInConnectorEndRoom = 0;

    Transform floorPathObject =
        pathRoot.transform.GetChild(7).GetChild(pathRoot.transform.GetChild(7).childCount - 1);

    for (int i = 0; i < floorPathObject.transform.childCount; i++)
    {
        if (floorPathObject.GetChild(i).childCount != 0)
        {
            for (int child = 0; child < floorPathObject.transform.GetChild(i).childCount; child++)
            {
                string tempRoomName = floorPathObject.transform.GetChild(i).GetChild(child).name;
```

```
                if (tempRoomName == startRoomName)
                {
                    indexOfConnectorStartRoom = i;

pathListOfCoordinates.Add(floorPathObject.transform.GetChild(i).GetChild(child));
                    pathListOfCoordinates.Add(floorPathObject.transform.GetChild(i));
                }

                if (tempRoomName == endRoomName)
                {
                    indexOfConnectorEndRoom = i;
                    indexOfChildInConnectorEndRoom = child;
                }
            }
        }
    }

    if (0 <= indexOfConnectorStartRoom && indexOfConnectorStartRoom <= 4) // StartRoom Red
    {
        if (0 <= indexOfConnectorEndRoom && indexOfConnectorEndRoom <= 4) // StartRoom Red <==>
EndRoom Red
        {
        }
        else
        {
            pathListOfCoordinates.Add(floorPathObject.GetChild(8));

            if (5 <= indexOfConnectorEndRoom && indexOfConnectorEndRoom <= 13 ||
                indexOfConnectorEndRoom == 18) // StartRoom Red <==> EndRoom Blue
            {
                if (indexOfConnectorEndRoom == 18)
                {
                    pathListOfCoordinates.Add(floorPathObject.GetChild(9));
                }
            }
            else if (14 <= indexOfConnectorEndRoom &&
                     indexOfConnectorEndRoom <= 17) // StartRoom Red <==> EndRoom Green
            {
                pathListOfCoordinates.Add(floorPathObject.GetChild(12));
            }
            else if (indexOfConnectorEndRoom == 21) // StartRoom Red <==> EndRoom Orange
            {
                pathListOfCoordinates.Add(floorPathObject.GetChild(11));
            }
            else if (indexOfConnectorEndRoom == 19) //StartRoom Red <==> EndRoom Pink
            {
                pathListOfCoordinates.Add(floorPathObject.GetChild(9));
            }
            else if (indexOfConnectorEndRoom == 22) // StartRoom Red <==> EndRoom Grey
            {
                pathListOfCoordinates.Add(floorPathObject.GetChild(9));
                pathListOfCoordinates.Add(floorPathObject.GetChild(19));
                pathListOfCoordinates.Add(floorPathObject.GetChild(20));
            }
        }
    }
    else if (5 <= indexOfConnectorStartRoom && indexOfConnectorStartRoom <= 13 ||
             indexOfConnectorStartRoom == 18) // StartRoom Blue
    {
        if (0 <= indexOfConnectorEndRoom && indexOfConnectorEndRoom <= 4) // StartRoom Blue <==>
EndRoom Red
        {
            if (indexOfConnectorStartRoom == 18)
            {
                pathListOfCoordinates.Add(floorPathObject.GetChild(9));
            }

            pathListOfCoordinates.Add(floorPathObject.GetChild(8));
        }
        else if (5 <= indexOfConnectorEndRoom && indexOfConnectorEndRoom <= 13 ||
                 indexOfConnectorEndRoom == 18) // StartRoom Blue <==> EndRoom Blue
        {
            if (indexOfConnectorStartRoom == 18)
            {
                pathListOfCoordinates.Add(floorPathObject.GetChild(9));
            }
        }
        else if (14 <= indexOfConnectorEndRoom &&
                 indexOfConnectorEndRoom <= 17) // StartRoom Blue <==> EndRoom Green
        {
```

```
            if (indexOfConnectorStartRoom == 18)
            {
                pathListOfCoordinates.Add(floorPathObject.GetChild(9));
            }

            pathListOfCoordinates.Add(floorPathObject.GetChild(12));
        }
        else if (indexOfConnectorEndRoom == 21) // StartRoom Blue <==> EndRoom Orange
        {
            if (indexOfConnectorStartRoom == 5 || indexOfConnectorStartRoom == 7 ||
indexOfConnectorStartRoom == 18)
            {
                if (indexOfConnectorStartRoom == 5 || indexOfConnectorStartRoom == 7)
                {
                    pathListOfCoordinates.Add(floorPathObject.GetChild(9));
                }

                pathListOfCoordinates.Add(floorPathObject.GetChild(19));
            }
            else
            {
                pathListOfCoordinates.Add(floorPathObject.GetChild(11));
            }
        }
        else if (indexOfConnectorEndRoom == 19) //StartRoom Blue <==> EndRoom Pink
        {
            if (indexOfConnectorStartRoom == 5 || indexOfConnectorStartRoom == 7 ||
indexOfConnectorStartRoom == 18)
            {
                if (indexOfConnectorStartRoom == 5 || indexOfConnectorStartRoom == 7)
                {
                    pathListOfCoordinates.Add(floorPathObject.GetChild(9));
                }

                pathListOfCoordinates.Add(floorPathObject.GetChild(19));
            }
            else
            {
                pathListOfCoordinates.Add(floorPathObject.GetChild(11));
                pathListOfCoordinates.Add(floorPathObject.GetChild(21));
            }
        }
        else if (indexOfConnectorEndRoom == 22) // StartRoom Blue <==> EndRoom Grey
        {
            if (indexOfConnectorStartRoom == 5 || indexOfConnectorStartRoom == 7 ||
indexOfConnectorStartRoom == 18)
            {
                if (indexOfConnectorStartRoom == 5 || indexOfConnectorStartRoom == 7)
                {
                    pathListOfCoordinates.Add(floorPathObject.GetChild(9));
                }

                pathListOfCoordinates.Add(floorPathObject.GetChild(19));
            }
            else
            {
                pathListOfCoordinates.Add(floorPathObject.GetChild(11));
                pathListOfCoordinates.Add(floorPathObject.GetChild(21));
            }

            pathListOfCoordinates.Add(floorPathObject.GetChild(20));
        }
    }
    else if (14 <= indexOfConnectorStartRoom && indexOfConnectorStartRoom <= 17)
    {
        if (14 <= indexOfConnectorEndRoom && indexOfConnectorEndRoom <= 17) // StartRoom Green <==>
EndRoom Green
        {
        }
        else
        {
            pathListOfCoordinates.Add(floorPathObject.GetChild(12));

            if (0 <= indexOfConnectorEndRoom && indexOfConnectorEndRoom <= 4) // StartRoom Green
<==> EndRoom Red
            {
                pathListOfCoordinates.Add(floorPathObject.GetChild(8));
            }
            else if (5 <= indexOfConnectorEndRoom && indexOfConnectorEndRoom <= 13 ||
                     indexOfConnectorEndRoom == 18) // StartRoom Green <==> EndRoom Blue
```

```
            {
                if (indexOfConnectorEndRoom == 18)
                {
                    pathListOfCoordinates.Add(floorPathObject.GetChild(9));
                }
            }
            else if (indexOfConnectorEndRoom == 21) // StartRoom Green <==> EndRoom Orange
            {
                pathListOfCoordinates.Add(floorPathObject.GetChild(11));
            }
            else if (indexOfConnectorEndRoom == 19) //StartRoom Green <==> EndRoom Pink
            {
                pathListOfCoordinates.Add(floorPathObject.GetChild(11));
                pathListOfCoordinates.Add(floorPathObject.GetChild(21));
            }
            else if (indexOfConnectorEndRoom == 22) // StartRoom Green <==> EndRoom Grey
            {
                pathListOfCoordinates.Add(floorPathObject.GetChild(11));
                pathListOfCoordinates.Add(floorPathObject.GetChild(21));
                pathListOfCoordinates.Add(floorPathObject.GetChild(20));
            }
        }
    }
    else if (indexOfConnectorStartRoom == 21) // StartRoom Orange
    {
        if (0 <= indexOfConnectorEndRoom && indexOfConnectorEndRoom <= 4) // StartRoom Orange <==>
EndRoom Red
        {
            pathListOfCoordinates.Add(floorPathObject.GetChild(19));
            pathListOfCoordinates.Add(floorPathObject.GetChild(9));
            pathListOfCoordinates.Add(floorPathObject.GetChild(8));
        }
        else if (5 <= indexOfConnectorEndRoom && indexOfConnectorEndRoom <= 13 ||
                 indexOfConnectorEndRoom == 18) // StartRoom Orange <==> EndRoom Blue
        {
            if (indexOfConnectorEndRoom == 5 || indexOfConnectorEndRoom == 7 ||
indexOfConnectorEndRoom == 18)
            {
                pathListOfCoordinates.Add(floorPathObject.GetChild(19));

                if (indexOfConnectorEndRoom == 5 || indexOfConnectorEndRoom == 7)
                {
                    pathListOfCoordinates.Add(floorPathObject.GetChild(9));
                }
            }
            else
            {
                pathListOfCoordinates.Add(floorPathObject.GetChild(11));
            }
        }
        else if (14 <= indexOfConnectorEndRoom &&
                 indexOfConnectorEndRoom <= 17) // StartRoom Orange <==> EndRoom Green
        {
            pathListOfCoordinates.Add(floorPathObject.GetChild(11));
            pathListOfCoordinates.Add(floorPathObject.GetChild(12));
        }
        else if
            (indexOfConnectorEndRoom == 21 ||
             indexOfConnectorEndRoom ==
             19) // StartRoom Orange <==> EndRoom Orange || StartRoom Orange <==> EndRoom Pink
        {
        }
        else if (indexOfConnectorEndRoom == 22) // StartRoom Orange <==> EndRoom Grey
        {
            pathListOfCoordinates.Add(floorPathObject.GetChild(20));
        }
    }
    else if (indexOfConnectorStartRoom == 19) // StartRoom Pink
    {
        if (0 <= indexOfConnectorEndRoom && indexOfConnectorEndRoom <= 4) // StartRoom Pink <==>
EndRoom Red
        {
            pathListOfCoordinates.Add(floorPathObject.GetChild(9));
            pathListOfCoordinates.Add(floorPathObject.GetChild(8));
        }
        else if (5 <= indexOfConnectorEndRoom && indexOfConnectorEndRoom <= 13 ||
                 indexOfConnectorEndRoom == 18) // StartRoom Pink <==> EndRoom Blue
        {
            if (indexOfConnectorEndRoom == 5 || indexOfConnectorEndRoom == 7 ||
indexOfConnectorEndRoom == 18)
```

```
            {
                if (indexOfConnectorEndRoom == 5 || indexOfConnectorEndRoom == 7)
                {
                    pathListOfCoordinates.Add(floorPathObject.GetChild(9));
                }
            }
            else
            {
                pathListOfCoordinates.Add(floorPathObject.GetChild(21));
                pathListOfCoordinates.Add(floorPathObject.GetChild(11));
            }
        }
        else if (14 <= indexOfConnectorEndRoom &&
                 indexOfConnectorEndRoom <= 17) // StartRoom Pink <==> EndRoom Green
        {
            pathListOfCoordinates.Add(floorPathObject.GetChild(21));
            pathListOfCoordinates.Add(floorPathObject.GetChild(11));
            pathListOfCoordinates.Add(floorPathObject.GetChild(12));
        }
        else if
            (indexOfConnectorEndRoom == 21 ||
             indexOfConnectorEndRoom ==
             19) // StartRoom Orange <==> EndRoom Orange || StartRoom Pink <==> EndRoom Pink
        {
        }
        else if (indexOfConnectorEndRoom == 22) // StartRoom Pink <==> EndRoom Grey
        {
            pathListOfCoordinates.Add(floorPathObject.GetChild(20));
        }
    }
    else if (indexOfConnectorStartRoom == 22) //StartRoom Grey
    {
        if (indexOfConnectorEndRoom == 22) // StartRoom Grey <==> EndRoom Grey
        {
        }
        else
        {
            pathListOfCoordinates.Add(floorPathObject.GetChild(20));

            if (0 <= indexOfConnectorEndRoom && indexOfConnectorEndRoom <= 4) //StartRoom Grey <==>
EndRoom Red
            {
                pathListOfCoordinates.Add(floorPathObject.GetChild(19));
                pathListOfCoordinates.Add(floorPathObject.GetChild(9));
                pathListOfCoordinates.Add(floorPathObject.GetChild(8));
            }
            else if (5 <= indexOfConnectorEndRoom && indexOfConnectorEndRoom <= 13 ||
                     indexOfConnectorEndRoom == 18) // StartRoom Grey <==> EndRoom Blue
            {
                if (indexOfConnectorEndRoom == 5 || indexOfConnectorEndRoom == 7 ||
indexOfConnectorEndRoom == 18)
                {
                    pathListOfCoordinates.Add(floorPathObject.GetChild(19));

                    if (indexOfConnectorEndRoom == 5 || indexOfConnectorEndRoom == 7)
                    {
                        pathListOfCoordinates.Add(floorPathObject.GetChild(9));
                    }
                }
                else
                {
                    pathListOfCoordinates.Add(floorPathObject.GetChild(21));
                    pathListOfCoordinates.Add(floorPathObject.GetChild(11));
                }
            }
            else if (14 <= indexOfConnectorEndRoom &&
                     indexOfConnectorEndRoom <= 17) // StartRoom Grey <==> EndRoom Green
            {
                pathListOfCoordinates.Add(floorPathObject.GetChild(21));
                pathListOfCoordinates.Add(floorPathObject.GetChild(11));
                pathListOfCoordinates.Add(floorPathObject.GetChild(12));
            }
        }
        // StartRoom Grey <==> EndRoom Orange || StartRoom Orange <==> EndRoom Pink
    }

    pathListOfCoordinates.Add(floorPathObject.GetChild(indexOfConnectorEndRoom));
    pathListOfCoordinates.Add(floorPathObject.GetChild(indexOfConnectorEndRoom)
        .GetChild(indexOfChildInConnectorEndRoom));
```

