МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Інформаційна система Audiences of SSU
на програмній платформі Unity»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Коробов А.Г.**

Студента групи ІН – 62 **Бабко О.В.**

СУМИ 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2020 р.

ЗАВДАННЯ

до випускної роботи

Студентки четвертого курсу, групи ІН-62 спеціальності "Інформатика" денної форми навчання Бабко Олени Вікторівни.

Тема: " Інформаційна система Audiences of SSU на програмній платформі Unity"

Затверджена наказом по СумДУ

№_____ от _____2020 р.

Зміст пояснювальної записки: 1) аналіз проблеми та постановка задачі; 2) метод пошуку оптимального шляху; 3) розробка інформаційного і програмного забезпечення системи

Дата видачі завдання "_____"_____2020 р.

Керівник випускної роботи _____ Коробов А.Г.

Завдання прийняв до виконання _____Бабко О.В.

РЕФЕРАТ

Структура та обсяг випускної роботи. Випускна робота складається із вступу, трьох розділів, висновків, списку використаних джерел і додатків. Загальний обсяг роботи становить 71 сторінку друкованого тексту, з яких анотація – на 1 стор., зміст – на 1 стор., перелік умовних позначень – 1 стор., основний текст – на 36 стор., список із 14 використаних джерел – на 2 стор., додатки – на 27 стор. Робота містить 4 діаграми, 1 таблиця та 47 рисунків.

Об'єкт дослідження. процес проектування та розробки інформаційних систем пошуку оптимальних маршрутів засобами платформи Unity3D.

Мета роботи. розробка алгоритмічного та програмного забезпечення інформаційної системи пошуку, побудови та 3d візуалізації оптимальних маршрутів руху засобами програмної платформи Unity3D за визначених умов функціонування.

Методи дослідження базуються на принципах і методах теорії графів під час розроблення алгоритмів пошуку оптимальних маршрутів руху, технологій моделювання та 3d графіки під час реалізації програмної платформи.

Результати — Виконано аналітичний огляд сучасного стану мобільних додатків пошуку, побудови та візуалізації оптимального маршруту пересування із місця знаходження до місця призначення за умов функціонування всередині приміщень. Виконано аналітичний огляд алгоритмів пошуку найкоротшого шляху на базі методів теорії графів. Розроблено та програмно реалізовано алгоритм пошуку оптимальних маршрутів Розроблено та реалізовано дизайн та 3d візуалізацію оптимального шляху у мобільному додатку за допомогою програмної платформи Unity3D та мови C#.

Ключові слова: інформаційна система, теорія графів пошуку найкоротшого маршруту, unity3d

ЗМІСТ

СПИСОК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

APK (Android Package) – формат архівних файлів-додатків для «Android».

GUI (Graphical User Interface) – графічний інтерфейс користувача.

SoC (System on Chip) – Система на чипі.

Android – операційна система для мобільних пристроїв (планшетів, смартфонів, ноутбуків, електронних книг).

Евристики – сукупність методів пошуку вирішення задачі / спеціальні методи розв’язування задач, які звичайно протиставляються формальним методам розв’язання, що спираються на точні математичні моделі.

Упередження – хибна думка, що складається щодо кого-, чого-небудь наперед, без ознайомлення, та пов’язане з нею відповідне ставлення.

Скрипт – це програма, яка автоматизує деяке завдання, яке без сценарію користувач робив би вручну, використовуючи інтерфейс програми.

Ігровий рушій (Game engine) – це набір систем, інструментів які спрощують роботу написання будь-якого додатку додатку.

Ігровий Ассет (від англ. Game asset) – ресурс, що складається з однотипних даних, що є частиною додатку чи гри. Може зберігатися у вигляді файлу.

Сцена – сукупність графічних моделей розташованих в певному порядку та площині.

Текстура – растрове цифрове зображення, що відтворює візуальні властивості будь-яких поверхонь, матеріалів або об'єктів.

Спрайт – графічний об'єкт, який можна відобразити на екрані.

ВСТУП

Сучасний світ – світ розвинутих технологій. Людина сьогодні не уявляє собі день без мобільного телефону. Сьогодні ці маленькі гаджети не тільки засоби зв'язку та спілкування, але й помічники вдома, на роботі, у дорозі. Усім відомі календарі, онлайн карти, фотокамери, музичні альбоми, перекладачі та багато іншого.

Якщо розібрати мобільний телефон на частини, всередині можна побачити, що він не є складним як персональний комп'ютер. Складається всього з таких частин – дисплей, акумулятор, SoC, внутрішня та оперативна пам'ять, модеми, камера та датчики. Те що видно одразу, це екран телефону, тобто дисплей. Усе, що ми бачимо на екрані, оброблюється та контролюється внутрішніми компонентами. Акумулятор підтримує працездатність смартфону. Найважливішим компонентом слугує – материнська плата з процесором або SoC. Містить у собі не тільки звичайний процесор, але й графічний, LTE-модем, контролер екрана, бездротові адаптери та інше. Та жоден смартфон не може працювати без оперативної пам'яті та системного сховища.

Оскільки мобільні телефони все ще потребують комунікації для здійснення дзвінків, відправлень текстових повідомлень, підключення до всесвітньої павутини internet, їм допомагають у цьому модеми.

Для гарних фотографій та відеозаписів у всіх смартфонів є фронтальна та основна камери. Вони будуть мати різні відтінки, насиченість та контрастність у порівнянні з іншими мобільними телефонами.

У більшості сучасних смартфонів вбудовано декілька основних датчиків: акселерометр (для визначення орієнтації пристрою та його рухів), гіроскоп (для визначення повороту мобільного телефону), цифровий компас (для орієнтації на мапах), датчик освітлення (автоматично встановлює яскравість екрана в залежності від навколишнього світла), датчик наближення (для небажаних дотиків до поверхні екрана).

Сьогодні Android одна з популярних платформ для мобільних засобів. Програмні додатки, створені для операційної системи Android, мають доступ до

великого відкритого торгового майданчика для публікації різних застосунків: Google Play, де їх може знайти будь-який користувач смартфону.

Мобільні додатки створюються не тільки для Android, але й для такої операційної системи як iOS, яку підтримують популярні лінійки смартфонів iPhone від Apple. Якщо за мету узяти розробку програму, яка буде працювати для обох операційних системах, то доведеться використовувати різні комплекти програмного забезпечення. [1]

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Теоретичні аспекти проектування та розробки мобільних додатків

Одні компанії запроваджують щось нове у свої моделі, випускаючи нові смартфони. Інші, намагаються не відставати, створюють та оновлюють мобільні додатки, унікальні продукти і пропонують різні варіації рішень.

Розробка мобільного додатку для таких операційних систем як iOS або Android – це складний виробничий процес, що потребує ретельно спланованих етапів. Підготовка та створення першого прототипу додатка – є важливими частинами проекту. Узгоджений список вимог із замовником, дозволяє переконатися, що ми розуміємо один одного та схожі уявлення про остаточний результат роботи.

Отже, етапи розробки додатків :

1) Ідея;
2) Проектування;
3) Дизайн;
4) Розробка;
5) Тестування;
6) Публікація;

Ідея – це етап, на якому найважливіше отримати узгоджений, із замовником, опис того, що який додаток буде створено. Обговорюється і формулюється ідея, пропонуються оптимальні напрямки реалізації і складається список основних вимог.

Проектування – створення начебто карти з усіма функціями та можливостями продукту, прототипами, які описують усі екрани та меню додатка, схеми переходів між ними. У залежності від потреб, прототипи можуть бути статичними або такими, що взаємодіють з користувачем.

Дизайн – створення дизайну усіх екранів та інтерфейсів користувача майбутнього додатка і відтворюємо різні можливості для всіх сценаріїв

використання. Усі елементи користувацького інтерфейсу ми досліджуємо на частоту користувань, щоб упевнитися, які прийняті дизайн-рішення є зручними та дають змогу користувачу ефективно вирішувати поставлені ним завдання.

Розробка – це безпосередньо сам процес створення програмного продукту для мобільних пристроїв. Налічує декілька етапів, на яких замовник долучається до процесу створення програмного продукту. Додаткові файли додатку періодично відправляються замовнику для ознайомлення та подальшого узгодження.

Тестування – перевірка відповідності до ТЗ за допомогою тестування. Додаток встановлюється на пристрої тестування, і працює зовсім як якщо це було завантажено з таких торгових майданчиків як Google Play або AppStore. Саме на цьому етапі виправляються усі незбіжності й знайдені при тестуванні проблеми.

Публікація – передача готового програмного продукту замовнику, для подальшої можливості викладення у магазини Google Play або AppStore і користування усіма верствами населення. Кожен додаток підлягає ретельній перевірці перед публікацією. Цим займаються спеціальні команди відповідних мобільних платформ. Після успішної публікації при потребі додаток переходить у стадію технічної підтримки. Саме цей етап переглядає й оцінює старий та новий функціонал, складає плани майбутнього розвитку та удосконалення [2].

1.2 Огляд відомих програмних рішень для задач побудови оптимального маршруту руху

MAPS.ME. Це один з найкращих додатків, що використовують офлайнову навігацію. З кожним оновленням у ньому з'являються нові цікаві функції, і на сьогодні MAPS.ME чудово вміє прокладати маршрути, володіє високою та якісною деталізацією мап, відмінною працездатністю. Даний додаток необмежений на кількість мап для завантаження - ви можете мати хоч усю земну півкулю, якщо місткість накопичувача смартфона зможе це дозволити (рис 1.1).

Застосунок дозволяє визначити місце розташування користувача з використанням технології GPS, прокласти маршрути для автомобілів і пішоходів

між будьякими двома пунктами шляху (також між різними областями та країнами), проводити пошук за назвою чи описом об’єктів, а також надавати детальну інформацію про різні заклади, відображені на мапі. Також для користувачів є доступною голосова навігація.

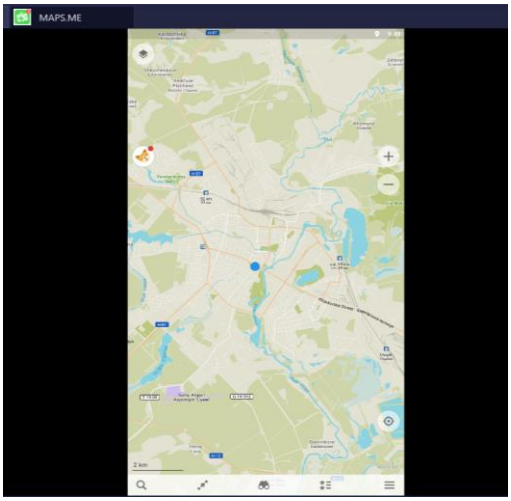

Рисунок 1.1 – головна сторінка додатку MAPS.ME

Google Maps. Додаток розроблений компанією Goggle. Має схожий функціонал у порівнянні з MAPS.ME, при цьому важливою перевагою є можливість його використання в офлайновому режимі. Завантажувати ділянки карт в цьому додатку можна було і раніше, а зовсім нещодавно розробниками була додана повноцінна навігація і пошук об'єктів без підключення до інтернету. Так що тепер Google Maps можна сміливо використовувати в тих місцях, де недоступне підключення до Мережі (рис 1.2).

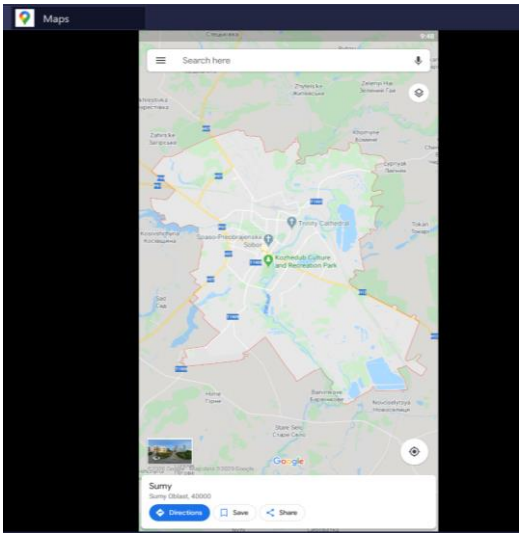

Рисунок 1.2 – головна сторінка додатку Google Maps

HERE WeGo Maps. Він досить функціональний, повністю безкоштовний, але має застарілий інтерфейс користувача. Однак, на інформативність карт і швидкість роботи програми це ніяк не впливає - користуватися HERE Maps дуже зручно. До того ж це додаток вміє прокладати маршрути не тільки для пішоходів і автомобілістів, а й для пасажирів громадського транспорту [3]. Приклад роботи наведено на рисунку 1.3.

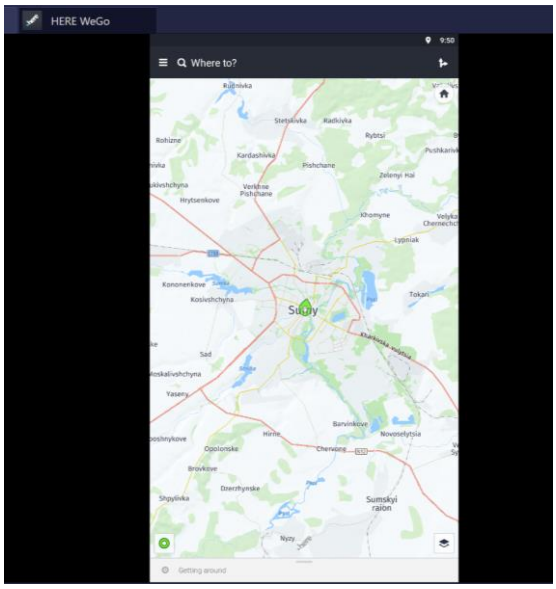

Рисунок 1.3 – головна сторінка додатку Google Maps

Розклад СумДУ та Мій університет. Android додатки, розроблені в Сумському державному університеті для студентів та викладачів з актуальним розкладом занять. Основні особливост даних додатківі:

- Можливість завантаження розкладу на увесь семестр;
- Пошук по групах, викладачам та аудиторіям;
- Перегляд розкладу без доступу до інтернету;

1.3 Аналіз алгоритмів пошуку найкоротшого шляху

Переважна більшість додатків та ігор використовують рішення для набору маршрутів на основі алгоритму «A*». Хоча цей алгоритм ефективний та його можна легко реалізувати, він не може працювати безпосередньо з даними рівня гри. Це вимагає, щоб рівень гри був представлений у певній структурі даних:

спрямований невід’ємний зважений графік. Алгоритм Дейкстри використовується для прийняття тактичних рішень, ніж проходження маршруту, і це найпростіший варіант алгоритму «A*».

Ні «A*», ні Дейкстра алгоритми не можуть працювати над геометрією, що складає ігровий рівень. Вони покладаються на спрощену версію рівня, яка повинна бути представлена у вигляді графа.