```
    //================Add Visible Part================
    visibleObjectList.Add(pathRoot.transform.GetChild(7).GetChild(0).gameObject);

visibleObjectList.Add(pathRoot.transform.GetChild(7).GetChild(pathRoot.transform.GetChild(7).childCo
unt - 1)
        .gameObject);
    //================Add Visible Part================
}
```

Розібравши детально найважливіші методи можна перейти до розгляду метода, який відіграє роль загального пошуку маршруту.

З наведених методів, які були розглянуті і пояснені в §8, було створено графи переходів, усіх можливих ситуацій, в залежності від початкових даних, введених користувачем.

Керуючись табличкою та методами, створено алгоритм пошуку. Для його розуміння, замість того щоб виконувати і ще один детальний лістининг коду, використовується схема у вигляді переходів з одного метода до іншого.

І такими переходами ми збираємо весь масив опорних точок, що знаходяться на окремих ділянках поверхів. Кожний метод позначений окремим кольором. Також пронумерований за яким порядком має визиватися метод, для правильної побудови маршруту.

Пункт 9.1 Загальна схема роботи Алгоритму з пошуку маршруту з використанням розбиття поверхів на підмножини

Загальні позначення методів на схемах:

- **FindStartAndEndRoomOnThe AF_N/LF_N –** де N це номер поверху.
  На графі переходів буде зображено як **FindSandEon _N**;
  (Методи які будуть будувати шлях за умови що початкова та кінцева аудиторія знаходиться на одному поверсі)

- **FindEndRoomOnThe AF_N/LF_N => FindEon _N;**
  (Методи які будуть будувати шлях за умови що кінцева аудиторія знаходиться на обраному поверсі)

- **BuildPathPassesThroughThe AF_N/LF_N => BuildThrough_N;**
  (Методи які будуть будувати шлях через поверх, що знаходиться між початковою та кінцевою аудиторіями)

- **FindStartRoomOnThe AFN AndPathToAnotherFloor => FindSon_N_andPathToNextF**
  (Методи які будують шлях через поверх, на якому знаходиться початкова кімната. Маршрут будується від початкової кімнати до відповідного переходу на наступний поверх)

За умови що :

Підмножина початкової аудиторії	Підмножина кінцевої аудиторії
AUD	AUD

Якщо початковий поверх - **це перший поверх** з підмножини аудиторних поверхів, тоді всі можливі варіанти маршруту представлені на Рис.49:

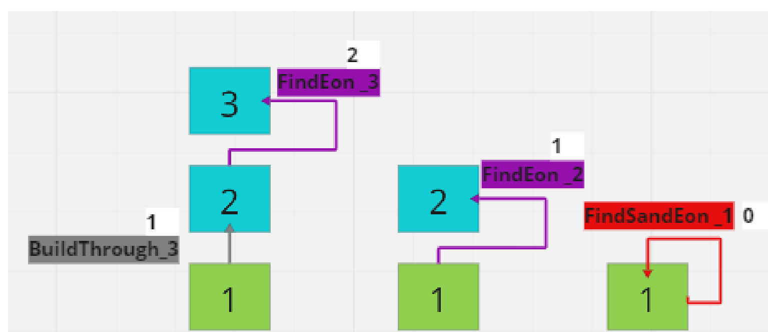

Рис.49

Якщо початковий поверх - **це другий поверх** з підмножини аудиторних поверхів, тоді всі можливі варіанти маршруту представлені на Рис.50:

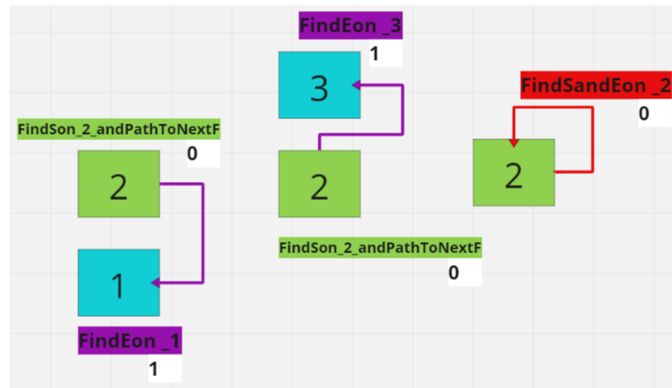

Рис.50

Якщо початковий поверх - **це третій поверх** з підмножини аудиторних поверхів, тоді всі можливі варіанти маршруту представлені на Рис.48:

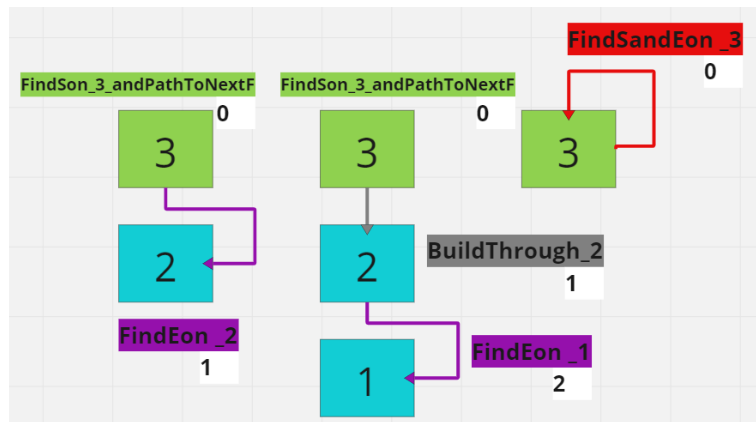

Рис.51

За умови що :

Підмножина початкової аудиторії	Підмножина кінцевої аудиторії
AUD	LAB

Якщо початковий поверх - **це другий поверх** з підмножини аудиторних поверхів, тоді всі можливі варіанти маршруту представлені на Рис.49:

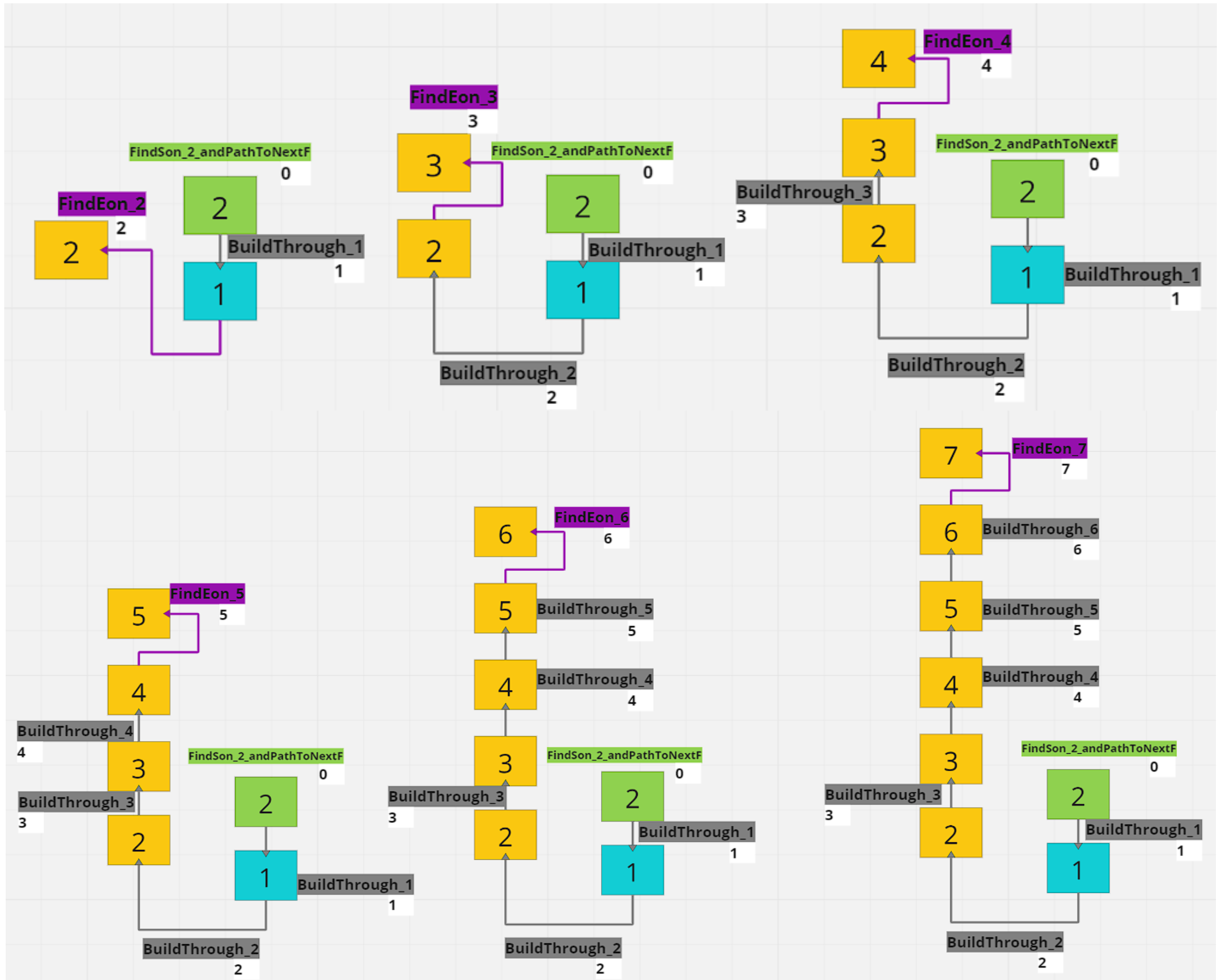

Рис.52

Якщо початковий поверх - **це третій поверх** з підмножини аудиторних поверхів, тоді всі можливі варіанти маршруту представлені на Рис.50, Рис.51 та Рис.52 :

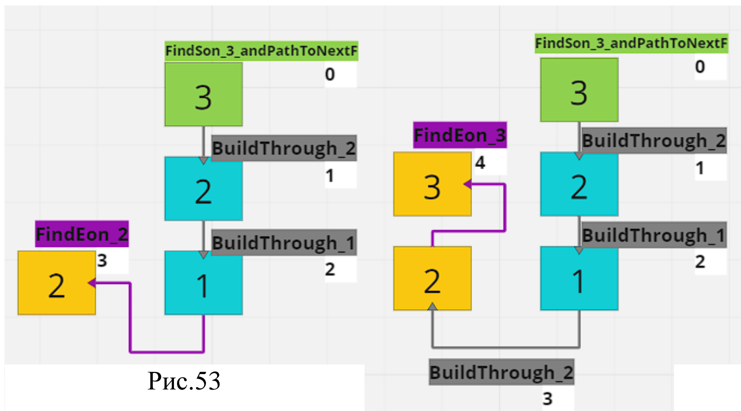

Рис.53

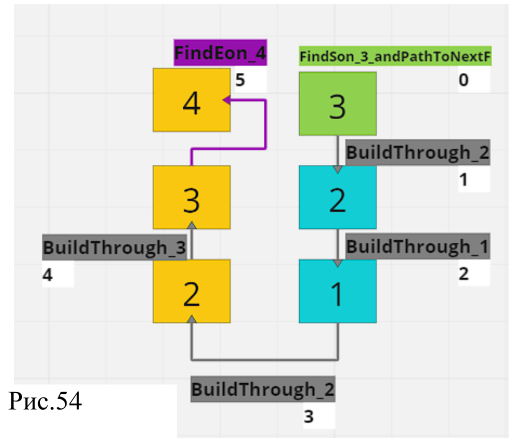

Рис.54

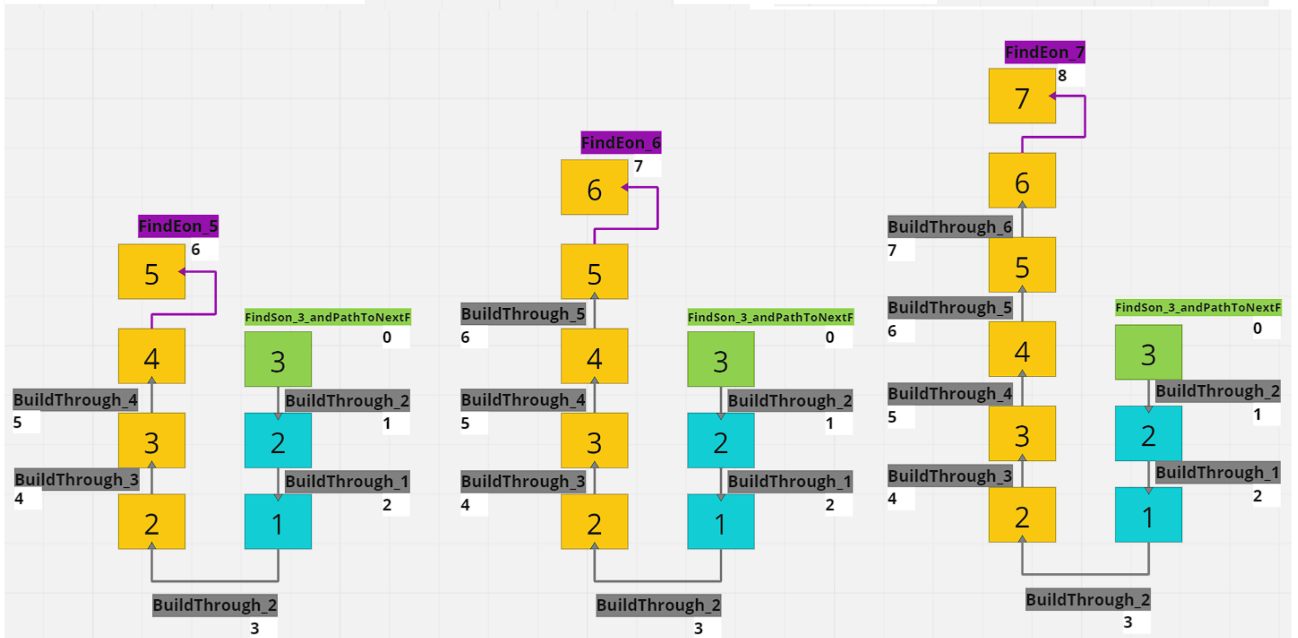

Рис.55

Якщо початковий поверх - **це перший поверх** з підмножини аудиторних поверхів, тоді всі можливі варіанти маршруту представлені на Рис.53:

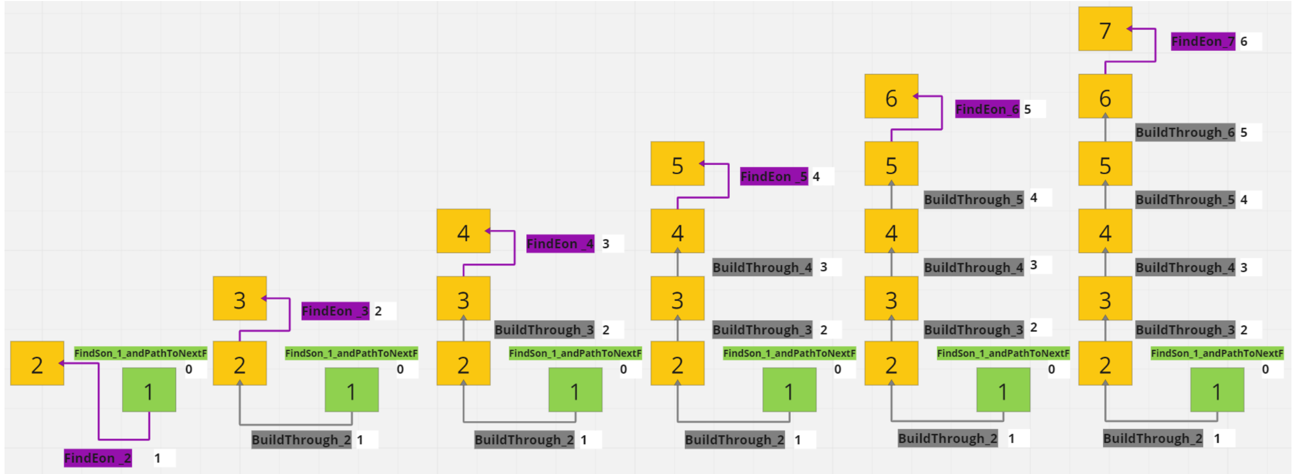

Рис.57

За умови що :

Підмножина початкової аудиторії	Підмножина кінцевої аудиторії
LAB	AUD

Якщо початковий поверх - **це сьомий поверх** з підмножини поверхів, які відносяться до поверхів лаб-корпусу, тоді всі можливі варіанти маршруту представлені на Рис.54:

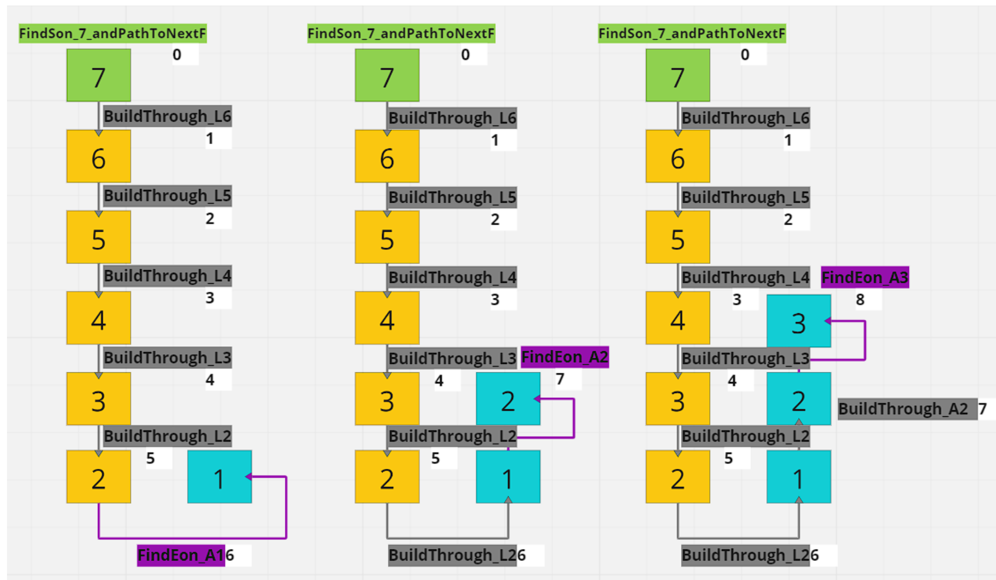

Рис.58

Якщо початковий поверх - **це шостий поверх** з підмножини поверхів, які відносяться до поверхів лаб-корпусу, тоді всі можливі варіанти маршруту представлені на Рис.55:

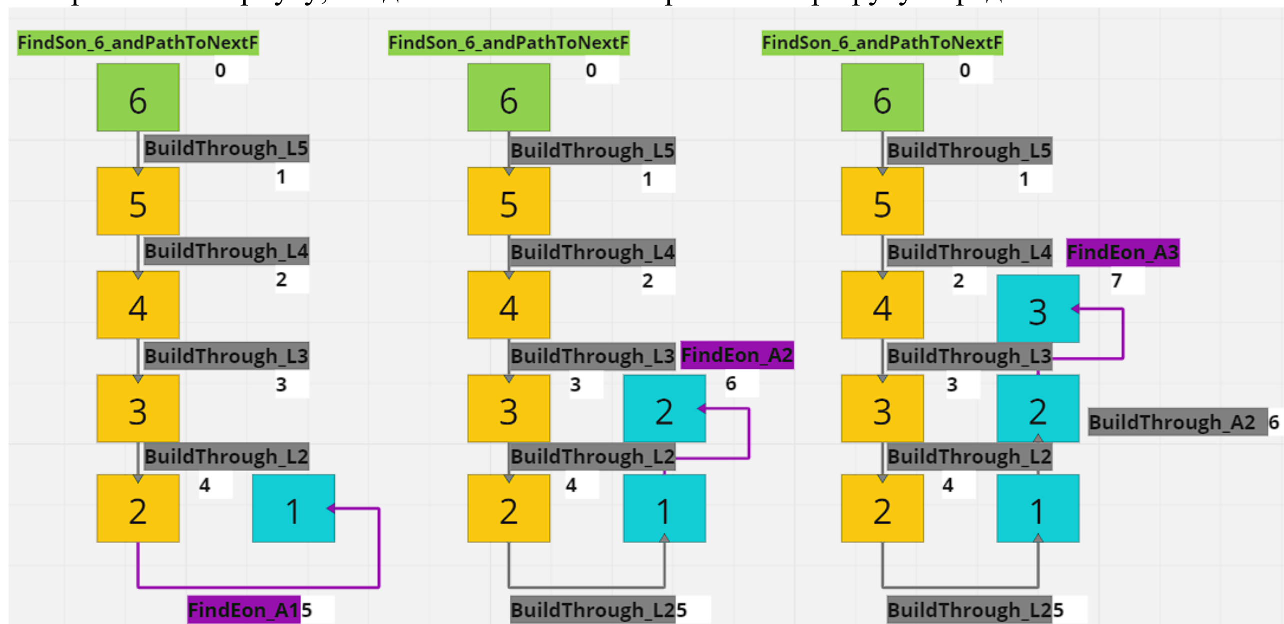

Рис.59

Якщо початковий поверх - **це п'ятий поверх** з підмножини поверхів, які відносяться до поверхів лаб-корпусу, тоді всі можливі варіанти маршруту представлені на Рис.56:

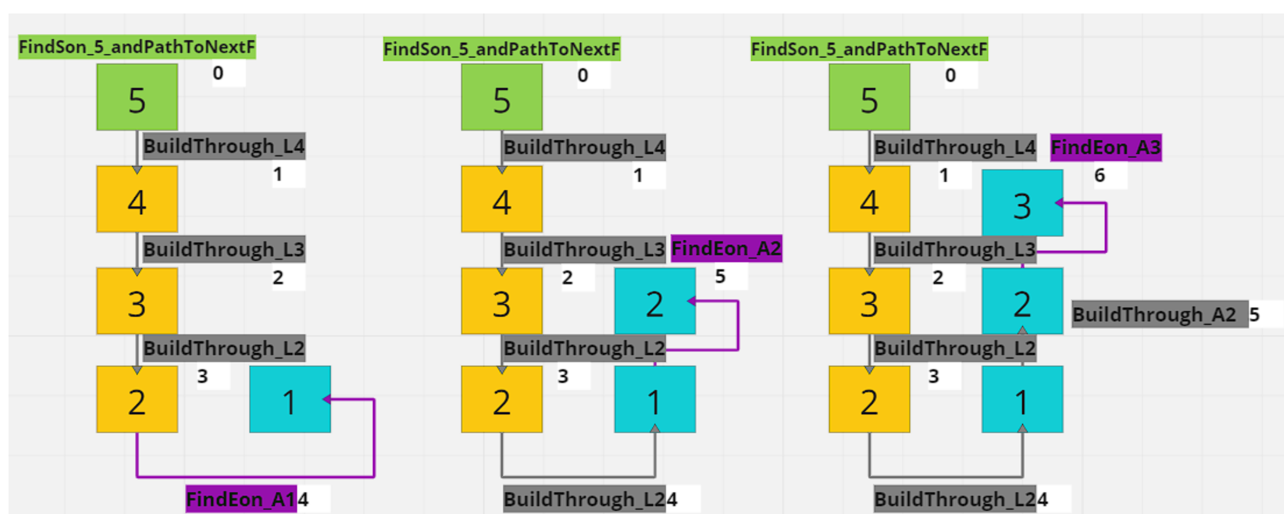

Рис.60

Для 4,3 та 2 поверхів подібний алгоритм побудови маршруту, тому розгляд графів переходу маршруту для даних поверхів не є доцільним.

За умови що :

Підмножина початкової аудиторії	Підмножина кінцевої аудиторії
LAB	LAB

Якщо початковий поверх - **це сьомий поверх** з підмножини поверхів, які відносяться до поверхів лаб-корпусу, тоді всі можливі варіанти маршруту представлені на Рис.57:

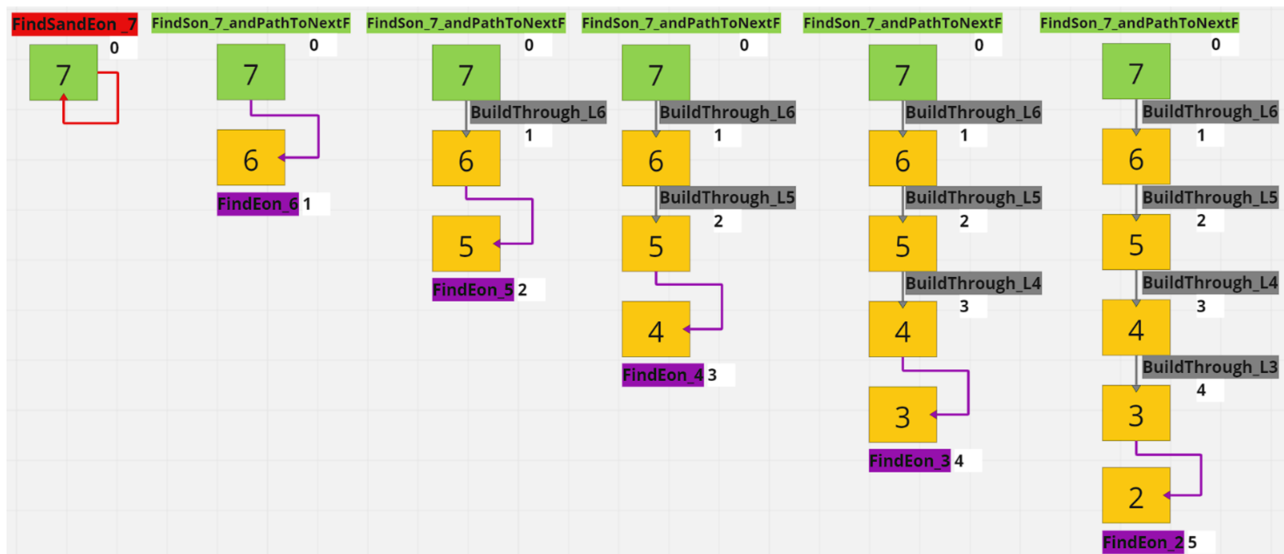

Рис.61

Якщо початковий поверх - **це шостий поверх** з підмножини поверхів, які відносяться до поверхів лаб-корпусу, тоді всі можливі варіанти маршруту представлені на Рис.58:

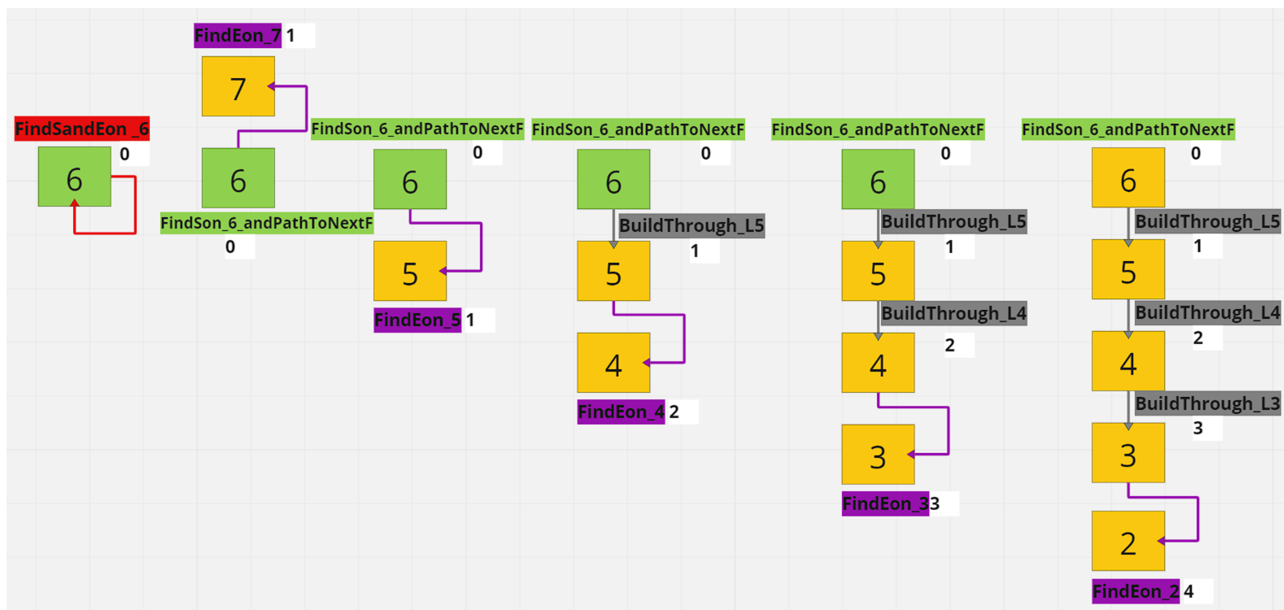

Рис.62

Якщо початковий поверх - **це п'ятий поверх** з підмножини поверхів, які відносяться до поверхів лаб-корпусу, тоді всі можливі варіанти маршруту представлені на Рис.59:

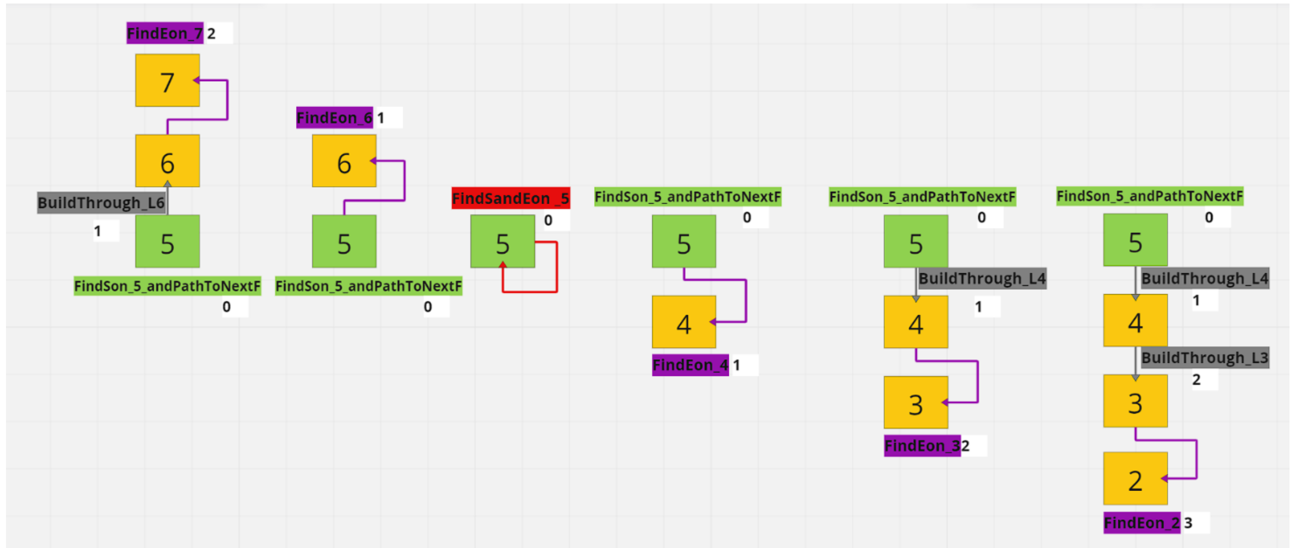

Рис.63

Для 4,3 та 2 поверхів подібний алгоритм побудови маршруту, тому розгляд графів переходу маршруту для даних поверхів не є доцільним.

Реалізація алгоритму, що зображений на Рис.57:

```
else if (startRoomInputText.Length > 2 &&
         endRoomInputText.Length > 2) //Start Room in LAB <==> End Room in LAB
{
    if (700 <= startRoomNumberInt && startRoomNumberInt <= 711) //Start Room in LF7
    {
        if (700 <= endRoomNumberInt && endRoomNumberInt <= 711)
                 //Start Room in LF7 <===> End Room in LF7
        {
            FindStartAndEndRoomOnTheLF7(startRoomInputText, endRoomInputText);
        }
        else if (600 <= endRoomNumberInt &&
                 endRoomNumberInt <= 611) //Start Room in LF7 <===> End Room in LF6
        {
            inputToLF6 = FindStartRoomOnTheLF7AndPathToAnotherFloorWithStairsR(startRoomInputText);
            FindEndRoomOnTheLF6(inputToLF6, endRoomInputText);
        }
        else if (500 <= endRoomNumberInt &&
                 endRoomNumberInt <= 511) //Start Room in LF7 <===> End Room in LF5
        {
            inputToLF6 = FindStartRoomOnTheLF7AndPathToAnotherFloorWithStairsR(startRoomInputText);
            inputToLF5 = BuildPathPassesThroughTheLF6(inputToLF6);
            FindEndRoomOnTheLF5(inputToLF5, endRoomInputText);
        }
        else if (400 <= endRoomNumberInt &&
                 endRoomNumberInt <= 411) //Start Room in LF7 <===> End Room in LF4
        {
            inputToLF6 = FindStartRoomOnTheLF7AndPathToAnotherFloorWithStairsR(startRoomInputText);
            inputToLF5 = BuildPathPassesThroughTheLF6(inputToLF6);
            inputToLF4 = BuildPathPassesThroughTheLF5(inputToLF5);
            FindEndRoomOnTheLF4(inputToLF4, endRoomInputText);
        }
        else if (300 <= endRoomNumberInt &&
                 endRoomNumberInt <= 311) //Start Room in LF7 <===> End Room in LF3
        {
            inputToLF6 = FindStartRoomOnTheLF7AndPathToAnotherFloorWithStairsR(startRoomInputText);
            inputToLF5 = BuildPathPassesThroughTheLF6(inputToLF6);
            inputToLF4 = BuildPathPassesThroughTheLF5(inputToLF5);
            inputToLF3 = BuildPathPassesThroughTheLF4(inputToLF4);
            FindEndRoomOnTheLF3(inputToLF3, endRoomInputText);
        }
        else if (200 <= endRoomNumberInt &&
                 endRoomNumberInt <= 233) //Start Room in LF7 <===> End Room in LF2
        {
            inputToLF6 = FindStartRoomOnTheLF7AndPathToAnotherFloorWithStairsR(startRoomInputText);
            inputToLF5 = BuildPathPassesThroughTheLF6(inputToLF6);
            inputToLF4 = BuildPathPassesThroughTheLF5(inputToLF5);
            inputToLF3 = BuildPathPassesThroughTheLF4(inputToLF4);
            inputToLF2 = BuildPathPassesThroughTheLF3(inputToLF3);
            FindEndRoomOnTheLF2(inputToLF2, endRoomInputText);
        }
    }
```

§10 Відображення підписів аудиторій на поверхах

Хоча в UnityEngine і є базовий клас **3DText**, однак при використанні даного об’єкту є досить важливий недолік:

Текст створений за допомогою даного об’єкту, рендериться поверх всі об`єктів, тому що це об’єкт належить сімейству об’єків, які використовують при роботі з UI. І при відображення декількох поверхів, підписи нижніх аудиторій відображаються під моделлю поверха, який знаходиться на д підписами. Це є суттєвим недоліком. Для вирішення цієї проблеми було створено масив 3D об’єктів символів українського алфавіту та скрипт, який формує правильний підпис, та його позицію відносно позиції кімнати.

Для створення масиву потрібних символів використовувався **3D DCC** редактор **Blender**. Всі моделі, що були створенні не використовували освітлення, яке додається при створені моделі. Це б впливало на освітлення, яке генерується за допомогою движку **Unity.**

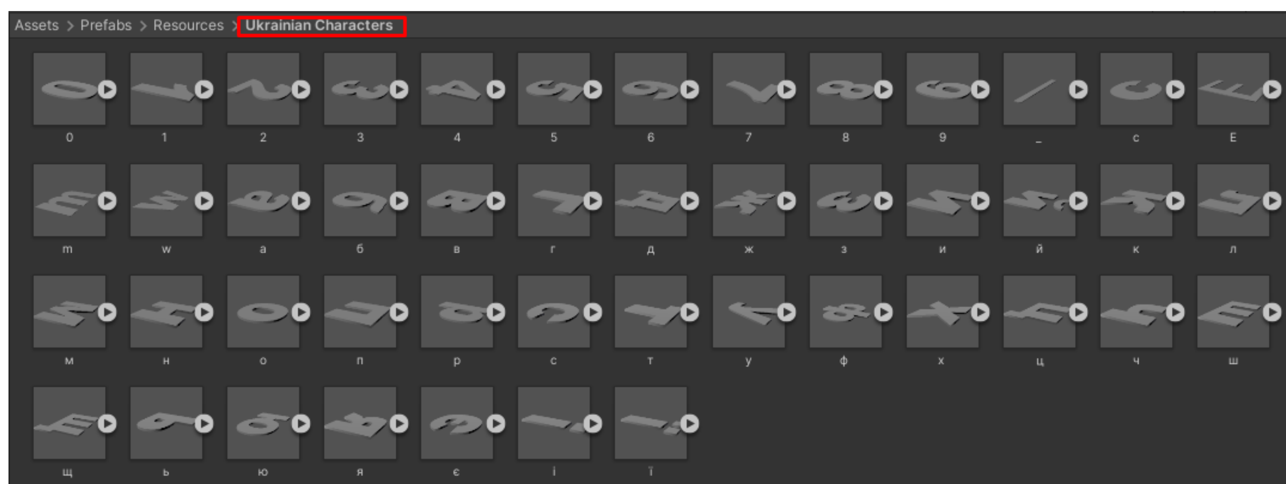

Рис.64

Також даний редактор використовувався при створені таких об’єктів як сходи(на Рис.80. зображено 1) та стіни сферичної форми (на Рис. 80. зображено 2):

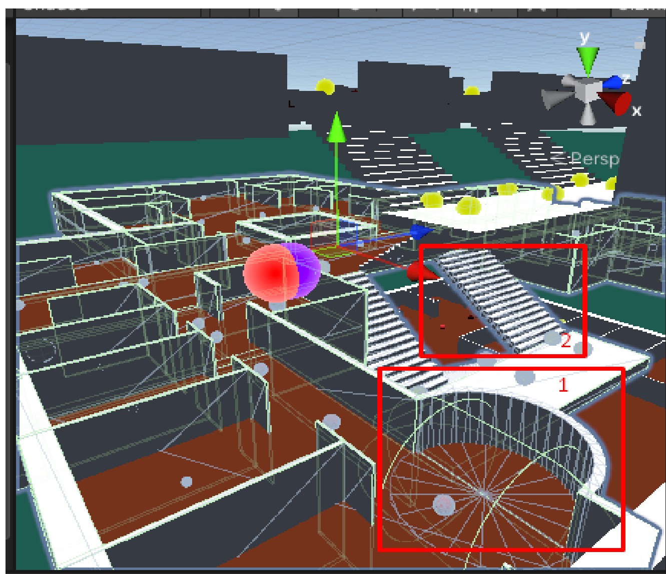

Рис.65

Далі розглянемо алгоритм за яким все ж таки відбувається побудова підпису, але перед цим потрібно розібрати компонент, без якого алгоритм би не мав можливості для реалізації

Пункт 10.1 Алгоритм побудови підпису аудиторій

Маючи позицію аудиторіїї та її назву, ми маємо всю необхідну інформацію для створення підпису. Розпишимо більше детально кроки даного алгоритму, який реалізований за допомогою функції **DrawLabelOfVisibleRoom()**. Кроки алгоритму розглянемо на принципі лише однієї аудиторії, так як для всіх інших аудиторій дії будуть аналогічні:

1. З назви аудиторії отримати всі символи, з яких складається назва.
2. Для кожного символа знайти відповідну 3D модель.
3. Всі знайдені моделі, створити на сцені на позиції об'єкта, що відграє роль аудиторії(**tag:room**)
4. Для кожної моделі створити опорну точку в найнижчій вершині з якої складається модель. Дана опорна точка буде знаходитись :
   - X(компонента) – найлівіша вершина 3D моделі
   - Y(компонента) – центр 3D моделі
   - Z(компонента) – найнижча точка по даній осі

5. Надначаємо новий об'єкт(**anchorPoint**) для кожної моделі символа зі знайденими координатми, як **PObj.**
6. Проходимо по всім опорним точкам, і знаходимо точку з найнижчим значенням по осі Z.
7. Маючи позицію найнижчою опорної точки, вирівнюємо всі опорні точки за найнижчою опорною точкою.

Після вирівнювання всі опорних точок, ми вирівнювали всі букви, тепер настав час правильно їх розмістити, зі сталою відстанню між літерами:

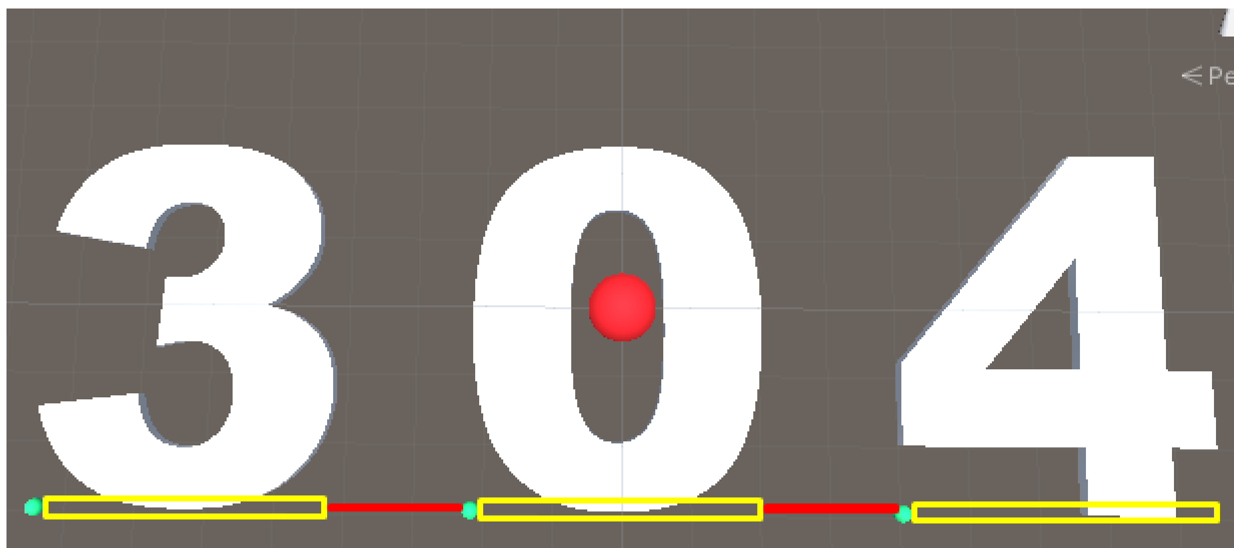

Рис.67

Жовтим виділена ширина кожної моделі літери. Червоним виділена стала довжина, для відступу.

8. Для кожного символу ми зазначаємо нову позицію (***позиція попередньої опорної точки* + *ширина символу* + *стала відтань відступу***)

9. Дізнаємось центр утвореного надпису і на даній позиції створюємо **PObj(mainAnchor)**, дочірніми об'єкта якого являються всі утворені літери, з яких утворений підпис аудиторії.
10. Присвоюємо позицію для об'єкта **mainAnchor** позицію обраної кімнати(**tag:room**)
11. Для кожного дочірнього об'єкта **mainAnchor** назначаємо об'єкт кімнату як **POBj**.
12. Видаляємо непотрібний **mainAnchor.**

Після виконаних дій ми відключаєм відображення лише об`єкта кімнати та всі інших коннекторів, так як на карті непотрібно відображати опорні точки.

Реалізація даного алгоритму:

```
private void DrawLabelOfVisibleRoom()
{
    GameObject[] rooms = GameObject.FindGameObjectsWithTag("Room");
    GameObject[] connectors = GameObject.FindGameObjectsWithTag("connector");
    GameObject[] listOfPrefabsLabels = Resources.LoadAll<GameObject>("Ukrainian Characters");

    foreach (var room in rooms)
    {
        GameObject mainAnchor = new GameObject();
        List<GameObject> tempWord = new List<GameObject>();
        List<GameObject> numberInLable = new List<GameObject>();
        List<GameObject> anchorPoints = new List<GameObject>();

        //собиреам префабы, которые необходимы для отображения символов, которые находяться в метке
комнаты.
        char[] numbersInRoomNumber = room.name.ToCharArray(0, room.name.Length);

        for (int i = 0; i < numbersInRoomNumber.Length; i++)
        {
            string checkNumber = numbersInRoomNumber[i].ToString();

            for (int j = 0; j < listOfPrefabsLabels.Length; j++)
            {
                if (checkNumber == listOfPrefabsLabels[j].name)
                {
                    tempWord.Add(listOfPrefabsLabels[j]);
                }
            }
        }

        // создаем все символы в одной точке, кординаты которой отвечают кординатам позици метки
комнаты
        foreach (var prefab in tempWord)
        {
            var position2 = room.transform.position;
            GameObject numberInlabel = Instantiate(
                prefab,
                new Vector3(position2.x, position2.y, position2.z),
                Quaternion.Euler(0, 0, 0)
            );

            numberInlabel.name = prefab.name + "_L";
            numberInLable.Add(numberInlabel);
        }
        //создаем опорные точки, которые помогут нам в построении слова

        foreach (var number in numberInLable)
        {
            List<Vector3> tempReferencePoint = FindReferencePoints(number);

            GameObject leftRefPoint = Instantiate(symbolAnchor);
            leftRefPoint.transform.position = tempReferencePoint[0];
            number.transform.SetParent(leftRefPoint.transform);

            anchorPoints.Add(leftRefPoint.gameObject);
        }

        var lowestZ = anchorPoints.Select(point => point.transform.position.z).OrderBy(z =>
z).First();

        for (int i = 0; i < anchorPoints.Count; i++)
        {
```

```
            anchorPoints[i].transform.position = new Vector3(anchorPoints[i].transform.position.x,
                anchorPoints[i].transform.position.y, lowestZ);
        }

        for (int i = 1; i < anchorPoints.Count; i++)
        {
            var previosTempLetter = anchorPoints[i - 1].transform;

            var tempOffset = previosTempLetter.GetComponent<MeshRenderer>().bounds.size.x +
offsetStep;

            anchorPoints[i].transform.position = new Vector3(anchorPoints[i -
1].transform.position.x + tempOffset,
                anchorPoints[i].transform.position.y,
                anchorPoints[i].transform.position.z);
        }

        var centreOfLabel = (anchorPoints[0].transform.position.x +
                             anchorPoints[anchorPoints.Count - 1].transform.position.x) / 2.0f;

        var position1 = room.transform.position;

        mainAnchor.transform.position = new Vector3(centreOfLabel,
            position1.y,
            position1.z);

        for (int i = 0; i < anchorPoints.Count; i++)
        {
            anchorPoints[i].transform.SetParent(mainAnchor.transform);
        }

        mainAnchor.transform.SetParent(room.transform);
        mainAnchor.name = $"mainAnchor{room.name}";

        var position = mainAnchor.transform.position;
        position = new Vector3(room.transform.position.x, position.y, position.z);
        mainAnchor.transform.position = position;

        for (int i = 0; i < mainAnchor.transform.childCount; i++)
        {
            mainAnchor.transform.GetChild(i).GetChild(0).SetParent(room.transform);
        }

        Destroy(mainAnchor);
        tempWord.Clear();
        numberInLable.Clear();
        anchorPoints.Clear();
    }

    foreach (var room in rooms)
    {
        var visibility = room.GetComponent<MeshRenderer>();
        visibility.enabled = false;
    }

    foreach (var connector in connectors)
    {
        var visibility = connector.GetComponent<MeshRenderer>();
        visibility.enabled = false;
    }
}
```

§11 Алгоритм знаходження необхідної кількості створених генераторів частинок, при побудові маршруту

За нашими налаштуваннями при русі генератора частинок, ми отримуємо певний відрізок лінії, яка складається з маленьких створених частинок. Однак виникає певна проблема.

Якщо **кількість частинок** буде налаштована некоректно, на певній ділянці маршруту будуть з'являтись відрізки з пробілами. Щоб уникнути даної проблеми було прийнято рішення створити функцію, яка регулюватиме цю характеристику.

Функція **FindAmountPathDrawers**:

```
private int FindAmountPathDrawers(List<Transform> ListCordinates)
{
    float distance = 0.0f;
    for (int i = 0; i < ListCordinates.Count - 1; i++)
    {
        distance += Vector3.Distance(ListCordinates[i].position, ListCordinates[i + 1].position);
    }