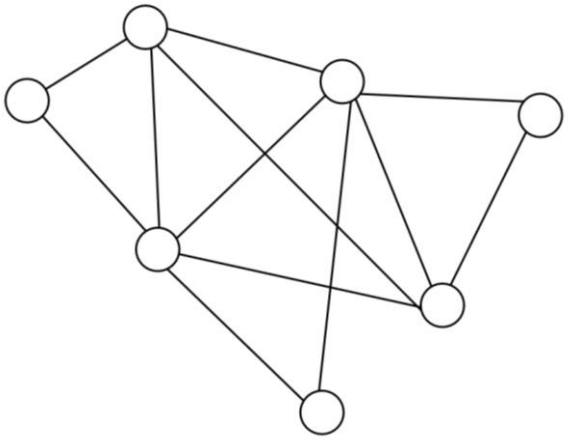

Рисунок 1.4 – структура графа

Граф – це математична структура, яка часто представлена схематично. Складається з двох різних типів елементів: вузлів, які часто зображені у вигляді точки або кола, та зв’язків, які з’єднують вузли між собою. На рисунку 1.4 показана структура графа.

Для пошуку маршрутів кожен вузол зазвичай представляє деяку площину ігрового рівня, кімнату, коридор, платформу, галявину або якусь область зовнішнього середовища. Зв’язки показують, які локації під’єднані. Якщо кімната прилягає до коридору, то вузол, який представляє кімнату, матиме з’єднання з вузлом коридору. Таким чином увесь рівень гри розділений на окремі території, які з’єднані між собою.

Щоб дістатися з одного місця в інше, ми використовуємо з’єднання, зазначені вище. У більшості випадків нам доведеться використовувати з’єднання з проміжними вузлами.

Зважені графіки.

Зважений графік складається з вузлів і з’єднань, як і звичайний графік. Окрім них для кожного з’єднання, ми додаємо числове значення. У теорії графів це називається вагою, а в ігрових додатках – вартістю. На рисунку 1.5 зображений зважений графік.

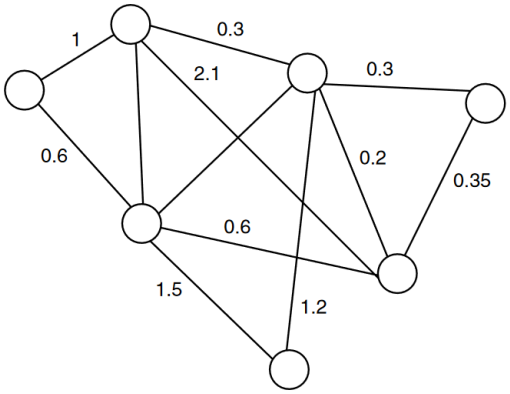

Рисунок 1.5 – структура зваженого графа

Витрати в графі часто представляють як час або відстань. Якщо вузол однієї області, знаходиться на великій відстані від вузла в іншій області, то вартість з’єднання буде великою.

Для повного маршруту через граф, від вузла початку до вузла кінця, ми можемо підрахувати загальну вартість шляху. Це сума витрат кожного з’єднання в маршруті. Розглянемо деякий маршрут на рисунку 1.6. Якщо ми будемо рухатися від вузла A до вузла C, також пройдемо через проміжний вузол B, то загальна вартість маршруту складатиме 9.

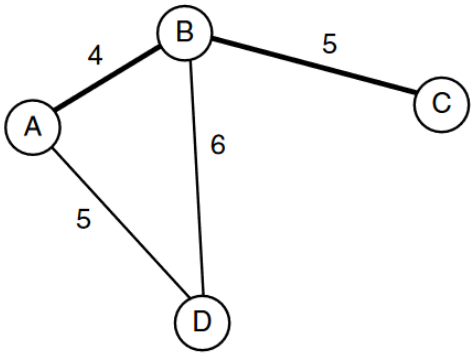

Рисунок 1.6 – Загальна вартість графу

Вузли-представники на певних територіях.

Якщо дві області з’єднані, то відстань між ними буде нульовою. Для прикладу візьмемо коридор та кімнату. Якщо ви стоїте у дверях, то переміщення з кімнату в коридор буде миттєвим. Дуже часто вартість відстані вимірюють від репрезентативної точки або представника (вузла, який визначає певну територію). Отже, вибираємо центр кімнати та центр коридору та з’єднуємо ці вузли. Приклад такого вигляду графу показаний на рисунку 1.7.

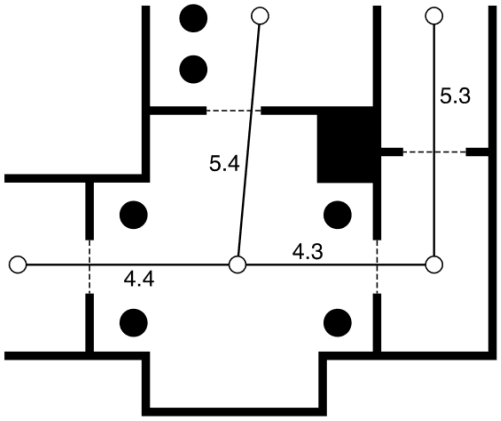

Рисунок 1.7 – схема зваженого графу з вузлами-представниками

Невід’ємні обмеження.

Можливо, для алгоритмів пошуку найкоротшого шляху, немає сенсу мати негативні витрати. Ми не зможемо мати від’ємну відстань між точками як і нам не знадобиться від’ємна кількість часу на переміщення. Теорія графів дозволяє мати негативну вагу. Алгоритми, які можуть працювати з від’ємною вагою, складніші. Зокрема, алгоритми Дейкстра та A * слід використовувати з від’ємною вагою. Також для них можна побудувати граф з негативними вагами, який зможе навіть повернути розумний результат. Але у більшості випадків ці алгоритми переходять у нескінченність. І це не є помилкою. Математично, не має такого поняття як найкоротший шлях у графах з від’ємною вагою, рішення просто не існує. Вартість шляху завжди позитивна.

Направлені графи.

Основні алгоритми прокладання найкоротших маршрутів використовують складнішу форму графа, таку як спрямований граф, який зображений на рисунку

1.8. Раніше можна було сказати, якщо можливе переміщення від A до B, то також і від B до A. Вартість такого шляху буде однаковою в обох напрямках. Натомість цей граф припускає, що з’єднання йдуть лише в одному напрямку. Якщо ми будемо рухатися від A до B і навпаки від B до A, то вийде два різних з’єднання.

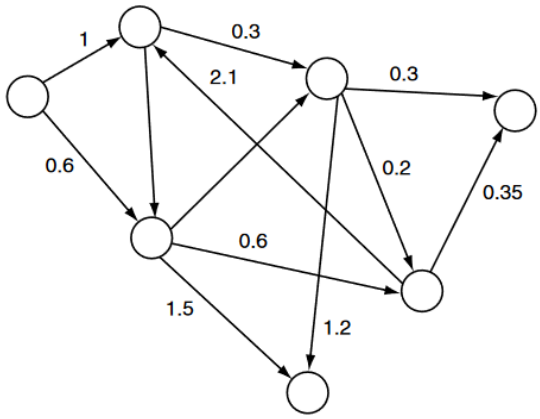

Рисунок 1.8 – схема спрямованого зваженого графу

Не завжди буває, що відстань від A до B означає, що точку B можна досягти від A. Розглянемо приклади. Якщо A другий поверх якоїсь будівлі, а вузол B – перший, то гравець зможе легко зістрибнути з вузла A на вузол B, але не зможе так же швидко повернутись назад до вузла A. Це означає, що ці два з’єднання мають різні витрати. Додамо сходи між поверхами. Якщо подивитися на витрати в часі, то для того щоб зістрибнути майже не потрібно часу, а для підйому по сходах – усього декілька секунд. Отже, тепер з’єднання від A до B має невелику вартість, а від B до A трохи більшу.

Алгоритм Дейкстра.

Алгоритм Дейкстра названий на честь математика Едсгера Дейкстра, який його розробив. Спочатку цей алгоритм слугував вирішенням математичної теорії графів. Якщо в іграх для проходження деякого маршруту створювали одну початкову та кінцеву точки, то для цього алгоритму можна було мати лише початкову, бо він знаходить найкоротшу відстань до будь-якої точки.

Якщо маємо граф та два вузли, що є початковою та кінцевою точками, ми хочемо отримати шлях, який буде мінімальних з усіх можливих між цими двома вузлами. Той маршрут, який ми отримаємо буде складатися із з’єднань, а не вузлів, тому нам важливо, які саме вузли будуть використані.

Отже, алгоритм Дейкстра поширюється від точки початку, тобто стартового вузла, уздовж своїх з'єднань. Коли за кінцеву точку беруть найбільш віддалений вузол, він записує напрямок, по якому він йшов. Як тільки цільовий вузол буде досягнуто, алгоритм попрямує назад до початкової точки, щоб згенерувати повний маршрут. Робиться це ітераціями. При кожній ітерації він враховує вузол і слідкує за з'єднаннями, які з нього виходять. Для кожного такого зв'язку він знаходить кінцевий вузол і зберігає загальну вартість шляху, який вже пройшов. У першій ітерації, де початковий вузол – це поточний вузол, загальна ціна – це просто вартість з'єднання. Це показано на рисунку 1.9. Для всіх наступних ітерацій, вартість шляху являє собою суму вартості кожного обраного з'єднання. Рисунок 1.10 показує ситуацію після першої ітерації.

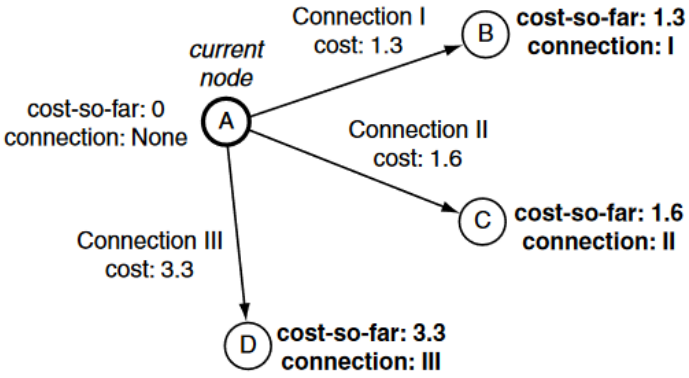

Рисунок 1.9 – схема першої ітерації алгоритму Дейкстра

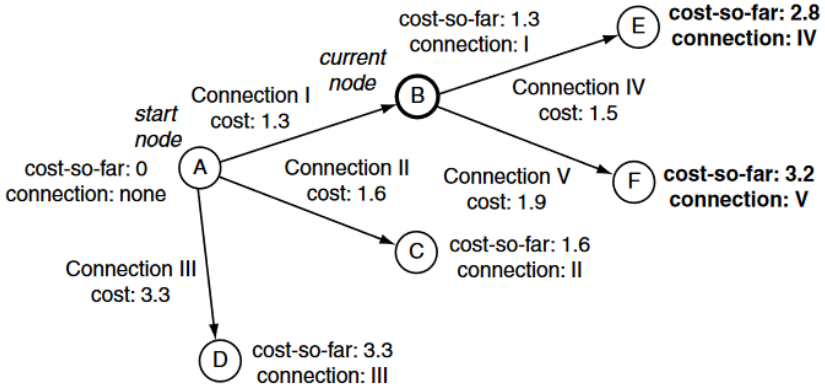

Рисунок 1.10 – схема наступних ітерацій алгоритму Дейкстра

Останнім етапом є пошук шляху. Робиться він починаючи від кінцевої точки та дивлячись на з'єднання, які були використані для прибуття. Алгоритм відстежує зв'язки, поки не буде досягнуто початкового вузла. Отриманий список з'єднань не

в правильному порядку, то ми повертаємо його назад і отримуємо правильне рішення. На рисунку 1.11 показаний простий граф зі знайденим маршрутом.

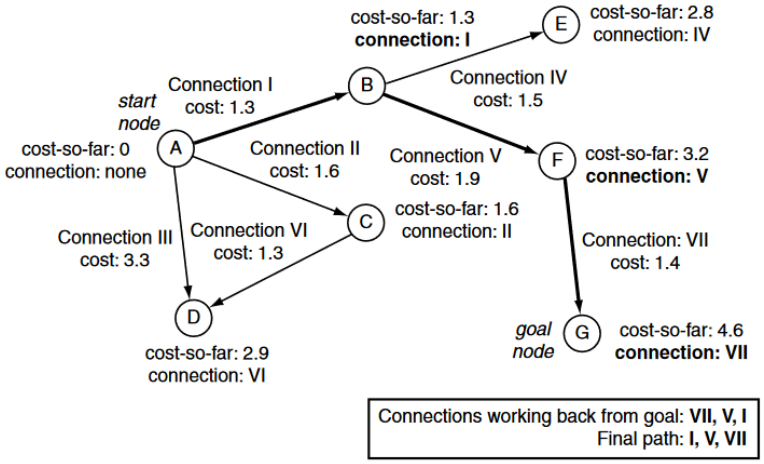

Рисунок 1.11 – схема графу зі знайденим маршрутом

Алгоритм A*.

Шлях в іграх частіше пов’язують з алгоритмом A*. Він простий у виконанні, дуже ефективний, та має багато можливостей оптимізації. Як і алгоритм Дейкстра, A* призначений для точкового визначення шляху і не використовується для вирішення найкоротшого шляху в теорії графів.

Завдання алгоритму таке ж як і в алгоритмі Дейкстра. Є спрямований невід’ємний зважений граф та два вузли (початкова та кінцева точки). Ми б хотіли створити шлях таким, що б загальна витрата шляху була мінімальна серед усіх можливих. Цей шлях повинен складатися зі списку з’єднань.

Алгоритм працює так само, як і Дейкстра. Розглядаючи вузли з найнижчою вартістю, ми вибирали той наступний вузол, який швидше за все, призведе до найкоротшого шляху. Поняття «найімовірніше» контролюється евристикою. Якщо евристика точна, алгоритм буде ефективним. Якщо евристика жахлива, то вона може працювати навіть гірше ніж Дейкстра. Алгоритм A* також працює з ітераціями. При кожній ітерації він враховує один вузол графа і слідкує за його вихідними з’єднаннями. Вузол вибирається за допомогою алгоритму вибору, аналогічного до Дейкстра, але зі значною різницею евристики.