    int amountOfPathDrawers = Mathf.RoundToInt(distance / DistanceBetweenPoints);
    return amountOfPathDrawers;
}
```

Додатковий рисунок для пояснення алгоритму знаходження потрібної кількості генераторів частинок:

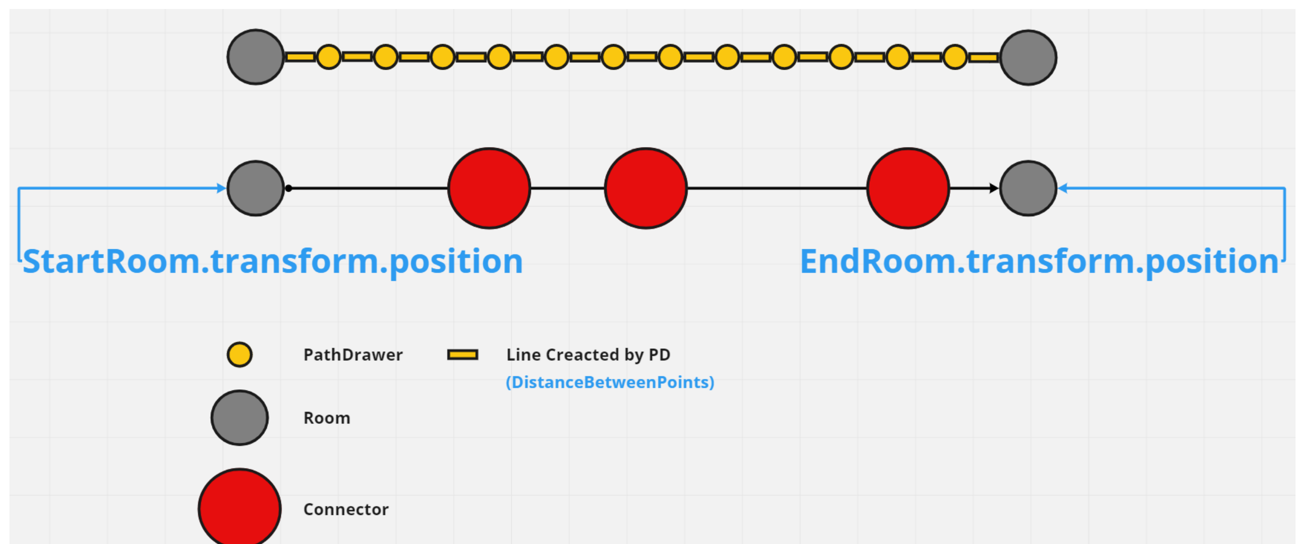

Рис.68

Маючи список опорних точок, по яких будується маршрут, можна знайти загальну довжину маршруту між позиціями початкової та кінцевої кімнат.

Також маючи значення потрібної довжини відрізку між генераторами частинок, при умові що за довжини відрізка не виникають пробіли, можна знайти необхідну кількість генераторів частинок.

Знайдене значення довжини маршруту необхідно поділити на значення довжини відрізку, отримане значені буде дорівнювати кількості генераторів частинок, якої вистачить для уникнення появи пробілів.

Розділ IV. UI з яким взаємодіє користувач

Для взаємодії користувача та програми було додано об’єкт **Canvas**. Завдяки цьому об`єкту у розробника, що використовує движок **Unity** є можливість розробки користувацького інтерфейсу(**User Interfase**) для взаємодії користувача та додатку.

Перелічимо всі потреби користувача при взаємодії з інтерактивною картою будівлі(***У нашому випадку плану факультету***).

Користувачу потрібна можливість:

1. Задати назву початкової (***аудиторія біля якої знаходиться користувач***) та кінцевої(***аудиторія до якої потрібно потрапити користувачу***) аудиторій.
2. Задати початок малювання маршруту.
3. Припинити відображення маршруту.
4. Обертати модель карти за потреби.
5. Збільшувати або зменшувати масштаб відображення карти.
6. Повертати видимі частини карти в початкове положення.

Для реалізації пункту:

1 – створено два об`єкти **StartRoomInputField,EndRoomInputField,** які являють собою об`єкти **inputField.**

2 – створено об’єкт **StartDrawButton,** який являє собою об`єкт **Button**, базовий об`єкт при роботі з UI.

3 - створено об’єкт **StopDrawButton,** який являє собою об`єкт **Button**, базовий об`єкт при роботі з UI.

4 – створено функцію **Rotate_Object()** яка оброблює дотики користувача, та повертає видимі частини інтерактивної карти.

5 - створено функцію **Zoom()** яка оброблює дотики користувача, та зменшує або збільшує масштаб карти.

6 – створено об’єкт **SetStartPosition(),** який являє собою об`єкт **Button**, базовий об`єкт при роботі з UI.

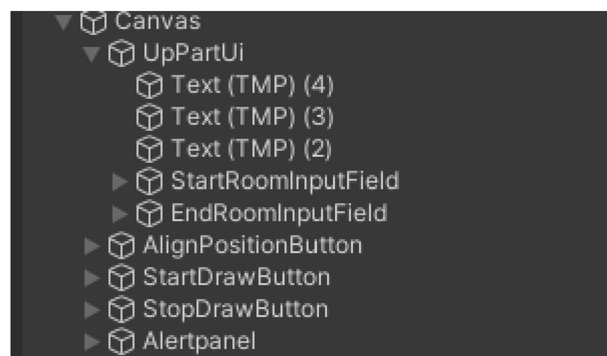

Рис.69. Об’єкти, що були створені для взаємодії з UI

Отже всі створенні об’єкти, досить зручно можна розмістити на об’єкті **Canvas**. Для коректного відображення всіх компонентів інтерфейсу необхідно використовувати прив`язки частин екрану.

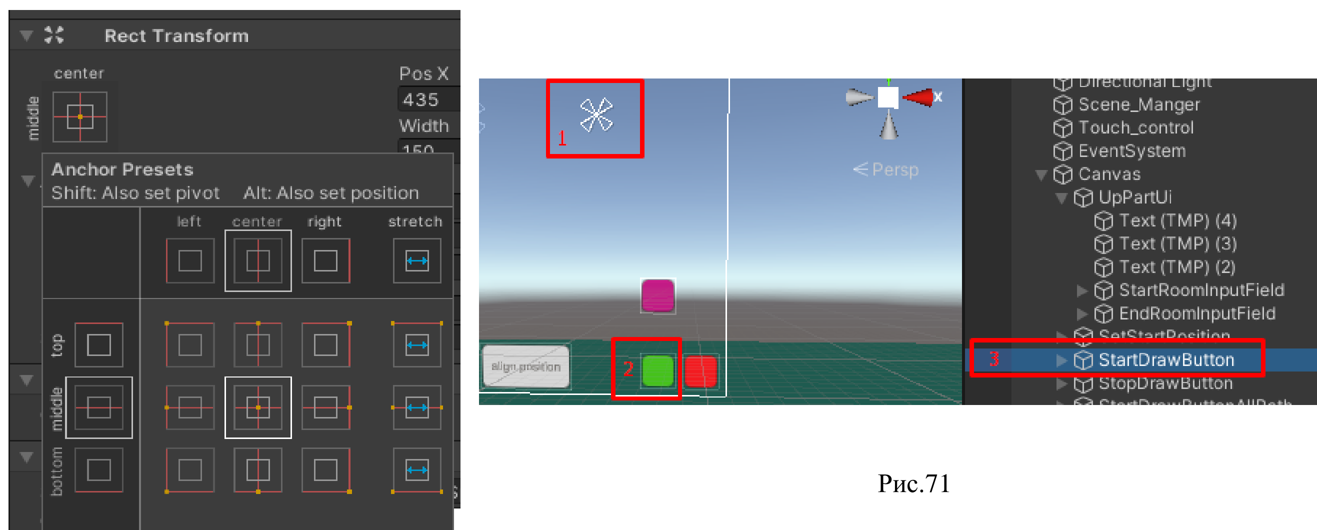

Рис.70

Рис.71

Також, щоб уникнути проблем з відображенням елементів UI на пристроях з різною роздільною здатністю, у об’єкта **Canvas** присутній компонент **Canvas Scaler :**

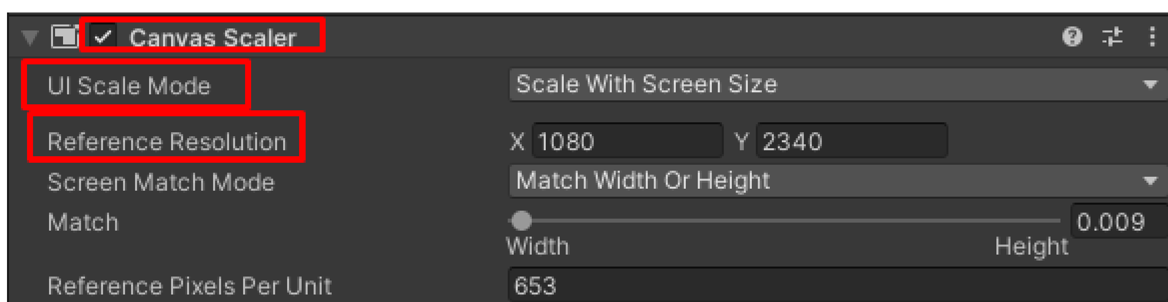

Рис.72

Компонент **Canvas Scaler** використовується для керування загальним масштабом і щільністю пікселів елементів інтерфейсу користувача в **Canvas.**

Властивості компоненту **Canvas Scaler** які потрібно налаштувати :

- **UI Scale Mode -** визначає, як масштабуються елементи інтерфейсу користувача на Canvas.
    - **Constant Pixel Size -** змушує елементи інтерфейсу користувача зберігати однаковий розмір у пікселях незалежно від розміру екрана.
    - **Scale With Screen Size -** збільшує елементи інтерфейсу користувача, чим більше екран.
    - **Constant Physical Size -** змушує елементи інтерфейсу користувача зберігати однаковий фізичний розмір незалежно від розміру та роздільної здатності екрана.
- **Reference Resolution -** роздільна здатність, для якої призначений макет інтерфейсу користувача. Якщо роздільна здатність екрана більша, інтерфейс користувача буде збільшено, а якщо воно менше, інтерфейс буде зменшено. Обрана роздільна здатність телефону на якому і тестувався додаток : Xiaomi Redmi 9T – 1080x2340.

§12 Обробка даних введених користувачем

Отже, на об'єкті **Canvas** присутні два об'єкта типу **InputFuild.** В цього об'єкта присутній компонент **TextMeshPro -InputField.** В даного копонента присутнє поле **Text.** Саме це поле буде містити в собі текст який вводить користувач.

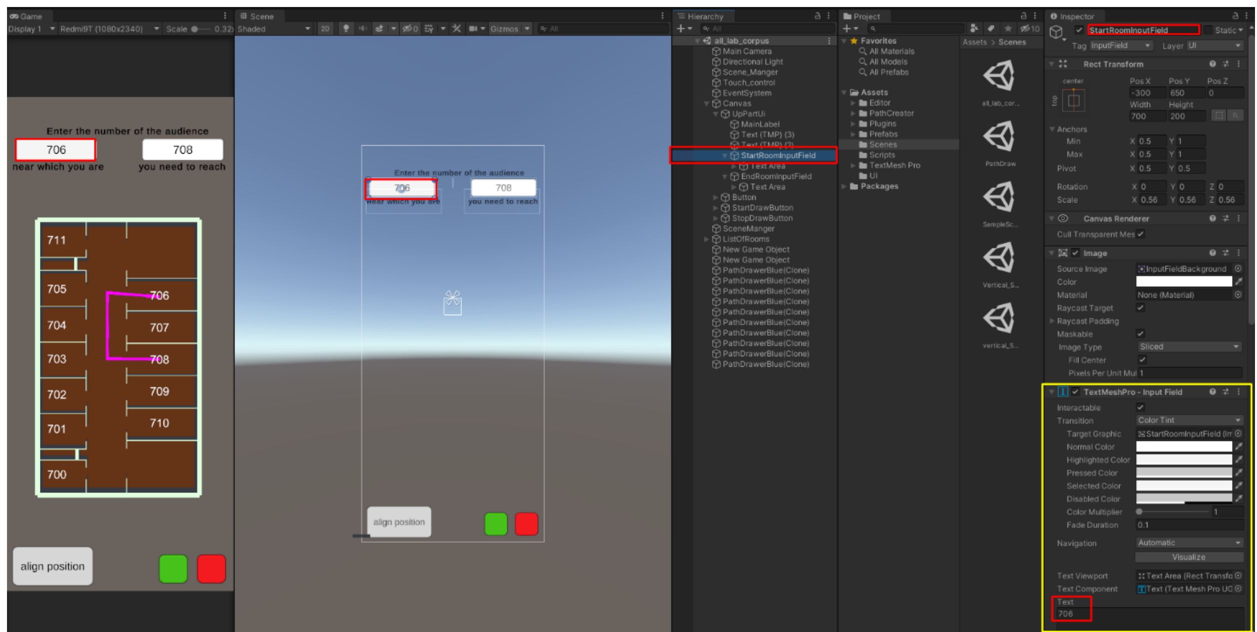

Рис.73

За допомогою коду ми дізнаємося поверх переводячи цей текст в числовий тип даних. Також можемо оброблювати чи коректно користувач вів дані.

```
char[] numbersInStartRoomNumber = start_room.ToCharArray( startIndex: 0,  length: start_room.Length - 1);
char[] numbersInEndRoomNumber = end_room.ToCharArray( startIndex: 0,  length: end_room.Length - 1);

int startRoomNumber = Convert.ToInt32(numbersInStartRoomNumber[0].ToString());
int endRoomNumber = Convert.ToInt32(numbersInEndRoomNumber[0].ToString());

Debug.Log( message: $"indexOfStartFloor : {startRoomNumber} | indexOfEndFloor : {endRoomNumber}");
```

При введені некоректних даних, користувачу буде виводитись панель з попередженням.

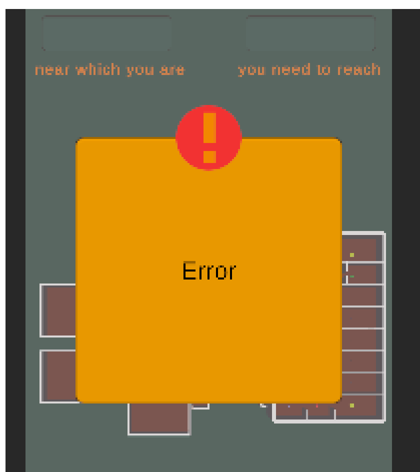

Рис.74

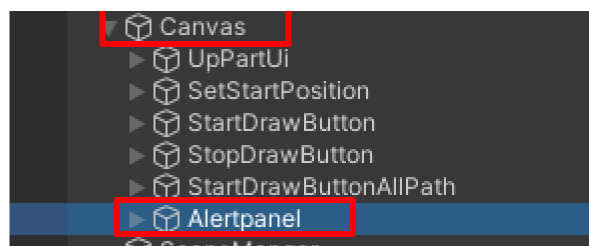

Рис.75

§13 Початок відображення маршруту та його припинення

Після того як користувач ввів дані. Після проходження перевірки користувач за бажанням може натиснути на кнопку **StartDrawButton().**

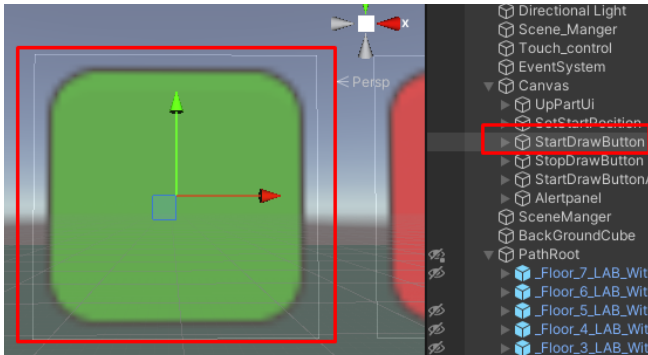

Рис.76

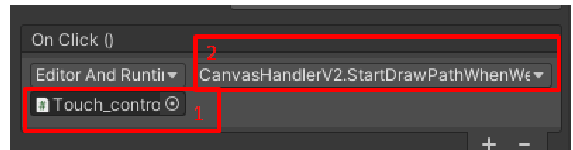

Рис.77

При натисканні на кнопку буде визиватися обрана функція (на рис позначено 2) з обраного скрипту(на рис. позначено 1)

```
public void StartDrawPathWhenWeHaveFire()
{
    ClearPathDrawers();
    CheckValueFromInputField();
    StartCoroutine(DrawPathRoomToRoom());
}
```

```
private void ClearPathDrawers()
{
    pathListOfCoordinates.Clear();
    foreach (var PathDraw in PathDrawers)
        Destroy(PathDraw);
    PathDrawers.Clear();
}
```

В функції **ClearPathDrawers()** видаляємо всі опорні точки з яких складався попередній маршрут. Також видаляємо всі генератори частинок, які проходили по маршруту та малювали траєкторію, за якою має пройти користувач.

```
private void CheckValueFromInputField()
{
    _inputValueOfRooms = GameObject.FindGameObjectsWithTag("InputField")
        .Select(GObj => GObj.GetComponent<TMP_InputField>()).ToArray();

    var startRoomName = _inputValueOfRooms[0].text;
    var endRoomName = _inputValueOfRooms[1].text;

    bool startWorkFlag = false;


    string startRoomInputText = CheckInputFeild(startRoomName);

    string endRoomInputText = CheckInputFeild(endRoomName);

    Debug.Log($"startRoomInputText : {startRoomInputText} | endRoomInputText : {endRoomInputText}");

    if (startRoomInputText == "err" || startRoomInputText == "errN")
    {
        if (startRoomInputText == "err")
        {
            OpenAlertPanel("<b> Error </b> with <b> audience near which you are </b>");
        }
        else if (startRoomInputText == "errN")
        {
            OpenAlertPanel("<b> Error </b> with <b> audience near which you are N </b>");
        }
```

```
    }
    else if (endRoomInputText == "err" || endRoomInputText == "errN")
    {
        if (endRoomInputText == "err")
        {
            OpenAlertPanel("<b> Error </b> with <b> audience you want to reach </b>");
        }
        else if (endRoomInputText == "errN")
        {
            OpenAlertPanel("<b> Error </b> with <b> audience you want to reach N </b>");
        }
    }
}
```

В даному скрипті перевіряємо введені дані користувача, при введені некоректних даних відкриваємо панель з попередженням.