Під час ітерації A* враховує кожне вихідне з'єднання з поточного вузла. Для кожного з'єднання він знаходить кінцевий вузол і зберігає загальну вартість шляху та з'єднання, з якого він прибув у цей вузол. Крім того, алгоритм A* зберігає ще одне значення: оцінку загальної вартості шляху через початковий, поточний і кінцевий вузли. Ця оцінка є сумою двох значень: вартість уже пройденого шляху та наскільки вона далека від цільового вузла. Ця оцінка формується окремим фрагментом коду і не є частиною алгоритму. Ці оцінки називають «евристичним значенням» вузла, і вони не можуть бути негативними. Створення цього евристичного значення є ключовою задачею при використанні алгоритму A*. На рисунку 1.12 зображений граф з вузлами та оцінками.

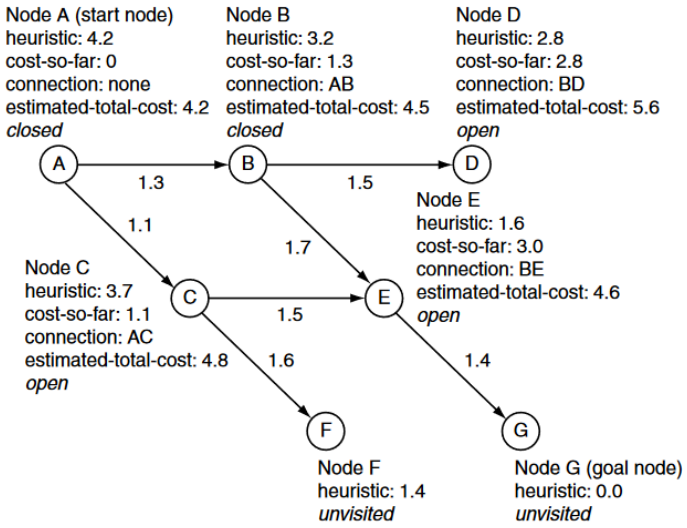

Рисунок 1.12 – алгоритм A* із загальною вартістю

Алгоритм зберігає відкритий список вузлів, які були відвідані, але не оброблені та закритий список вузлів, які були оброблені. Якщо вузол був знайдений на кінці з'єднання, він переноситься у відкритий список. У закритий потрапляють вузли, які пройшли ітерацію. На відміну від попереднього алгоритму, вузол з відкритого списку з найменшою оцінювальною загальною вартістю вибирається для нової ітерації. Ця зміна дозволяє алгоритму досліджувати вузли, які є найперспективнішими. Тобто якщо вузол має найменшу оцінювальну загальну вартість, то він повинен мати порівняно коротку вартість та відстань до кінцевого вузла.

На відміну від алгоритму Дейкстра, алгоритм A* може знайти кращі маршрути до вузлів, які вже є у закритому списку. Якщо попередня оцінка була дуже оптимістичною, то, можливо, вузол був оброблений як найкращий вибір, коли насправді може таким не бути. Отже, якщо такий сумнівний вузол був доданий у закритий список, то це означає, що всі його з'єднання були враховані. Може бути можливим, що весь набір вузлів мав свої значення, що базуються на вартості сумнівного вузла. Оновлення значень цього вузла недостатньо. Необхідним буде також перевірити його з'єднання та оновити значення. Для цього можна вийняти вузол із закритого списку та повернути його у відкритий. Потім він буде чекати, поки усі з'єднання не будуть переглянуті. Будь-які вузли, які покладаються на його значення, також з часом будуть ще раз оброблені. На рисунку 1.13 зображене оновлення закритого вузла. Був прокладений новий маршрут до вузла E через вузол C, бо він є найшвидшим, тому був здійснений перезапис вузла E, і він був переміщений до відкритого списку. На наступній ітерації значення для вузла G буде переглядатися. Так закриті вузли, значення яких були переглянуті, будуть видалені із закритого списку та розміщені у відкритому. Відкриті вузли, які змінили свої значення, залишаються у відкритому списку як і раніше.

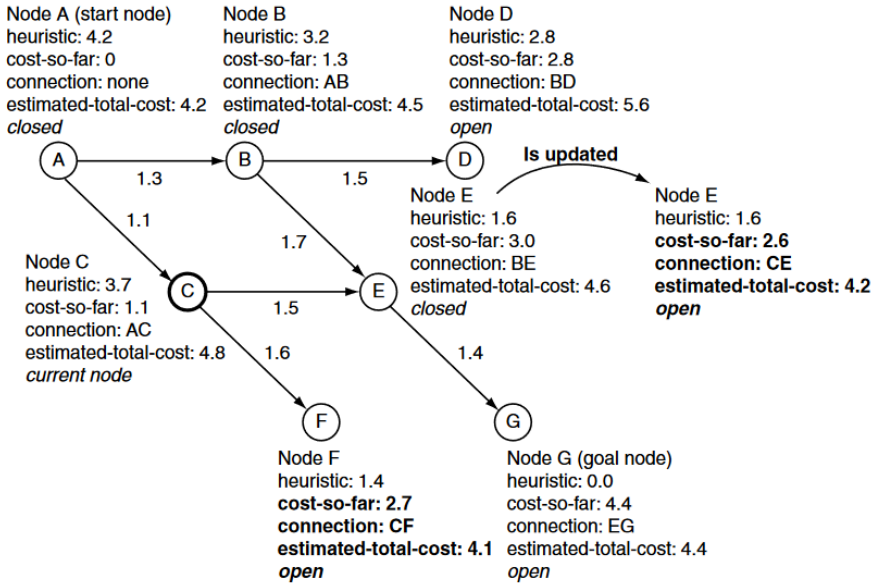

Рисунок 1.13 – схема оновлення закритого вузла в графі

Алгоритм A* закінчується тоді, коли вузли є найменшими вузлами у відкритому списку. Але це не може гарантувати того, що знайдений маршрут є

найкоротшим. Усунуте цю проблему можна, зробивши так, щоб алгоритм закінчувався тоді, коли вузол у відкритому списку з найменшою загальною вартістю (не оцінювальною загальною вартістю) має значення, яке перевищує вартість шляху, який був знайдений раніше. Тоді і тільки тоді, ми можемо гарантувати, що не буде знайдено жодного майбутнього шляху, який буде найкоротшим. Це майже та сама умова припинення роботи алгоритму, як і в алгоритмі Дейкстра. Але теоретично, це може призвести до неоптимальних результатів. Контролювати цю проблему можна за допомогою евристичної функції. Залежно від її вибору, ми можемо гарантувати оптимальні результати або неоптимальні, для швидшого виконання.

Ми отримуємо кінцевий шлях точно таким же чином: починаючи з початкового вузла, накопичуючи з'єднання, аж до кінцевого вузла, та назад. Після отримання усіх з'єднань, знову перевертаємо їх та отримуємо правильний шлях.

Різниця між алгоритмом A* та алгоритмом Дейкстра.

Алгоритм майже ідентичний алгоритму Дейкстра. Він додає особливу перевірку, що визначити, чи потрібне оновлення та видалення вузла із закритого списку. Алгоритм також додає два рядки для обчислення оцінювальної загальної вартості за допомогою евристичної функції та додаткове поле в структурі запису вузла, для зберігання цієї інформації. Існує набір обчислень для отримання евристичного значення зі значень вартості наявного вузла. Це робиться для того, щоб уникнути виклику евристичної функції більше, ніж це необхідно. Якщо у вузла вже було обчислене евристичне значення, то воно буде повторно використане, коли вузол потребуватиме оновлення. Тепер прокладання маршруту повертає запис вузла з найменшим значенням оцінювальної загальної вартості, а не найменше значення витрат.

Вибір евристики.

Чим точніша евристика, тим швидше буде працювати заповнення алгоритму. Якщо можливо отримати ідеальну евристику (ту, що завжди повертає точну мінімальну відстань шляху між двома вузлами), алгоритм A* відразу поверне правильну відповідь: алгоритм стане O(p), де p – кількість кроків на шляху.

Якщо евристика занадто низька, запуск алгоритму A* потребуватиме більше часу. Орієнтована загальна вартість буде упередженою. Тому A* буде вважати за краще переглянути найближчі вузли до початкової точки, а не до кінцевої. Це збільшить час, необхідний для пошуку шляху до кінцевого вузла. Якщо евристична оцінка занижена у всіх можливих випадках, то результатом алгоритму буде найкращій можливий шлях. Він буде таким самим, як і у алгоритмі Дейкстри. Це дозволяє уникнути проблеми з неоптимальними шляхами. Якщо евристика завищена, ця гарантія втрачається. Частіше у додатках, де точність важливіша за продуктивність, важливо переконатися, що евристика недооцінена (занижена).

Евклідова відстань.

Значення витрат для проходження маршруту стосуються відстаней на рівні(мапі) «гри». Вартість з’єднання формується відстанню двох вузлів-представників. У цьому випадку евристикою є евклідова відстань.

Евклідова відстань – це відстань «як ворона летить» («як ворона летить» звикли говорити, коли відстань вимірюється по прямій лінії між двома точками або місцями). Вона вимірюється між двома точками, можливо навіть через стіни або перешкоди. На рисунку 1.14 зображені евклідові відстані. Вартість з’єднання між двома вузлами задається відстанню між вузлами-представниками кожного регіону чи території. Оцінка отримується відстанню до репрезентативної точки вузла, навіть якщо прямого зв’язку немає.

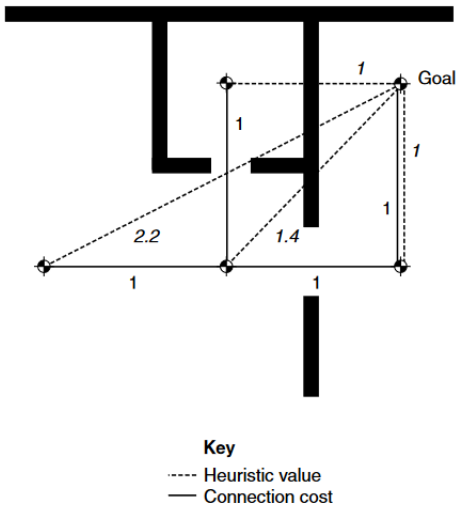

Рисунок 1.14 – схема евклідових відстаней

Кластерна евристика.

Евристика кластера працює за допомогою групування вузлів у кластери. Вузли в кластері представляють деяку область рівня, яка взаємозв’язана. Кластеризація частіше є ручною або побічним продуктом рівня. Створюється таблиця, яка дає найменшу довжину шляху між кожною парою кластерів. Цей етап обробляється в офлайн режимі та вимагає безлічі випробувань прокладання шляху між усіма парами. Згодом вибирається невеликий набір кластерів.

Коли в грі викликається евристика, якщо вузли початку та кінця знаходяться в одному кластері, тоді для отримання результату використовується евклідова відстань. В іншому випадку оцінка розглядається в таблиці. На рисунку 1.15 зображені групи кластерів, де кожне з’єднання має однакову вартість у будь-якому напрямку.

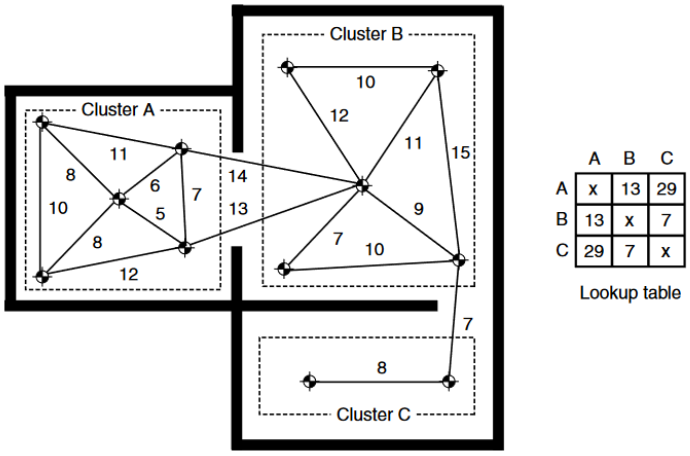

Рисунок 1.15 – схема кластерної евристики

Евристичний кластер часто різко покращує продуктивність пошуку маршруту у закритих приміщеннях на евклідовій відстані, оскільки він враховує складені маршрути, які пов’язують місця, що знаходяться поряд (відстань через стіну може бути невеликою, але маршрут для проходження між кімнатами може залучати безліч коридорів та проміжних ділянок).

Однак, якщо всі вузли кластера мають однакове евристичне значення, алгоритм A* не може легко знайти найкращий шлях через кластер. Але кластеризацію варто пробувати для рівнів з внутрішніми приміщеннями.

На рисунку 1.16 показані приклади заповнення алгоритму з різною евристикою на основі плитки. Перший приклад використовує евристичний кластер, другий застосував евклідову відстань, третій має нульову евристику (як якби ми використали алгоритм Дейкстри). Дивлячись на ці приклади зрозуміло, що з кожним наступним заповнення зростає (зліва на право); евристика кластера має дуже мало заповнення, тоді як нульова евристика заповнює більшу частину.

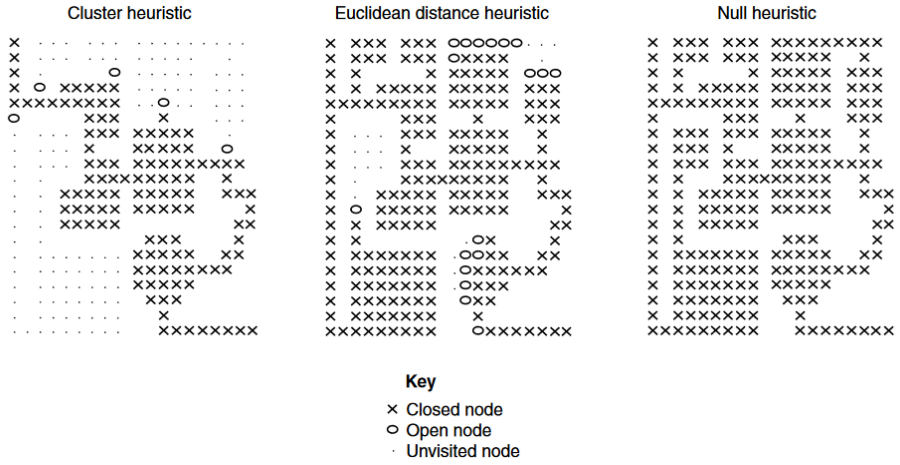

Рисунок 1.16 - схеми заповнення алгоритму у приміщенні

Отже, можна сказати, що алгоритм Дейкстра є підмножиною алгоритму A*. У A* ми обчислюємо оцінювальну загальну вартість вузла, додаючи евристичне значення. Тоді алгоритм вибирає вузол для обробки на основі цього значення. Якщо евристика завжди повертає 0, то оцінювальну загальна вартість завжди буде дорівнювати поточній вартості, що є ідентичним до алгоритму Дейкстри [4].

1.4 Постановка задачі

У результаті аналізу різних додатків зі схожою тематикою та потреб студентів, була виділена мета цієї роботи - розробка алгоритмічного та програмного забезпечення інформаційної системи пошуку, побудови та 3d візуалізації оптимальних маршрутів руху засобами програмної платформи Unity3D за визначених умов функціонування. Як базову мапу території на якій виконується офлайн навігація запропоновано використовувати мапу корпусів та поверхів Сумського державного університету. Визначена задача розробки потребує

розв’язання низки інженерно-практичних задач пов’язаних з розробкою мобільних додатків, таких, як:

1) Розробити проект графу, з точками – аудиторіями, для обраного алгоритму.

2) Розробити дизайн користувацького інтерфейсу, виділити його основні функції та можливості.

3) Програмно реалізувати алгоритм у 2D та 3D площинах та можливість користувача, самому вибирати потрібну аудиторію та спостерігати за прокладенням маршруту.

4) Реалізувати можливість вибору режиму роботи додатку у різних проекціях відображення (2D чи 3D) в залежності від апаратно-програмних вимог цільової платформи функціонування.

5) Виконати локалізацію інтерфейсу користувача програмної системи на різні мови, за для полегшення користування іноземними студентами.

РОЗДІЛ 2
МЕТОД ПОШУКУ ОПТИМАЛЬНОГО ШЛЯХУ

2.1 Основні компонети системи пошуку оптимального шляху

Для додатка необхідно реалізувати систему меню, яка дозволить вибирати корпус та аудиторію, і запускати мапу, повертатися до головного меню, чи виходити з додатка.

Головне меню буде складатися з таких частин:

1) Розкривні списки з опціями вибору корпусу та аудиторії (рис. 2.1);
2) Кнопка додаткового меню (рис. 2.2);
3) Пошук, завантажує мапу с вибраними опціями (рис. 2.3);

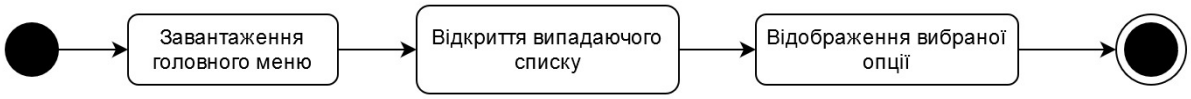

Рисунок 2.1 – Діаграма станів – Розкривні списки

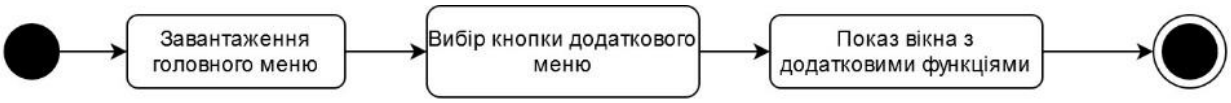

Рисунок 2.2 – Діаграма станів – Кнопка додаткового меню

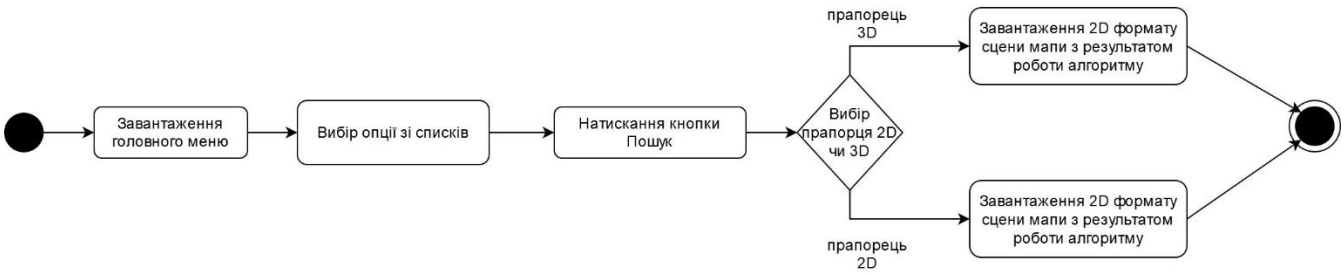

Рисунок 2.3 – Діаграма станів – Пошук та завантаження мапи

Додаткове меню, яке може бути використане користувачем має кнопки Продовжити та Вихід з додатка (рис. 2.4)

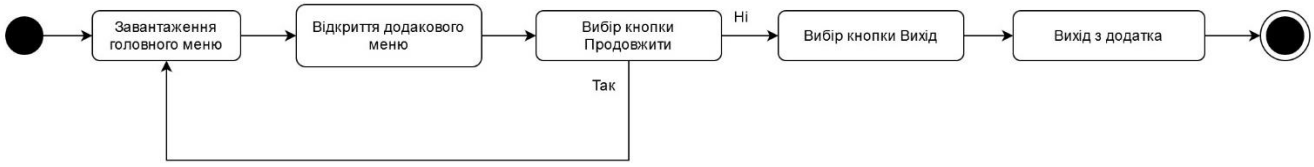

Рисунок 2.4 – Діаграма станів – Вікно додаткового меню

Для полегшення виконання завдання, було розроблено шаблон усіх екранів додатку: стартовий екран, головне меню та сцени з мапами (рис.2.5).

Рисунок 2.5 – шаблон інтерфейсу користувача

Splash screen або стартовий екран – це екран, що з’являється, після запуску будь-якого додатку на мобільному пристрої. Splash – з англійської перекладається як “сплеск”, тому, що ці екрани частіше з’являються на відносно короткий час і зникають. Традиційно на них зображують логотип і назву компанії чи програми, у якій було створено додаток, наприклад Unity, назву додатку зі смугою завантаження, тощо. Коли завантаження додатку завершиться, буде здійснений перехід на головний екран, де можна виконати дії.

2.2 Алгоритм «A*» для пошуку оптимального шляху

Головним завданням додатку буде зображення найкоротшого маршруту, до вибраної користувачем аудиторії, на мапі – 2D чи 3D моделі університету. Для цього за допомогою попереднього аналізу був обраний алгоритм A*. Основою даного алгоритму є зважений невід'ємний граф. Він складатиметься з вузлів, які є представниками аудиторій і кінцевими точками, та з'єднань. Щоб реалізувати даний алгоритм у додатку був підключений ассет з Asset Store (рис. 2.6).

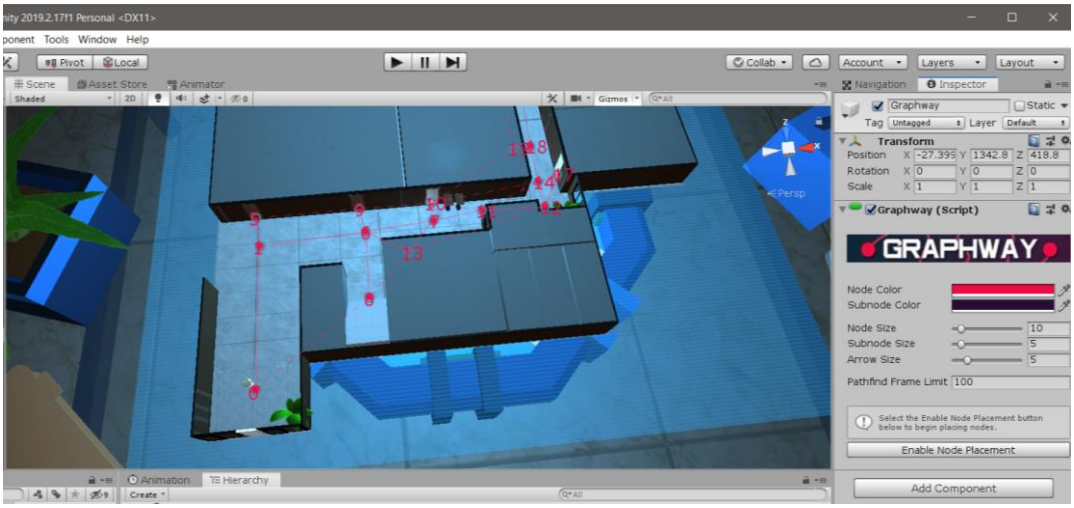

Рисунок 2.6 – граф на 3D мапі

У результаті було отримано n-ну кількість точок та з'єднань у вигляді ігрових об'єктів. Деякі зв'язки були приховані, оскільки проходять через інші об'єкти. На рисунку 2.7 та 2.8 зображена ієрархія об'єктів у сцені Unity.

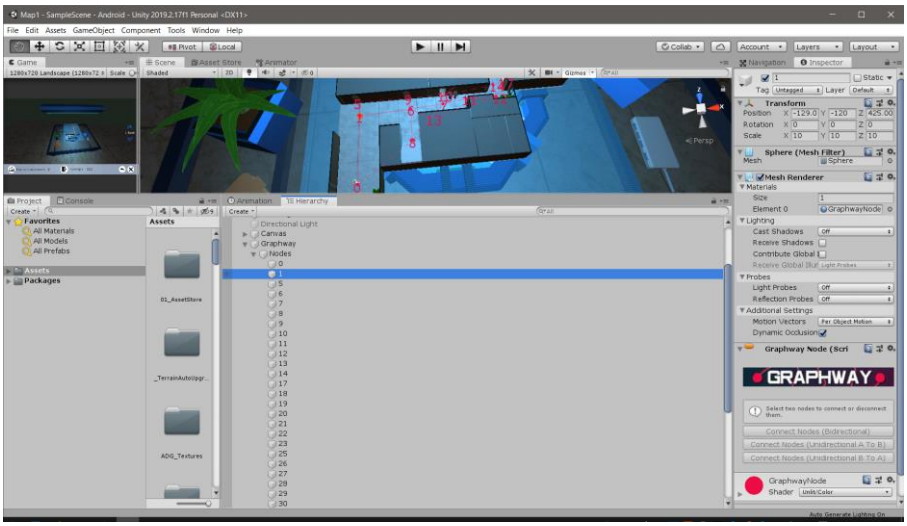

Рисунок 2.7 – ієрархія вузлів на сцені

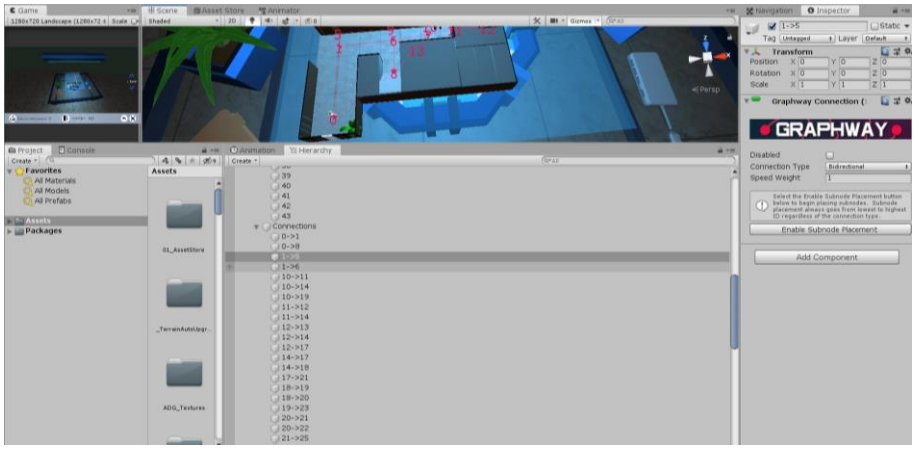

Рисунок 2.8 – ієрархія з’єднань на сцені

Список, у вигляді структури, з номерами аудиторій та їхніми координатами у просторі було реалізовано в скрипті та під’єднано до даного асету.

Псевдокод алгоритму

Як було сказано вище, той що буде шукати маршрут, приймає на вхід граф, початковий та кінцевий вузли. Він також вимагає об’єкт, який може генерувати оцінки вартості для досягнення мети з будь-якого вузла. У коді цей об’єкт є евристичним. Функція повертає масив об’єктів з’єднання, що представляє являє собою шлях від стартового вузла до кінцевого.

На вхід: А граф.

Вихід: А шлях між початковим та кінцевим вузлами.

1: повторення

2: Вибираємо $n_{\text{найкр.}}$ з O таке, що $f\left(n_{\text{найкр.}}\right) \leq f(n), \forall n \in O$.

3: Видаляємо $n_{\text{найкр.}}$ з O і додаємо до C.

4: Якщо $n_{\text{найкр.}} = q_{\text{мети}}$, ВИХІД.

5: Збільшуємо $n_{\text{найкр.}}$: для всіх $x \in Star\left(n_{\text{найкр.}}\right)$, що не знаходяться у C.

6: Якщо $x \notin O$ тоді

7: додаємо x до O.

8: Інакше Якщо $g\left(n_{\text{найкр.}}\right) + c(n_{\text{найкр.}}, x) < g(x)$ тоді

9: Оновити зворотний покажчик x від точки до $n_{\text{найкр.}}$

10: Кінець Якщо

11: Доти, поки O буде пустим.

РОЗДІЛ 3
ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Вибір середовища розробки

Існує велика кількість програм та інструментів, які допомагають створювати мобільні додатки.

Програми, що дозволяють створити мобільний додаток [5]:

- Android Studio – програмне середовище розробки мобільних додатків для операційної системи Android на основі IntelliJ IDEA;
- Android IDE – інтегроване середовище розробки під Android, основане на Eclipse;
- Intel XDK – дозволяє легко розробити крос-платформні мобільні додатки;
- Visual Studio – це велике середовище для розробки додатків під Windows та Windows Phone;
- Unity - багато платформовий інструмент для розробки дво- та тривимірних додатків та ігор, що працює на операційних системах Windows і OS X;
- Photoshop – дозволяє створити зображення та логотипи їх у форматі для web.

Засоби розробки додатків:

Unity Asset Store - це зростальна бібліотека ассетів. Unity Technologies і члени спільноти створюють ассети і публікують і в цьому магазині. Типи ассетів варіюються від текстур, анімацій і моделей до прикладів закінчених проектів, навчальних матеріалів і розширень редактора. Багато ассетів надаються безкоштовно, тоді як інші доступні за помірними цінами; всі ці ассети ви можете завантажувати безпосередньо в свій проект Unity.

Розглянемо переваги та недоліки ігрової платформи Unity. Отже, він має таки переваги:

1) Простота і продуктивність;

2) Велика кількість підтримуваних мобільних та комп’ютерних платформ, таких як Android, iOS, Windows, PS4, Xbox One, Mac &Linux;

3) Гарна продуктивність як слабких так і сильних проектах;

4) Підтримка 2D та 3D режимів;

5) Написання скриптів на мові C# або JavaScript;

6) Вбудований візуальний редактор;

7) Assets store, спеціальний магазин для розробників.

З недоліків можна виділити – не звичний і дещо складний візуальний редактор. Другим недоліком є відсутність повного програмного або ігрового циклу, кожен об’єкт може мати свої скрипти, компоненти и тп.