```
StartCoroutine(DrawPathRoomToRoom());
```

Останній прядок в функції, яка малює шлях, ми викликаємо загальну функцію, яка знаходить маршрут від початкової до кінцевої кімнати.

Для припинення відображення маршруту використовуємо кнопку **StopDrawButton** :

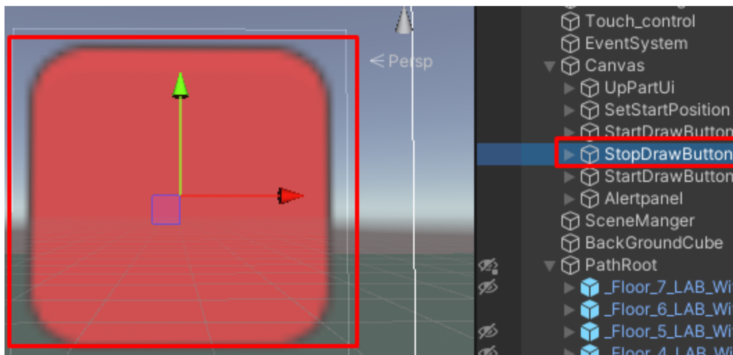

Рис.78

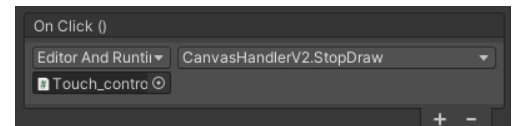

Рис.79

```
public void StopDraw()
{
    ClearPathDrawers();
    StopAllCoroutines();
}
```

§14 Обертання інтерактивної карти

Отже обговоримо загальний принцип за яким відбуваються всі події на сцені під час роботи додатку.

Порядок виконання функцій, які відповідають за події в движку Unity

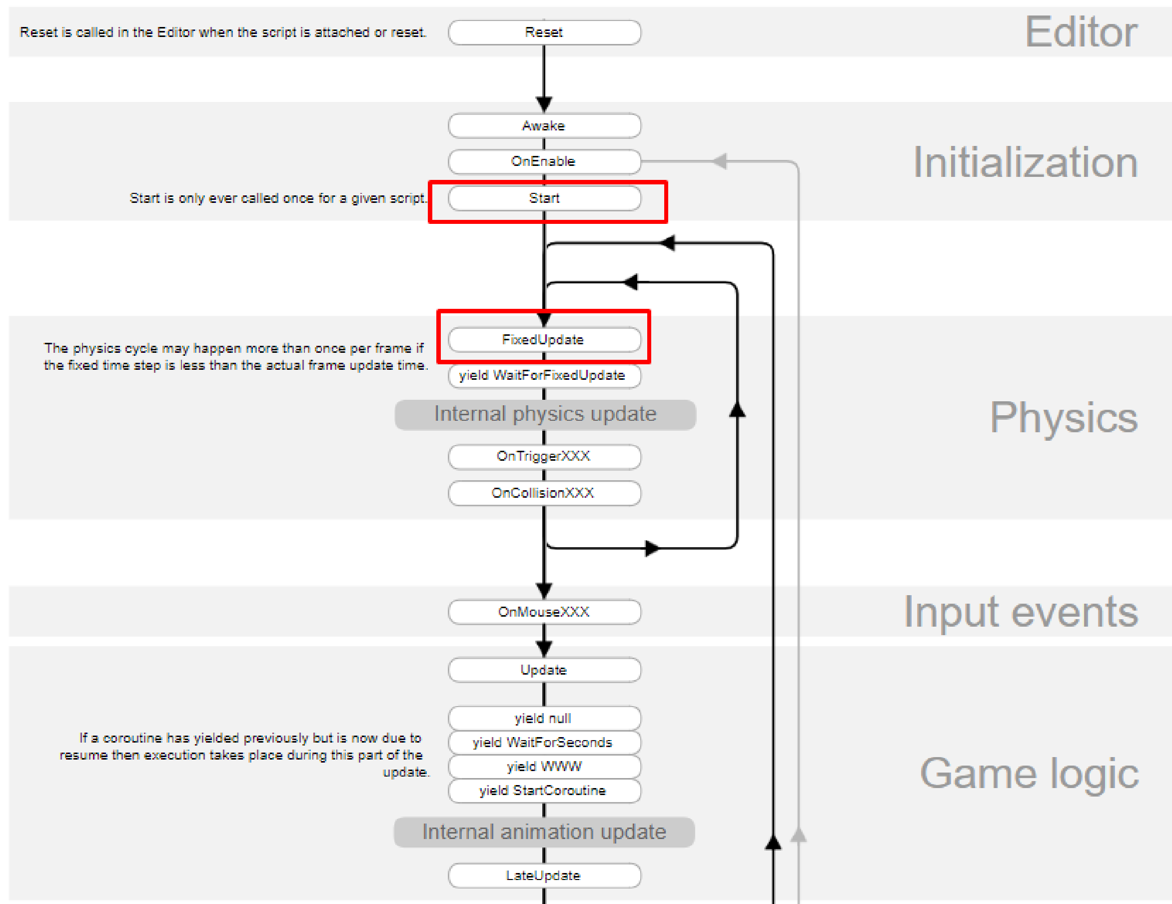

Рис.80

Обробку дотиків дозволяє реалізувати клас **Input,** який являє собою інтерфейс системи, який оброблює ввід інформації в движку Unity.

Отже при запуску сцени ми одразу створюємо два списка, в які будемо зберігати положення дотику по осям x та y.

```
public GameObject Canvas;   Canvas
public Camera cam;   Changed in 1 asset
public GameObject Flour;   Square
public GameObject Active_touch;   active_touch
public GameObject Previos_touch;   previous_touch
List<float> touch_positions_x = new List<float>( capacity: 10){};
List<float> touch_positions_y = new List<float>( capacity: 10){};
public int waiting_step;   "4"
public int length;   "8"
float angle;
```

Змінна **wating_step** відповідає за кількість викликів функції **FixedUpdate(на Рис. позначено фіолетовими стрілочками)** між поточним положенням дотику та дотиком який користувач виконував певну кількість часу тому.

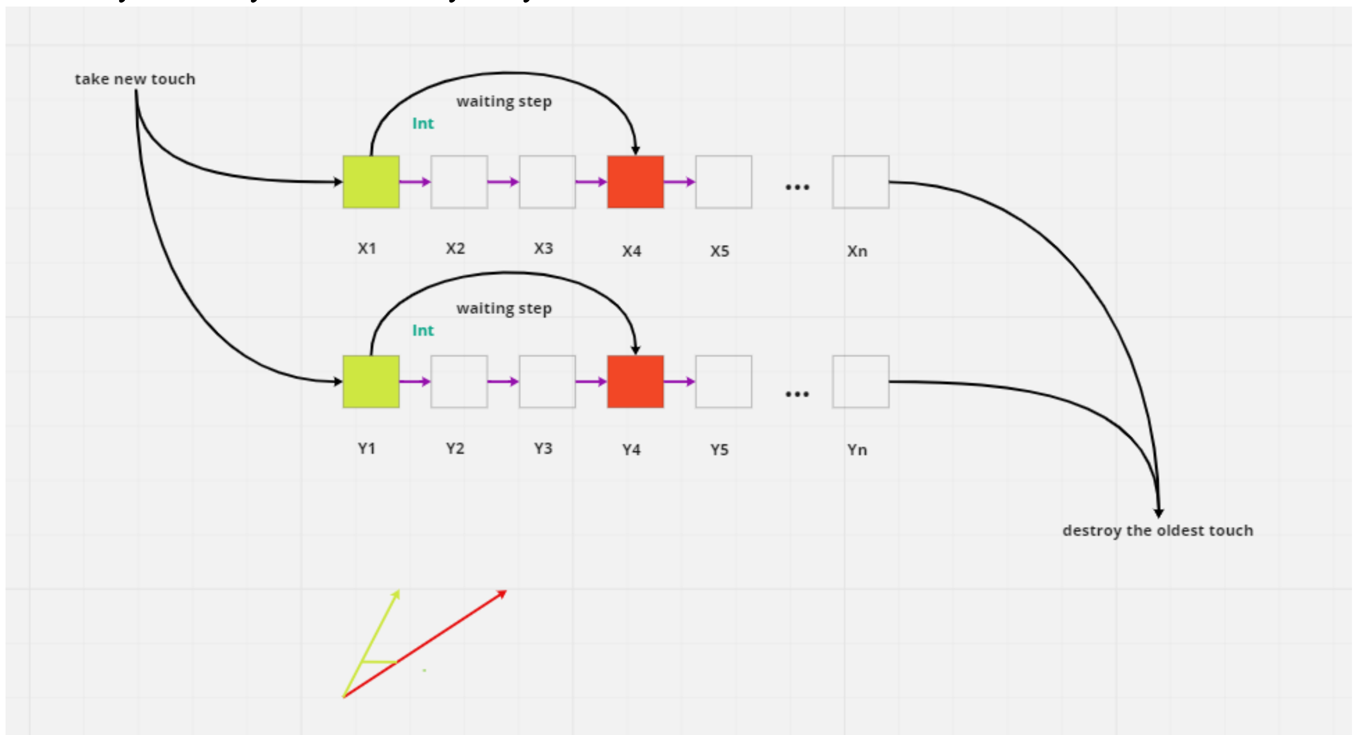

Рис.81

X1 – положення поточного дотику на осі X;

Y1 – положення поточного дотику на осі Y;

X4 – положення дотику збереженого певну кількість часу(***Time.waitTime * wating_step***) до поточного дотику на осі X.

Y4 – положення дотику збереженого певну кількість часу(***Time.waitTime * wating_step***) до поточного дотику на осі Y.

Розберемо фази, які може приймати дотик під час взаємодії користувача та сенсорного екрану:

- **TouchPhase.Began** – палець торкнувся екрану.
- **TouchPhase.Moved** – палець почав рухатись по екрану.
- **TouchPhase.Ended** – палець покинув екран.

Ми дізнаємось кут між теперішнім дотиком та дотиком, який записався певний час тому назад під час фази **Moved**. І в залежності від того куди рухався дотик користувача, по горизонталі чи вертикалі, ми повертаємо видимі частини карти навколо осей X чи Y відповідно.

Після того як палець покинув екран, ми видаляємо зі списку всі позиції дотиків. Це робиться з метою обійти баг, який виникає через те, що виникає досить великий кут між останім положенням попереднього дотику і стартовим положенням поточного дотику. Кут виникає на стільки великим, що модель може поводити себе непередбачувано при повороті.

На наступній сторінці наведений скрипт, який відповідає за поворот інтерактивної карти карти.

```
void Rotate_Object()
{
    Vector2 delta = Input.GetTouch(0).deltaPosition;

    Touch touch = Input.GetTouch(0);

    switch (touch.phase)
    {
        case TouchPhase.Began: //fix with change touch postion

            for (int i = 0; i <= waiting_step; i++)
            {
                touch_positions_x[i] = touch.position.x;
                touch_positions_y[i] = touch.position.y;
            }

            break;

        case TouchPhase.Moved:

            if (touch_positions_x.Count > length)
            {
                touch_positions_x.RemoveRange(length - 1, touch_positions_x.Count - length);
                touch_positions_y.RemoveRange(length - 1, touch_positions_y.Count - length);
            }

            for (int i = touch_positions_x.Count - 1; i >= 1; i--)
            {
                touch_positions_x[i] = touch_positions_x[i - 1];
                touch_positions_y[i] = touch_positions_y[i - 1];
            }

            touch_positions_x.Insert(0, touch.position.x);
            touch_positions_y.Insert(0, touch.position.y);

            activeTouch.transform.position =
            new Vector2(touch_positions_x[0], touch_positions_y[0]);

            previosTouch.transform.position =
            new Vector2(touch_positions_x[waiting_step], touch_positions_y[waiting_step]);

            angle =
            Vector2.SignedAngle(previosTouch.transform.position, activeTouch.transform.position);

            if (Mathf.Abs(delta.x) > Mathf.Abs(delta.y))
            {
                if (Mathf.Abs(touch.position.y) <= Screen.height / 2)
                {
                    pathRoot.transform.Rotate(0, angle * RotationalSpeed, 0, Space.World);
                }
                else
                {
                    pathRoot.transform.Rotate(0, -angle * RotationalSpeed, 0, Space.World);
                }
            }
            else
            {
                pathRoot.transform.Rotate(angle * RotationalSpeed, 0, 0, Space.World);
            }

            break;

        case TouchPhase.Ended:

            if (buttonTouch == false)
            {
                for (int i = 0; i < touch_positions_x.Count; i++)
                {
                    touch_positions_x.RemoveAt(i);
                    touch_positions_y.RemoveAt(i);
                }
            }
            else
            {
                buttonTouch = false;
            }