3.2 Програмне проектування графічного інтерфейсу користувача

Для створення стартового екрана та головного меню в ігровій платформі Unity було використано Canvas, який є головним елементом сцени й всі інші елементи – кнопки, тексти, прапорці, списки, панелі, створюються як дочірні до нього. Canvas має вигляд великого прямокутника, який дозволяє з легкістю керувати дочірніми елементами. Для завдання було створено 3 розкривні списки з вибором корпусів, аудиторій, ролей, 2 прапорці для керування форматом наступної сцени, 2 кнопки : кнопка Пошук, що зберігає та передає дані зі сцени на якій розміщується та завантажує наступну сцену; кнопка додаткового меню, яка відкриває невелике меню з кнопкою повернення до головного меню та кнопкою виходу з додатка. На рисунку 3.1 можна побачити проектування початкового екрана. Він складається з двох малюнків, тексту та анімації.

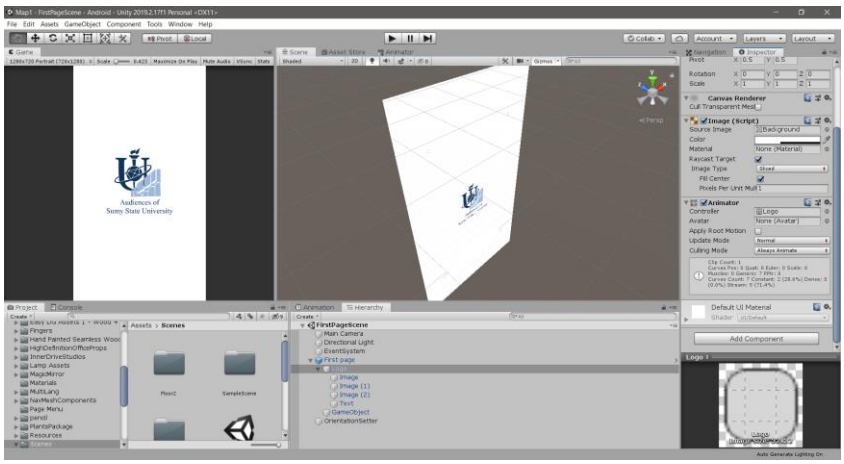

Рисунок 3.1 – проект стартового екрану у програмній платформі Unity

Та на рисунку 3.2 зображене головне меню майбутнього додатка.

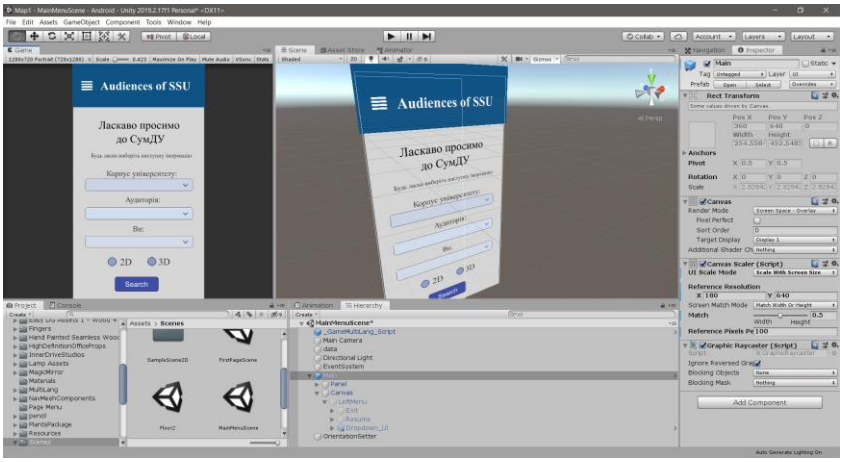

Рисунок 3.2 – головне меню додатка

Після створення головних об’єктів меню необхідно прив’язати певні дії. Для цього створено скрипт з функціями записування у змінну, вибраних опцій з розкривних списків, вибору мапи при певному виборі прапорця та порівнюванням вибраної аудиторії з вже наявними у програмному коді, відкриття додаткового меню.

Для gui або графічного інтерфейсу користувача на сценах з мапами було застосовано panel, оскільки цей об’єкт UI має здатність змінювати розмір в залежності від роздільної здатності екрану мобільного пристрою. Створені кнопки повернення до головного меню та виходу з додатка, і інформаційне поле з іконками та текстом, для нагадування користувачу про нещодавній вибір пошуку (рис. 3.3).

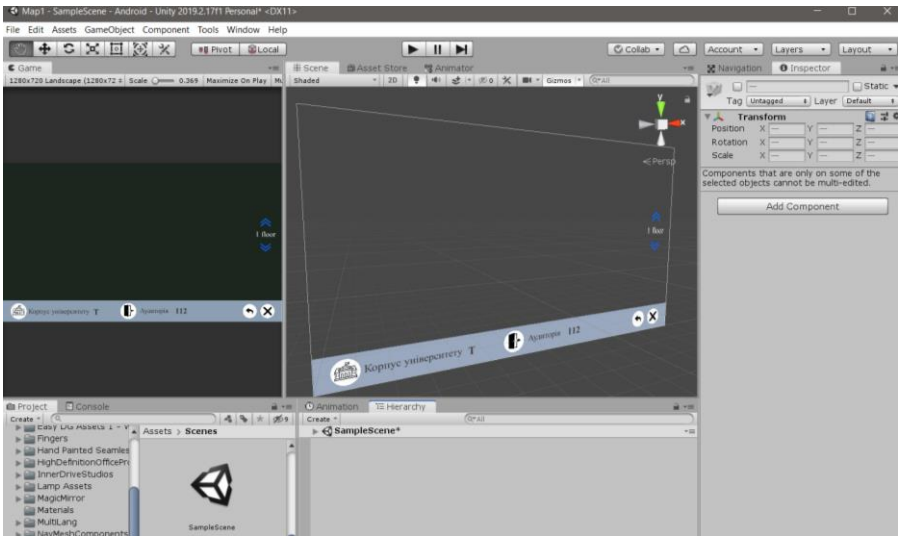

Рисунок 3.3 – Допоміжний інтерфейс для сторінки с мапою

3.3 Створення внутрішнього середовища мапи

Локація з моделлю мапи виконана у двох стилях: для 2D моделі – детективний стиль, для 3D – стиль майбутнього. Для цього використано спеціальні пакети ассетів з об’єктами (рис. 3.4)

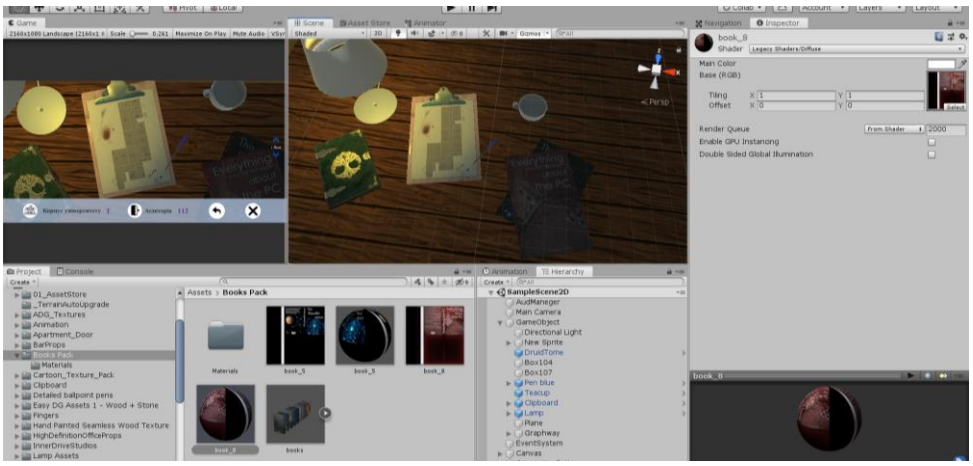

Рисунок 3.4 – пакети ассетів з об’єктами, матеріалами та спрайтами

Усі обрані моделі, для сцени з 2D мапою, та їх матеріали розміщено на спеціальній площині, що має текстуру дерева, імітуючи тим самим стіл. На тонкий прямокутник папки-планшета прикріплена текстура зі схемою правого крила першого поверху Н корпусу (рис 3.5).

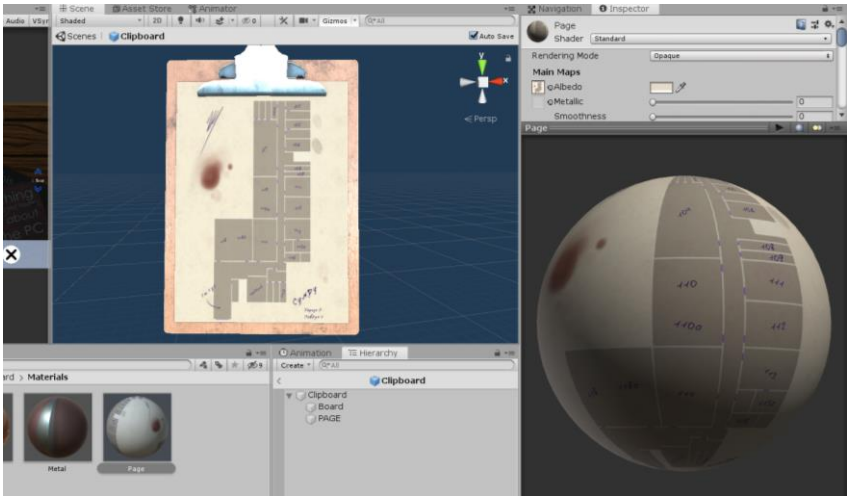

Рисунок 3.5 – зображення моделі планшету

Для сцени з 3D мапою використані інші об’єкти, які розміщені на столі зі склом. Безпосередньо під моделлю мапи був поставлений голограмний прожектор, для створення образу голограми для мапи (рис 3.6).

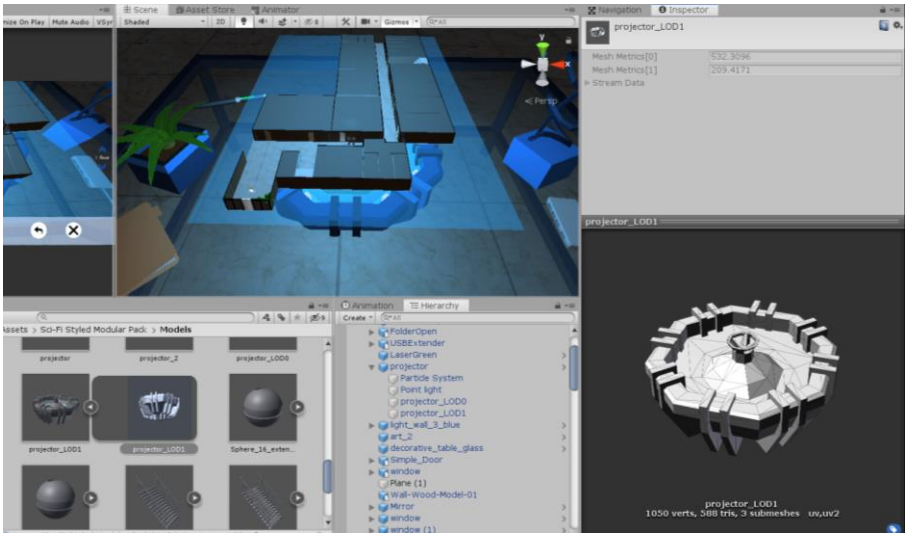

Рисунок 3.6 – зображення моделі прожектора

Особливе освітлення встановлено за допомогою об’єкта Point Light та Directional Light. Для першої сцени було додано об’єкт Directional Light з типом освітлення «з точки у певному напрямі» (spot), для другої – об’єкт Point Light з типом освітлення «точка» (point).

Для можливості комфортного користування сценами користувачу на мобільному пристрої, створені та прикріплені до камери скрипти, які зчитують кількість дотиків пальців до екрана, та викликають певні функції, такі як: переміщення камери по осях x та z та масштабування.

```
…
if (Input.touchCount > 0 && Input.GetTouch(0).phase == TouchPhase.Moved)
        {
            Vector2 touchDeltaPosition = Input.GetTouch(0).deltaPosition;
            transform.Translate(-touchDeltaPosition.x * speed, touchDeltaPosition.y *
speed, 0);
            transform.position = new Vector3(
               Mathf.Clamp(transform.position.x, 149.0f, 1353.0f),
               Mathf.Clamp(transform.position.y, -302.0f, 489.0f),
                Mathf.Clamp(transform.position.z, 306.0f, 1106.0f)
               );
    }
        if(Input.touchCount == 2)
        {
            Touch touchZero = Input.GetTouch(0);
            Touch touchOne = Input.GetTouch(1);
            Vector2 touchZeroPrevPos = touchZero.position -
touchZero.deltaPosition;
            Vector2 touchOnePrevPos = touchOne.position - touchOne.deltaPosition;
            float prevTouchDeltaMag = (touchZeroPrevPos -
touchOnePrevPos).magnitude;
            float touchDeltaMag = (touchZero.position -
touchOne.position).magnitude;
            float deltaMagnitudeDiff = prevTouchDeltaMag - touchDeltaMag;
            Camera.main.fieldOfView += deltaMagnitudeDiff * 0.1f;
            Camera.main.fieldOfView = Mathf.Clamp(Camera.main.fieldOfView,
10.0f, 50.0f);
        }
…  (див. додаток В)
```

На рисунку 3.7 показана робота додатку у платформі Unity та результат алгоритму A*.

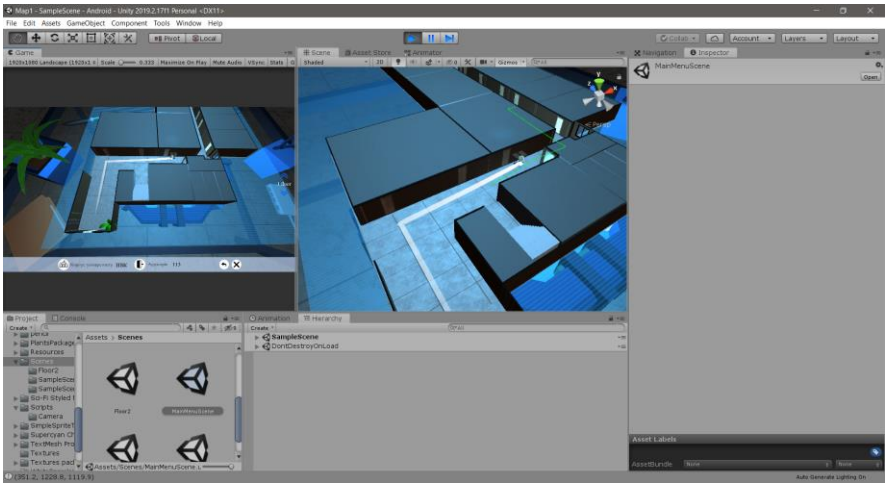

Рисунок 3.7 – запуск додатку на платформі Unity.

3.4 Реалізація вибору між режимами 2D та 3D візуалізації

У початковій бета-версії додатку ми маємо лише два поверхи, тому завантаження сцени буде виглядати таким чином:

```
void TaskOnClick()
  {
    if (Toggle2d.isOn)
    {
      if (Data.selectBuild == "НВК")
        if ((Data.selectAud == "102") || (Data.selectAud == "103") || (Data.selectAud == "104") ||
(Data.selectAud == "105") || (Data.selectAud == "106") || (Data.selectAud == "108") ||
(Data.selectAud == "109") || (Data.selectAud == "110") || (Data.selectAud == "111") ||
(Data.selectAud == "112") || (Data.selectAud == "113") || (Data.selectAud == "114") ||
(Data.selectAud == "116") || (Data.selectAud == "118") || (Data.selectAud == "209"))
        SceneManager.LoadScene(sceneName1);
     } else if (Toggle3d.isOn)
       {
         if (Data.selectBuild == "НВК")
           if ((Data.selectAud == "102") || (Data.selectAud == "103") || (Data.selectAud == "104")
|| (Data.selectAud == "105") || (Data.selectAud == "106") || (Data.selectAud == "108") ||
(Data.selectAud == "109") || (Data.selectAud == "110") || (Data.selectAud == "111") ||
(Data.selectAud == "112") || (Data.selectAud == "113") || (Data.selectAud == "114") ||
(Data.selectAud == "116") || (Data.selectAud == "118") || (Data.selectAud == "209"))
         SceneManager.LoadScene(sceneName2);
       }
       …
```

(див. додаток В)

Для правильної роботи цього скрипта, необхідно додати у наступні налаштування:

1) Відкрити необхідну сцену;
2) Перейти до вкладки File – Build settings;
3) У відкритому вікні вибрати платформу Android та натиснути кнопку Add Open Scenes;
4) Після усіх вище сказаних операцій, у полі над кнопкою з’явилася назва сцени, яку додали;

Після цього було прикріплено даний скрипт до камери. У спеціальні поля, що знаходяться у лівій частині програми, яка називається Інспектор, додані усі назви сцен: SampleScene2D з мапою у 2D форматі та SampleScene з мапою у 3D; перемикачів: Toggle2d і Toggle3d; кнопки Пошук. На рисунку 3.8 зображене вікно Інспектора з вище зазначеним скриптом.

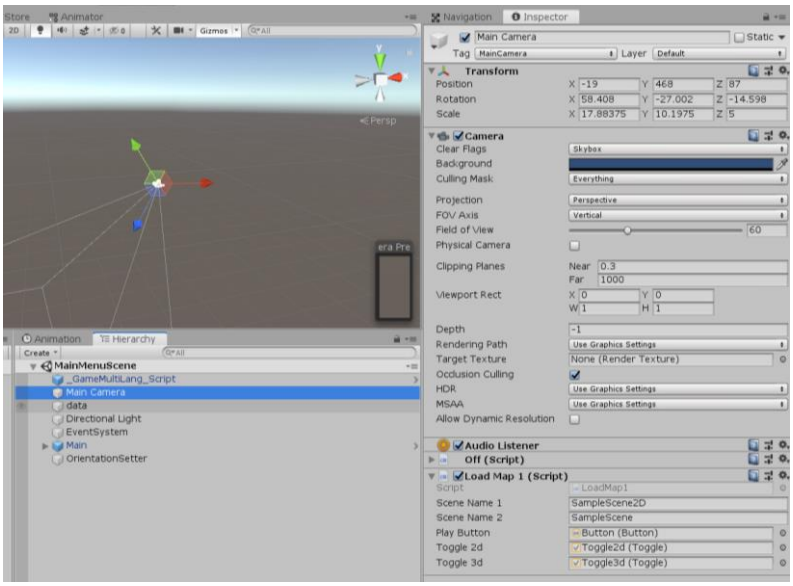

Рисунок 3.8 – вікно програми з вкладкою Inspector

3.5 Локалізація мобільного додатку

Після аналізу схожих додатків виявлено, що однією з головних причин їхньої популярності те, що вони локалізовані кількома мовами, тим самим можуть охопити більшу аудиторію. Оскільки у Сумському державному університеті

навчаються іноземні студенти, і саме вони частіше запитують шлях до аудиторій, ця функція буде дуже зручною.

Для вирішення цього завдання було використано префаб з готовим локалізатором (рис. 3.9).

Префаби (prefabs) – це набір заздалегідь встановлених об’єктів (Game Objects) і компонентів, які будуть використовуватися у програмах чи додатках, і не один раз. Має таки переваги:

1) Можна створити префаб з повним функціоналом за допомогою одного рядка коду;

2) Його легко і швидко налаштувати, протестувати і модифікувати;

3) Є можливість змінювати префаб без зміни коду, що прикріплений до нього.

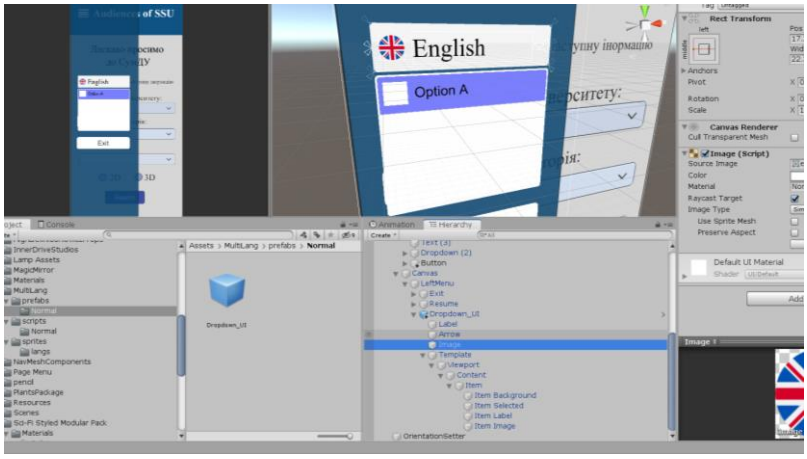

Рисунок 3.9 – префаб з локалізатором

Після визначення основних частин тексту для перекладу, на додаткове меню головного екрану додано dropbox або розкривний список, та прикріплено до його скрипта 2 текстових файли. Перший файл містить назви змінних та їх значення (value), яке містить текст додатку англійською мовою (рис 3.10). Другий файл має теж саме, але у значення записаний текст українською мовою (рис. 3.11). У майбутньому є можливість додати інші мови світу. Усі текстові файли та скрипти зазначені у додатку Б.

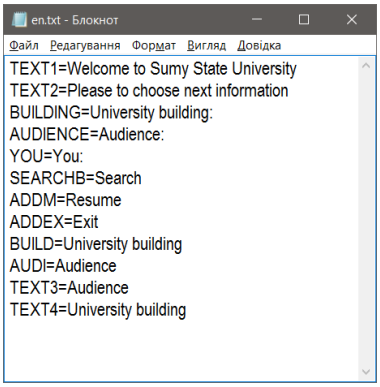

Рисунок 3.10 – файл для локалізації на англійській мові

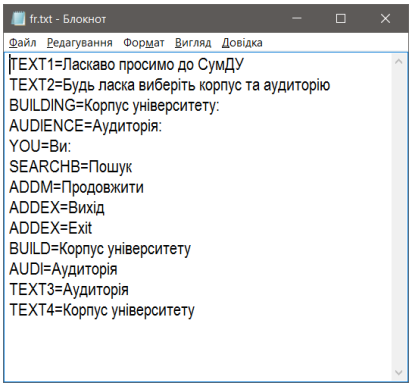

Рисунок 3.11 -файл для локалізації на українській мові

3.6 Тестування розробленого мобільного додатку

Для перевірки працездатності розробленого додатка для мобільних пристроїв, необхідно зібрати весь проект у файл з форматом .apk, який дозволяє встановлювати цілі та працездатні додатки. Він складається з повних архівних кодів Android-додатків.

Наступним кроком є завантаження і встановлення додатку на мобільний пристрій (рис. 3.12-14). Для тестування використовувалися мобільні пристрої: Samsung, з операційною системою Android 10 та роздільною здатністю 2400 x 1080; Samsung, з операційною системою Android 9 та роздільною здатністю 2340 x 1080.

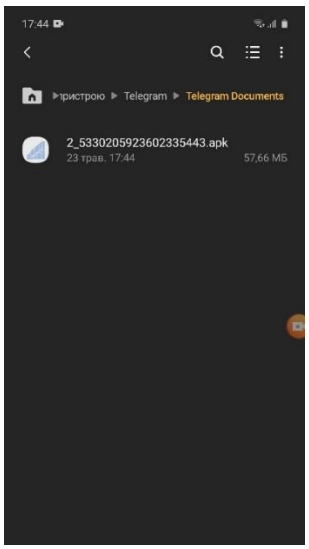

Рисунок 3.12 – apk додатка

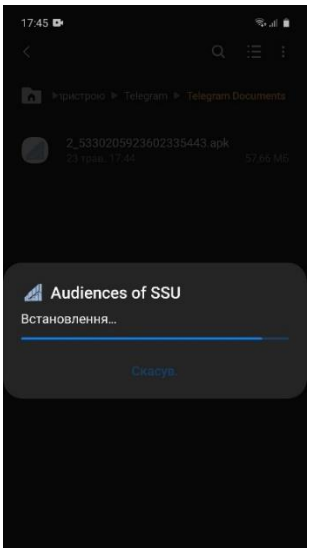

Рисунок 3.13 – встановлення додатку

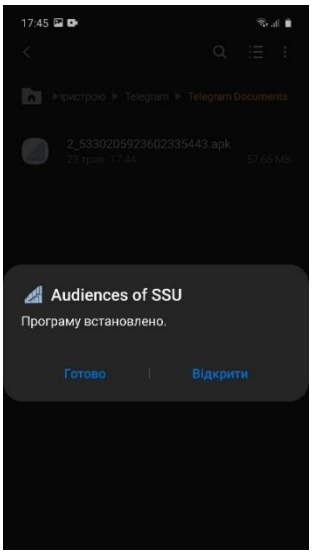

Рисунок 3.14 – меню для відкриття додатку

Після відкриття додатку спрацював стартовий екран та був здійснений перехід до головного меню (рис. 3.15-18). Після введення наступних даних: корпус – НВК, аудиторія – 105, прапорець на 2D, додаток завантажив екран с мапою (рис. 3.19-20). Алгоритм A* працює добре, navmesh agent нешвидко рухається по отриманому маршруту.

Рисунок 3.15 – стартовий екран. Перший пристрій

Рисунок 3.16 – головне меню. Перший пристрій

Рисунок 3.17 - стартовий екран. Другий пристрій

Рисунок 3.18 – головне меню. Другий пристрій

Рисунок 3.19 – знімок екрану першого пристрою з 2D мапою

Рисунок 3.20 – знімок екрану другого пристрою з 2D мапою

Після повернення на головний екран, було введено ті ж дані, але прапорець виставлений на 3D (рис. 3.21-22). Додаток з тим же часом завантажив іншу мапу. На ній алгоритм теж гарно спрацював.

Рисунок 3.21 – знімок екрану першого пристрою з 2D мапою

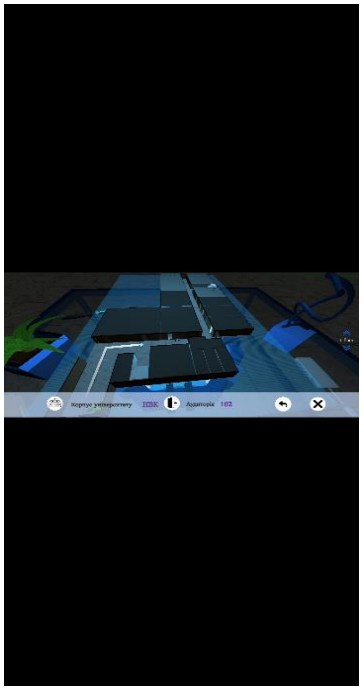

Рисунок 3.22 – знімок екрану другого пристрою з 2D мапою

ВИСНОВКИ

У випускній роботі виконано аналіз сучасного стану мобільних додатків за результатами якого визначено, що у вільному доступі відсутні додатки, які б задовольнили потреби у сфері пошуку, побудови та візуалізації опитимального маршруту пересування із місця знаходження до місця призначення за умов функціонування всередині приміщень.

В рамках даної проблеми було виконано аналітичний огляд алгоритмів пошуку найкоротшого шляху на базі методів теорії графів. Визначено, що в рамках заданої задачі оптимальним є використання алгоритму «A*», який дозволяє визначити оптимальний шлях до пункту призначення та має низьку обчислювальну складність, що робить доречним його використання в мобільних 3D додатках.

Виконано програмну реалізацію запропонованого алгоритму «A*» на базі прогамної платформи Unity3D.

Розроблено та програмно реалізовано дизайн графічного інтерфейсу користувача програмної системи, який здатен функціонувати в режимі 2D та 3D візуалізації.

Виконано програмну локалізацію інтерфейсу для користування в англомовному середовищі та перевірена працездатність розробленої системи в умовах навчального процесу в Сумському державному університеті.

СПИСОК ЛІТЕРАТУРИ

1. Антон Юдін, «Розробка додатків для Android, про що важливо знати», 2020. [Електронний ресурс]. Доступно: https://marketer.ua/ua/android-applocation-development/. Дата звернення: Травень 30, 2020.

2. «Створення (розробка) мобільних додатків». [Електронний ресурс]. Доступно: https://pbb.lviv.ua/posluhy/inshi-posluhy/stvorennia-rozrobka-mobilnykh-dodatkiv. Дата звернення: Травень 30, 2020.

3. Сергій Депутат, «Поза мережею. П’ять кращих додатків для навігації офлайн», 2018. [Електронний ресурс]. Доступно: https://nv.ua/techno/it-industry/zhizn-offlajn-5-prilozhenij-s-oflajn-kartami-2446402.html. Дата звернення: Травень 30, 2020.

4. Ian Millington, «Chapter 4: Pathfinding», Artificial intelligence for games, San Francisco, CA: Elsevier Inc., 2006, pp. 203-300.

5. Талалаєв В.О., «Лекція 8 Методології і технології розробки мобільних додатків».

6. Авраменко В.В., «Основи програмування. Алгоритмічна мова C#». [Електронний ресурс]. Доступно: https://dl.sumdu.edu.ua/textbooks/71071/index.html. Дата звернення: Травень 30, 2020.