            break;
    }
}
```

§14 Збільшення та зменшення масштабу відображення карти

Використовуючи клас **Input** у розробника з`являється безліч корисних полей, в які записуються різноманітні характеристики. Одне з важливих полів що, використовується при обробці дотиків, поле **touchCount.**

Input. touchCount – зберігає в собі кількість дотиків. Гарантовано не змінюється по проміжку всього кадру. (***поле лише для читання***).

Для збільшення або зменшення масштабу користувачі звикли використовувати два дотики.

При збільшенні масштабу, якщо порівнювати поточну відстань(AB) між дотиками та відстань між дотиками, яку ми записали певну кількість часу перед зчитуванням поточної відстані (**ab**):

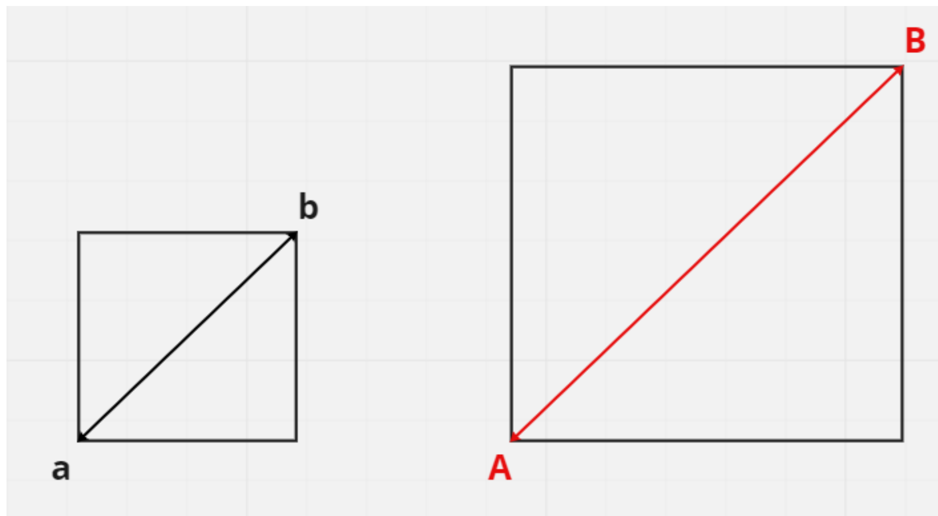

Рис.82. Демонстрація різниці відстані між дотиками при збільшені масштабу

|AB| > |ab| - умова за якої буде відбуватися збільшення масштабу.

При зменшені масштабу, якщо порівнювати поточну відстань(AB) між дотиками та відстань між дотиками, яку ми записали певну кількість часу перед зчитуванням поточної відстані (**ab**):

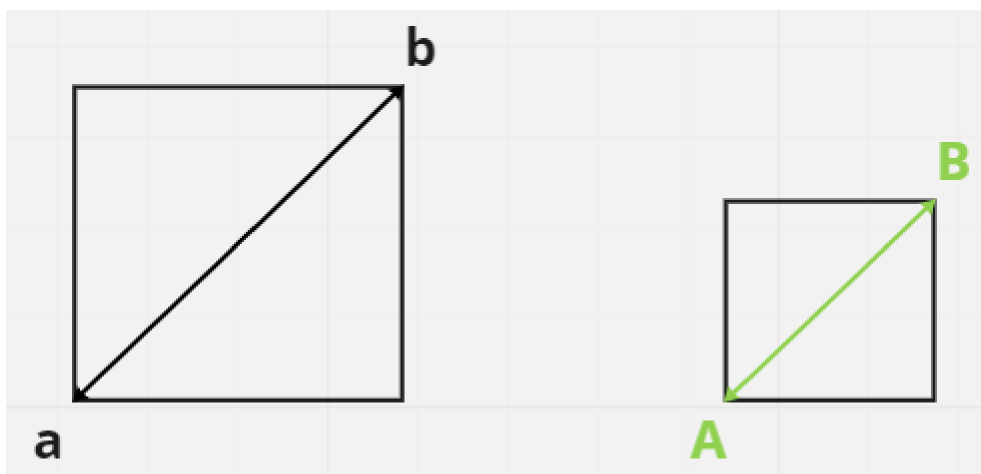

Рис.83. Демонстрація різниці відстані між дотиками при зменшені масштабу

|AB| < |ab| - умова за якої буде відбуватися збільшення масштабу.

Тепер, ми розібрали початкові умови за яких ми будемо змінювати масштаб інтерактивної карти. Далі розглянемо компоненти які використовуються для реалізації цієї задачі.

Пункт 14.1 Camera component

Камери - це пристрої, які знімають і показують світ користувачу. Їх можна налаштувати на відтворення в будь-якому порядку, у будь-якому місці екрана або лише певних частинах екрану. В даному проекті використовується лище одна камера(***Main Camera***).

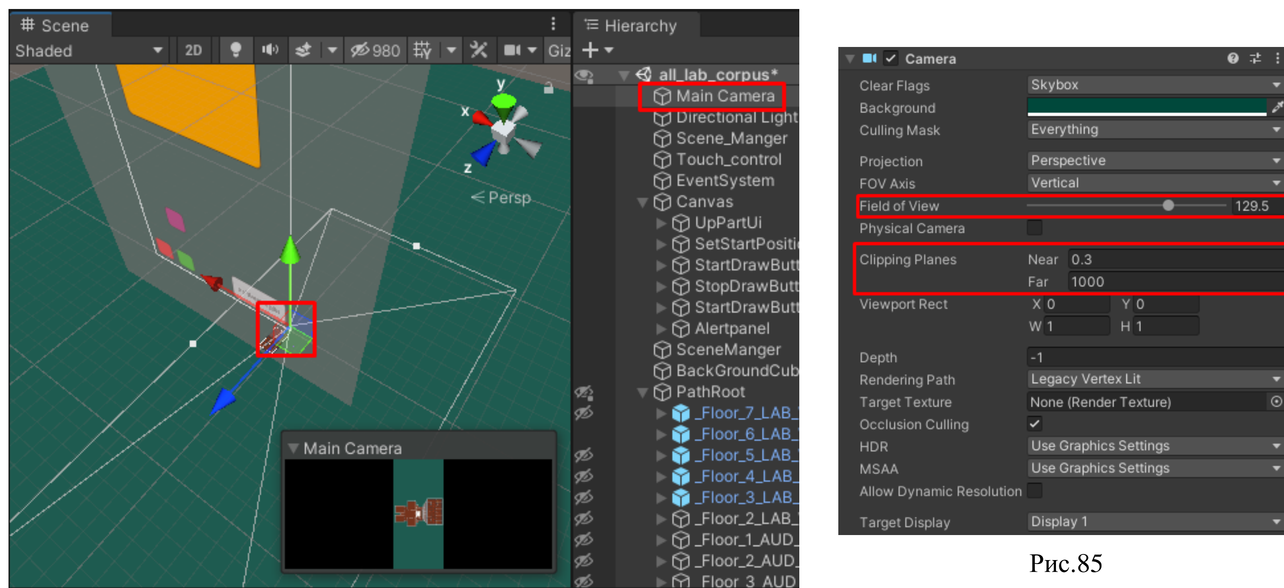

Рис.84

Рис.85

Властивості які необхідно налаштувати:

Clear Flags - визначає, які частини екрана будуть очищені. Це зручно при використанні кількох камер для малювання різних елементів гри.

Culling Mask - включає або пропускає шари об'єктів, які відображаються камерою. Призначає шари вашим об'єктам у Інспекторі.

Projection - перемикає здатність камери моделювати перспективне або отрографічне відображення :

- **Perspective** - камера відображатиме об'єкти з перспективою.
- **Orthographic** - камера буде відображати об'єкти рівномірно, без відчуття перспективи. Відкладена візуалізація не підтримується в орфографічному режимі.

Field of view - кут огляду камери, виміряний в градусах вздовж осі, зазначеної у спадному меню «Вісь FOV». Саме цю властивість ми будемо змінювати для реалізації зменшення та збільшення масштабу.

Clipping Planes - відстань від камери для початку та зупинки візуалізації**.**

- **Near -** найближча по відношенню до камери точка, в якій відбудеться малювання**.**
- **Far -** найдальша точка відносно камери, в якій буде відбуватися малювання.

При умові, що потрібно :

- збільшити масштаб, потрібно збільшити властивість камери **Field of view.**
- зменшити масштаб, потрібно зменшити властивість камери **Field of view.**

Для реалізації поставленої задачі було стоверно зміну, яка відповідає з якою швидкістюбуде зменшуватись/збільшуватись значення вдастивості **Field of view.**

```
public float ZoomingCoefficient = 0.7f;
```

Демонстрація функції **Zoom()** яка відповідає за налаштування властивості **Field of view**.

```
void Zoom()
{
    Vector2 finger_1 = Input.GetTouch(0).position;
    Vector2 finger_2 = Input.GetTouch(1).position;

    if (frameElapsedCounter != 0)
    {
        distance_active_frame = Vector2.Distance(finger_1, finger_2);

        if (distance_active_frame > distance_lact_frame)
        {
            camer.fieldOfView -= ZoomingCoefficient;
        }
        else if (distance_active_frame < distance_lact_frame)
        {
            camer.fieldOfView += ZoomingCoefficient;
        }
        else if (distance_active_frame == distance_lact_frame)
        {
            //nothing doing
        }
    }
    else
    {
        frameElapsedCounter++;
    }

    distance_lact_frame = distance_active_frame;
}
```

Змінні які використовуються:

```
private int frameElapsedCounter;
```

Починає збільшуватись, при першому проходженню, за для уникнення випадкових дотиків користувача, що можуть призвести до різких змін масштабу. При закінчені дій користувача з використанням двох дотиків ця зміна обнуляється.

```
private float distance_active_frame;
```

Записується відстань між поточними дотиками користувача.

```
private float distance_lact_frame
```

Записується відстань між дотиками користувача, які були збереженні певну кількість кадрів перед поточними дотиками.

Vector2.Distance – знаходить відстань між поточними дотиками, координати яких були збереженні у векторах з нульовою довжиною **finger_1, finger_2** відповідно.

§15 Повертання видимих частин карти в початкове положення

При потребі відобразити карту до моменту, коли користувач почав обертати карту, користувачу необхідно натиснути на кнопку **align position.**

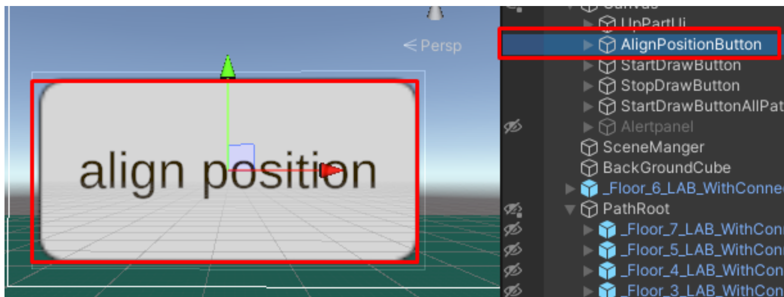

Рис.86

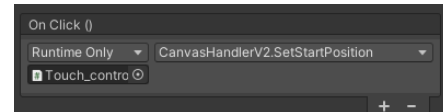

Рис.87

При натисканні викликається функція **SetStartPosition():**

```
public void SetStartPosition()
{
    var main = Camera.main;

    main!.transform.position = new Vector3(7, 4, 0);
    main.fieldOfView = 129.5f;

    pathRoot.transform.position = new Vector3(22, -247, 12);
    pathRoot.transform.rotation = Quaternion.Euler(0, 0, 0);
}
```

Всі видимі дочерні об’єкти, що належать об’єкту **Path.Root**, переміщуються в початкове положення, разом зі своїм батьківським об’єктом. При цьому повертаючи їх в початкове положення відносно поворотів по осям.

Для використання поворотів **Unity** клас **Quaternion**, в якому описані всі базові дії з кватерніонами.

Для камери ми присвоюємо стартове значення поля **Field Of View.**

Розділ V. Проміжні результати практичної частини.

diplom.3.0.0.apk – файл для встановлення додатку займає 20.00 МБ.

Рис.85

Diplom – назва додатку. Сам додаток займає в загльному лише 46,74 МБ.

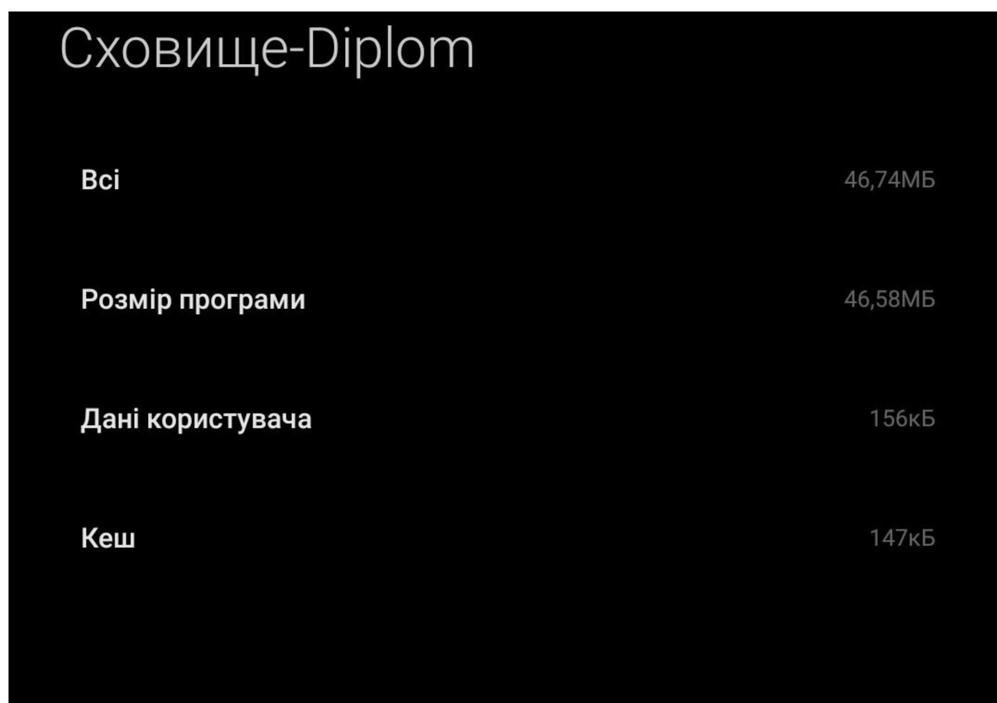

Рис.86

Застосунок можна встановити перейшовши за посиланням в додатку А.

Мінімальній рівень прошивки андроїд має бути Android 5.1 ‘Lollipop’

Рис.87

Виконана робота

Під час виконання кваліфікаційної роботи бакалавра використовуючи програмне забезпечення Unity, Blender та Rider IDE, було розроблено додаток для побудови маршруту та декілька корисних алгоритмів для полегшення задачі моделювання поверхів, побудови маршруту.

Висновки

- Порівнюючи Unity Engine з Android Studio можна сказати, що переваги Unity у відкритості документації та легкості налагодження проекту. Якщо порівнювати з AuthoCad, то гнучкіший вибір технологій, що можна використовувати при створенні проекту. Також більш розвинена база з програмної розробки, а не лише моделювання.
- Повноцінність застосунків і простота інтерфейсу Unity Engine дозволяє працювати над створенням проекту людям, які навіть не мають технічної освіти.
- Даний проект легко підтримувати та розвивати. Для цього достатньо скачати папку даного проекту з репозиторію Github та прочитати дану роботу.

Перелік посилань

1. UnityEngine.CoreModule.Bounds .
   URL: https://docs.unity3d.com/ScriptReference/Bounds.html

2. Unity documentation Mesh Renderer component.
   URL: https://docs.unity3d.com/Manual/class-MeshRenderer.html

3. Unity documentation GameObject.FindWithTag.
   URL: https://docs.unity3d.com/ru/2019.4/ScriptReference/GameObject.FindWithTag.html

4. Unity documentation Camera component.
   URL: https://docs.unity3d.com/Manual/class-Camera.html

5. Unity documentation GameObject.activeSelf.
   URL: https://docs.unity3d.com/ru/2019.4/ScriptReference/GameObject-activeSelf.html

6. Unity documentation UnityEngine.InputLegacyModule.
   URL: https://docs.unity3d.com/ScriptReference/Input.html

7. Unity documentation TouchPhase.
   URL: https://docs.unity3d.com/ScriptReference/TouchPhase.html

8. Unity UI documentation Canvas Scaler.
   URL: https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/script-CanvasScaler.html#hints

9. “Мой НГУ”.
URL: https://apps.apple.com/my/app/%D0%BC%D0%BE%D0%B9-%D0%BD%D0%B3%D1%83/id1582757183

Додаток А

Посилання на APK на GoogleDrive .

URL:
https://drive.google.com/drive/folders/1sH9cwNINOFE6NrccJC_0pTpFiC04AyDB?usp=sharing

Посилання на GitHub репозиторій

https://github.com/Xergofian/Diplom/tree/master