7. Jon Manning and Paris Buttfield-Addison, Mobile Game Development with Unity, CA: O’Reilly Media, 2017

8. Joseph Hocking, Unity in action. Multiplatform game development in C# : Manning Publications, 2015.

9. Акчурин Э.А. Программирование на языке C# в Microsoft Visual Studio .Net или SharpDevelop Учебное пособие для студентов направления «Информатика и вычислительная техника» - Самара, 2010

10. А. В. Ахо, Д. Хопкрофт, Д. Д. Ульман, Структуры данных и алгоритмы: М.: Вильямс, 2003.

11. Anders and Magnus Strand-Holm Vinther, Pathfinding in Two-dimensionalWorlds, Aarhus, Denmark: Aarhus University, June 2015.

12. Р. Хэзфилд, Л. Кирби, Искусство программирования на C. Фундаментальные алгоритмы, структуры данных и примеры применений. Энциклопедия программиста: Издательство «ДиаСофт», 2001.

13. Ryan Turner, C#: 3 books in 1 – The Ultimate Beginners, Intermediate and Expert Guide to learn C# programming step by step: N.B.L Publishing, December 2019.

14. Ian Millington, AI for Games: 3 Edition, Boca Raton, Florida, USA: Taylor & Francis Group, 2019.

ДОДАТОК А

ТЕХНІЧНЕ ЗАВДАННЯ
на розробку інформаційної системи Audiences of SSU на програмній платформі Unity

1. Програмне проектування графічного інтерфейсу користувача

1.1 Призначення інформаційної системи

Інформаційна система має вигляд мобільного додатку для пошуку, побудови та 3d візуалізації оптимальних маршрутів руху Сумським державним університетом.

1.2 Мета створення інформаційної системи

Підтримка українських та іноземних студентів Сумського державного університету.

1.3 Цільова аудиторія

У цільовій аудиторії даного додатку можна виділити такі групи:

1) Студенти університету;
2) Викладачі;
3) Гості.

2. Вимоги до інформаційної системи

2.1 Вимоги до системи в цілому

Інформаційна система повинна бути реалізована у вигляді мобільного додатку.

2.2 Вимоги до користувачів

Для роботи з додатком не потрібні особливі навички користування додатками, спеціальні знання програмних продуктів, окрім інформації щодо встановлення додатків на мобільні пристрої.

2.3 Вимоги до збереження інформації

Збереження даних про аудиторії реалізоване у вигляді змінних, які зберігаються після завантаження наступної сцени та використовується іншими функціями.

2.4 Вимоги до розмежування доступу

Користування мобільним додатком є загальнодоступним.

2.5 Вимоги до структури додатка

Мобільний додаток має складатися з наступних частин:

- Початковий екран;
- Головне меню;

- Додаткове меню;
- Сцени з 2D та 3D мапами;

2.6 Вимоги до навігації

Графічний інтерфейс користувача повинен забезпечувати інтуїтивно зрозуміле представлення структури розміщеної на головній формі функціональних кнопок та розкривних списків. Навігаційні елементи повинні забезпечувати розуміння користувачем їх змісту.

2.7 Вимоги до функціональних можливостей

Додаток повинен надавати можливість вибору певної аудиторії та корпусу, проводити пошук, побудову та візуалізацію оптимальних маршрутів.

2.8 Функціональні можливості

Головне меню має наступні навігаційні елементи:

- Розкривні списки з вибором корпусу, аудиторії, ролі;
- Кнопка Пошук;
- Кнопка додаткового меню;
- Кнопка Продовжити;
- Кнопка Вихід;
- Розкривний список із вибором мови;
- Кнопка зі стрілкою для повернення до головного меню;

2.9 Вимоги до функціональних можливостей

2.9.1 Вимоги до функціональних можливостей

Реалізація програми здійснена з використанням:

- Програмна платформа Unity3d;
- Мова програмування C#;
- Microsoft Visual Studio 2017;
- Pain Tool Sai 2;

2.9.2 Вимоги до програмних можливостей

Програмне забезпечення мобільного додатку повинне задовольняти наступним вимогам:

- Операційна система Android 7 та вище;

2.9.3 Вимоги до апаратного забезпечення

- Наявність вільного місця у пам’яті мобільного пристрою розміром 60-70 Мб.

3. Склад і зміст робіт для створення мобільного додатка

Повний опис етапів роботи зі створення мобільного додатку наведено в таблиці 1.

Таблиця 1 – Етапи створення додатку

№	Опис роботи	Термін розробки
1	Аналіз існуючих програмних рішень	3 дні
2	Аналіз та вибір алгоритмів пошуку найкоротшого шляху	5 днів
3	Проектування та реалізація стартового екрану та головного меню	4 дні
4	Реалізація вибраного алгоритму у середовищі додатку	30 днів
5	Реалізація внутрішнього середовища мапи	3 дні
6	Реалізація вибору між режимами 2D та 3D візуалізації	1 день
7	Реалізація локалізації мобільного додатку	1 день
8	Завершення роботи: проведення виправлень, тестування готового додатку.	7 днів
9	Загальна тривалість робіт (з урахуванням днів на виправлення помилок) і термін закінчення проекту	71 день

ДОДАТОК Б

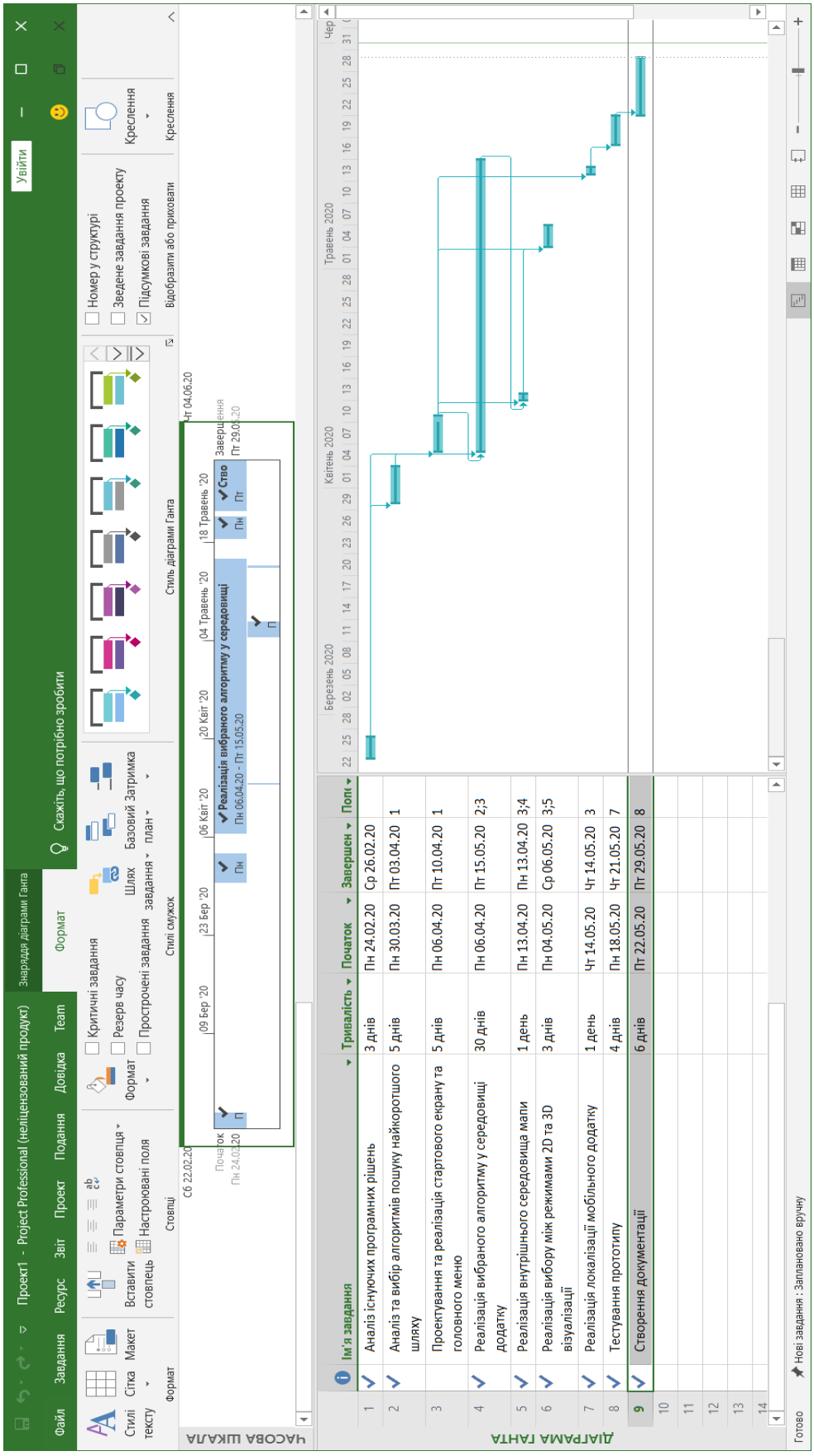

Рисунок 1 – Календарний план виконання проекту (Діаграма Ганта)

ДОДАТОК В

Програмна реалізація

LoadMap1.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class LoadMap1 : MonoBehaviour
{
    public string sceneName1;
    public string sceneName2;
    public Button playButton;
    public Toggle Toggle2d, Toggle3d;

     void Start()
    {
        playButton.onClick.AddListener(TaskOnClick);
    }

    void TaskOnClick()
    {
        if (Toggle2d.isOn)
        {
            if (Data.selectBuild == "HBK")
                if ((Data.selectAud == "102") || (Data.selectAud == "103") || (Data.selectAud == "104")
|| (Data.selectAud == "105") || (Data.selectAud == "106") || (Data.selectAud == "108") ||
(Data.selectAud == "109") || (Data.selectAud == "110") || (Data.selectAud == "111") ||
(Data.selectAud == "112") || (Data.selectAud == "113") || (Data.selectAud == "114") ||
(Data.selectAud == "116") || (Data.selectAud == "118") || (Data.selectAud == "209"))
                    SceneManager.LoadScene(sceneName1);

        }else if (Toggle3d.isOn)
        {
            if (Data.selectBuild == "HBK")
                if ((Data.selectAud == "102") || (Data.selectAud == "103") || (Data.selectAud == "104")
|| (Data.selectAud == "105") || (Data.selectAud == "106") || (Data.selectAud == "108") ||
(Data.selectAud == "109") || (Data.selectAud == "110") || (Data.selectAud == "111") ||
(Data.selectAud == "112") || (Data.selectAud == "113") || (Data.selectAud == "114") ||
(Data.selectAud == "116") || (Data.selectAud == "118") || (Data.selectAud == "209"))
                    SceneManager.LoadScene(sceneName2);
        }
    }
}
```

Data.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine.UI;
using UnityEngine;

public class Data : MonoBehaviour
{
   List<string> build = new List<string>() { "Виберіть корпус","НВК", "Ц", "БК", "М",
"Т","А","ЕТ", "Г"};
   List<string> aud = new List<string>() {"Виберіть аудиторію", "102", "103", "104", "105",
"106", "108", "109", "110", "111", "112", "113", "114", "116", "118", "119", "121", "209", "210",
"227", "314", "317"};
   List<string> role = new List<string>() { "Виберіть роль", "student", "teacher", "guest" };
   public Dropdown dropdown1, dropdown2, dropdown3;
   public static string selectAud;
   public static string selectBuild;
   public int builds;
   public int auds;

   public void Dropdown1_IndexChanged(int index)
   {
      selectBuild = build[index];
   }
   public void Dropdown2_IndexChanged(int index)
   {
      selectAud = aud[index];
   }
       void Start()
   {
      DontDestroyOnLoad(this.gameObject);
      PopulateList();
   }

   void PopulateList()
   {
      dropdown1.AddOptions(build);
      dropdown2.AddOptions(aud);
      dropdown3.AddOptions(role);
   }
   void Awake()
   {
      DontDestroyOnLoad(this.gameObject);
   }
}
```

ChoosedText.cs

```
using System.Collections;
```

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class ChoosedText : MonoBehaviour
{
   public Text Audtext;
   public Text Buildtext;

   void Update()
   {
      if(Data.selectBuild !=null)
         if(Data.selectAud !=null)
          Choosed();
   }

   void Choosed()
   {
      Buildtext.text = Data.selectBuild;
      Audtext.text = Data.selectAud;
   }
}
```

Loaduodownback.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class Loadupdownback : MonoBehaviour
{
   public string sceneName1;
   public string sceneName2;
   public string sceneName3;
   public Button back, up, down;

   void Start()
   {
      back.onClick.AddListener(TaskOnClickback);
      up.onClick.AddListener(TaskOnClickup);
      down.onClick.AddListener(TaskOnClickdown);
   }

   void TaskOnClickback()
   {
       SceneManager.LoadScene(sceneName1);
   }
   void TaskOnClickup()
```

```
    {
        SceneManager.LoadScene(sceneName2);
    }
    void TaskOnClickdown()
    {
        SceneManager.LoadScene(sceneName3);
    }
}
```

OrientationSetter.cs

```
using UnityEngine;

public class OrientationSetter : MonoBehaviour
{
    public enum Orientation
    {
        Any,
        Portrait,
        PortraitFixed,
        Landscape,
        LandscapeFixed
    }

    public Orientation ScreenOrientation;
    private void Start()
    {
        switch (ScreenOrientation)
        {
            case Orientation.Any:
                Screen.orientation = UnityEngine.ScreenOrientation.AutoRotation;
                Screen.autorotateToPortrait = Screen.autorotateToPortraitUpsideDown = true;
                Screen.autorotateToLandscapeLeft = Screen.autorotateToLandscapeRight = true;
                break;

            case Orientation.Portrait:
                Screen.orientation = UnityEngine.ScreenOrientation.Portrait;
                Screen.orientation = UnityEngine.ScreenOrientation.AutoRotation;
                Screen.autorotateToPortrait = Screen.autorotateToPortraitUpsideDown = true;
                Screen.autorotateToLandscapeLeft = Screen.autorotateToLandscapeRight = false;
                break;

            case Orientation.PortraitFixed:
                Screen.orientation = UnityEngine.ScreenOrientation.Portrait;
                break;

            case Orientation.Landscape:
                Screen.orientation = UnityEngine.ScreenOrientation.Landscape;
                Screen.orientation = UnityEngine.ScreenOrientation.AutoRotation;
                Screen.autorotateToPortrait = Screen.autorotateToPortraitUpsideDown = false;
                Screen.autorotateToLandscapeLeft = Screen.autorotateToLandscapeRight = true;
                break;
```

```
            case Orientation.LandscapeFixed:
                Screen.orientation = UnityEngine.ScreenOrientation.LandscapeLeft;
                break;
        }
      Destroy(gameObject);
    }
}
```

GraphwayTest.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GraphwayTest : MonoBehaviour
{
    public const int MAX_SPEED = 50;
    public const int ACCELERATION = 5;

    [Tooltip("Enable Debug Mode to see algoritm in action slowed down. MAKE SURE GIZMOS
ARE ENABLED!")]
    public bool debugMode = false;
    private GwWaypoint[] waypoints;
    private float speed = 0;
    private bool audienceWay = true;

void Update()
    {
       /* if (Input.GetMouseButtonDown(0))
        {
             RaycastHit hit;
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            if (Physics.Raycast(ray, out hit))
            {
                        Debug.Log(hit.point);
                        Graphway.FindPath(transform.position, hit.point, FindPathCallback, true,
debugMode);
            }
        }*/
     if (audienceWay)
        {
            var currentAudNum = Data.selectAud;
            if (currentAudNum != string.Empty)
            {
                var listAud = AudMeneger.instance.audData;
                foreach (var currentAudData in listAud)
                {
                    if(currentAudData.numAud == currentAudNum)
                    {
                        Debug.Log(currentAudData.positionAud);
```

```
                Graphway.FindPath(transform.position, currentAudData.positionAud,
FindPathCallback, true, debugMode);
                break;
              }
            }
          }
          audienceWay = false;
      }
if (waypoints != null && waypoints.Length > 0)
{
          speed = Mathf.Lerp(speed, MAX_SPEED, Time.deltaTime * ACCELERATION);
          speed = Mathf.Clamp(speed, 0, MAX_SPEED);
          transform.LookAt(waypoints[0].position);
          transform.position = Vector3.MoveTowards(transform.position, waypoints[0].position,
Time.deltaTime * waypoints[0].speed * speed);
          if (Vector3.Distance(transform.position, waypoints[0].position) < 0.1f)
    {
              NextWaypoint();
    }
}
else
{
speed = 0;
}
if (waypoints != null && waypoints.Length > 0)
{
Vector3 startingPoint = transform.position;
for (int i = 0 ; i < waypoints.Length ; i++)
              {
                    Debug.DrawLine(startingPoint, waypoints[i].position, Color.green, 2f, false);
                    startingPoint = waypoints[i].position;
              }
}

}
private void NextWaypoint()
    {
    if (waypoints.Length > 1)
    {
              GwWaypoint[] newWaypoints = new GwWaypoint[waypoints.Length - 1];
              for (int i = 1 ; i < waypoints.Length ; i++)
              {
                    newWaypoints[i-1] = waypoints[i];
              }
              waypoints = newWaypoints;
    }
    else
    {
              waypoints = null;
    }
}
```

```
    private void FindPathCallback(GwWaypoint[] path)
    {
      if (path == null)
            {
                  Debug.Log("Path to target position not found!");
            }
            else
            {
                  waypoints = path;
            }
    }
  }
```

AudData.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;

[Serializable]
public struct AudData
{
   public string numAud;
   public Vector3 positionAud;
}
```

AudMeneger.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class AudMeneger : MonoBehaviour
{
   public List<AudData> audData = new List<AudData>();
   public static AudMeneger instance = null;
   void Start()
   {
      if (instance == null)
      {
         instance = this;
      }
   }
}
```

Graphway.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Graphway : MonoBehaviour
```

```
{
    public static Graphway instance;
    [Tooltip("Color of NODES in editor.")]
    public Color nodeColor = new Color32(236, 13, 69, 255);
    [Tooltip("Color of SUBNODES in editor.")]
    public Color subnodeColor = new Color32(40, 12, 54, 255);
    [Tooltip("Size of NODES in editor.")]
    public float nodeSize = 10;
    [Tooltip("Size of SUBNODES in editor.")]
    public float subnodeSize = 5;
    [Tooltip("Size of unidirectional connection arrows in editor.")]
    public float arrowSize = 5;
   [Tooltip("Number of nodes that can be checked in a single frame at runtime.  A higher limit will find paths faster, but may impact performance.")]
   public int pathfindFrameLimit = 100;
   public int editorNodeCounter = 0; // Keep public so value is saved upon editor close
   public Dictionary<int, GwNode> nodes;
    private List<GwJob> jobs;
    private GwJob activeJob;
    private List<int> openNodes;
   private List<int> closedNodes;
   private int currentNodeID;

   void Awake()
   {
      instance = this;
   }

   void Start()
   {
      jobs = new List<GwJob>();
      if (!transform.Find("Nodes"))
      {
         Debug.LogError("Missing Graphway child object 'Nodes'.");
      }
      else if (!transform.Find("Connections"))
      {
         Debug.LogError("Missing Graphway child object 'Connections'.");
      }
      else
      {
         Transform nodesParent = transform.Find("Nodes").transform;
         Transform connectionsParent = transform.Find("Connections").transform;
         nodes = new Dictionary<int, GwNode>();
         foreach (Transform node in nodesParent.transform)
         {
            GraphwayNode nodeData = node.GetComponent<GraphwayNode>();
            nodes[nodeData.nodeID] = new GwNode(nodeData.nodeID, node.position);
         }
         foreach (Transform connection in connectionsParent.transform)
         {
```

```
            GraphwayConnection connectionData =
connection.GetComponent<GraphwayConnection>();
            Vector3[] subnodesAB = null;
            Vector3[] subnodesBA = null;
            if (connection.childCount > 0)
            {
               subnodesAB = new Vector3[connection.childCount];
               int i = 0;
               foreach (Transform subnode in connection.transform)
               {
                  subnodesAB[i] = subnode.position;
                  i++;
               }
               subnodesBA = (Vector3[])subnodesAB.Clone();
               Array.Reverse(subnodesBA);
            }
            if (connectionData.connectionType == GraphwayConnectionTypes.Bidirectional ||
connectionData.connectionType == GraphwayConnectionTypes.UnidirectionalAToB)
            {
               nodes[connectionData.nodeIDA].AddConnection(connectionData.nodeIDB,
connectionData.disabled, connectionData.speedWeight, subnodesAB);
            }
            if (connectionData.connectionType == GraphwayConnectionTypes.Bidirectional ||
connectionData.connectionType == GraphwayConnectionTypes.UnidirectionalBToA)
            {
               nodes[connectionData.nodeIDB].AddConnection(connectionData.nodeIDA,
connectionData.disabled, connectionData.speedWeight, subnodesBA);
            }
         }
         Destroy(nodesParent.gameObject);
         Destroy(connectionsParent.gameObject);
      }
   }

   void Update()
   {
      if (jobs.Count > 0 && activeJob == null)
      {
         StartCoroutine(ProcessJob(jobs[0]));
      }
   }
public static void FindPath(Vector3 origin, Vector3 targetPosition, Action<GwWaypoint[]>
callback, bool clampToEndNode = true, bool debugMode = false)
   {
            instance.PathFind(origin, targetPosition, callback, clampToEndNode, debugMode);
   }
   public void PathFind(Vector3 origin, Vector3 targetPosition, Action<GwWaypoint[]> callback,
bool clampToEndNode = true, bool debugMode = false)
   {
      jobs.Add(new GwJob(origin, targetPosition, callback, clampToEndNode, debugMode));
   }
```

```
    private IEnumerator ProcessJob(GwJob job)
    {
        activeJob = job;
        int startNodeID = ClosestNodeID(activeJob.origin);
        int targetNodeID = ClosestNodeID(activeJob.targetPosition);
        if (startNodeID == -1 || targetNodeID == -1)
        {
            activeJob.callback(null);
        }
        else
        {
            foreach (KeyValuePair<int, GwNode> node in nodes)
            {
                node.Value.ResetPathfindingVars();
            }
          bool pathFound = false;
            openNodes = new List<int>();
            openNodes.Add(startNodeID);
            closedNodes = new List<int>();
            currentNodeID = -1;
        int nodeCounter = 1;
            while (openNodes.Count > 0 && pathFound == false)
            {
                float lowestCost = -1;
                foreach (int openNode in openNodes)
                {
                    if (lowestCost == -1 || nodes[openNode].FCost() < lowestCost)
                    {
                        currentNodeID = openNode;
                        lowestCost = nodes[openNode].FCost();
                    }
                }
                openNodes.Remove(currentNodeID);
                closedNodes.Add(currentNodeID);
                if (currentNodeID == targetNodeID)
                {
                    pathFound = true;
                }
                else
                {
                    foreach (KeyValuePair<int, GwConnection> connection in
nodes[currentNodeID].connections)
                    {
                        int connectedNodeID = connection.Value.connectedNodeID;
                        if (connection.Value.disabled == false &&
closedNodes.Contains(connectedNodeID) == false)
                        {
                            float gCost = nodes[currentNodeID].gCost +
(CalculatePathLength(currentNodeID, connectedNodeID) / connection.Value.speedWeight);
                            float hCost = Vector3.Distance(nodes[connectedNodeID].position,
nodes[targetNodeID].position);
                            float fCost = gCost + hCost;
```

```
                    if (openNodes.Contains(connectedNodeID) && fCost < nodes[connectedNodeID].FCost())
                    {
                        nodes[connectedNodeID].gCost = gCost;
                        nodes[connectedNodeID].hCost = hCost;
                        nodes[connectedNodeID].parentNodeID = currentNodeID;
                    }
                    else if(!openNodes.Contains(connectedNodeID))
                    {
                        nodes[connectedNodeID].gCost = gCost;
                        nodes[connectedNodeID].hCost = hCost;
                        nodes[connectedNodeID].parentNodeID = currentNodeID;
                        openNodes.Add(connectedNodeID);
                    }
                }
            }
        }
        if (activeJob.debugMode)
        {
            yield return new WaitForSeconds(0.25f);
        }
        else if (nodeCounter % pathfindFrameLimit == 0)
        {
            yield return null;
        }

        nodeCounter++;
    }
    if (activeJob.debugMode)
    {
        yield return new WaitForSeconds(1);
    }
    if (pathFound)
    {
        GwWaypoint[] path = TracePath(activeJob.targetPosition, currentNodeID);

        activeJob.callback(path);
    }
    else
    {
        activeJob.callback(null);
    }
  }
  jobs.RemoveAt(0);
  activeJob = null;
}
private int ClosestNodeID(Vector3 position)
{
  int closestNodeID = -1;
  float closestNodeDistance = 0;
  foreach (KeyValuePair<int, GwNode> node in nodes)
  {
```

```
        float distance = Vector3.Distance(position, node.Value.position);
        if (closestNodeID == -1 || distance < closestNodeDistance)
        {
            closestNodeID = node.Value.nodeID;
            closestNodeDistance = distance;
        }
    }
    return closestNodeID;
}
private float CalculatePathLength(int nodeIDFrom, int nodeIDTo)
{
    float pathLength = 0;
    GwConnection connection = nodes[nodeIDFrom].connections[nodeIDTo];
    Vector3 currentPosition = nodes[nodeIDFrom].position;
    if (connection.subnodes != null && connection.subnodes.Length > 0)
    {
        for (int i = 0 ; i < connection.subnodes.Length ; i++)
        {
            pathLength += Vector3.Distance(currentPosition, connection.subnodes[i]);
            currentPosition = connection.subnodes[i];
        }
    }
    pathLength += Vector3.Distance(currentPosition, nodes[nodeIDTo].position);

    return pathLength;
}
private GwWaypoint[] TracePath(Vector3 targetPosition, int currentNodeID)
{
    List<GwWaypoint> path = new List<GwWaypoint>();
            if (activeJob.clampToEndNode == false)
            {
                    path.Add(new GwWaypoint(targetPosition));
            }

    while (currentNodeID != -1)
    {
        int parentNodeID = nodes[currentNodeID].parentNodeID;
        if (parentNodeID != -1)
        {
            GwConnection connection = nodes[parentNodeID].connections[currentNodeID];
            path.Add(new GwWaypoint(nodes[currentNodeID].position, connection.speedWeight));
             if (connection.subnodes != null && connection.subnodes.Length > 0)
             {
                for (int i = connection.subnodes.Length - 1 ; i >= 0 ; i--)
                {
                    path.Add(new GwWaypoint(connection.subnodes[i], connection.speedWeight));
                }
             }
        }
        else
        {
            path.Add(new GwWaypoint( nodes[currentNodeID].position));
```

```
        }
        currentNodeID = parentNodeID;
      }
      path.Reverse();
      return path.ToArray();
   }
   void OnDrawGizmosSelected()
   {
      if (Application.isPlaying)
      {
         foreach (KeyValuePair<int, GwNode> node in nodes)
         {
            Gizmos.color = nodeColor;
            Gizmos.DrawSphere(RaiseHeight(node.Value.position), nodeSize / 2);

         foreach (KeyValuePair<int, GwConnection> connection in node.Value.connections)
            {
               if ( ! connection.Value.disabled)
               {

                        Vector3 nodeAPosition = node.Value.position;
                  Vector3 nodeBPosition = nodes[connection.Value.connectedNodeID].position;
                  Vector3 lastPosition = nodeAPosition;
                   if (connection.Value.subnodes != null && connection.Value.subnodes.Length >
0)
                        {
                           for (int i = 0 ; i < connection.Value.subnodes.Length ; i++)
                           {
                              Vector3 subnodePosition = connection.Value.subnodes[i];
                              Gizmos.color = subnodeColor;
                              Gizmos.DrawSphere(RaiseHeight(subnodePosition),  subnodeSize  /
2);
                              Gizmos.color = (i == 0 ? nodeColor : subnodeColor);
                              Gizmos.DrawLine(RaiseHeight(lastPosition),
RaiseHeight(subnodePosition));
                              if                                    (                                    !
nodes[connection.Value.connectedNodeID].connections.ContainsKey(node.Value.nodeID))
                              {
                                 DrawArrow(RaiseHeight(lastPosition),
RaiseHeight(subnodePosition - lastPosition) / 2, (i == 0 ? nodeColor : subnodeColor));
                              }
                     lastPosition = subnodePosition;
                           }
                        }
                              Gizmos.color = nodeColor;
                        Gizmos.DrawLine(RaiseHeight(lastPosition),
RaiseHeight(nodeBPosition));
                  if                                    (                                    !
nodes[connection.Value.connectedNodeID].connections.ContainsKey(node.Value.nodeID))
                  {
                     DrawArrow(RaiseHeight(lastPosition),          RaiseHeight(nodeBPosition          -
lastPosition) / 2, nodeColor);
```

```
            }
            }
            }
        }

                if (activeJob != null && activeJob.debugMode)
        {
            int parentID;
            for (int i = 0 ; i < openNodes.Count ; i++)
            {
                int nodeID = openNodes[i];
                Gizmos.color = Color.white;
                Gizmos.DrawSphere(nodes[nodeID].position, nodeSize);
                parentID = nodes[nodeID].parentNodeID;
                if (parentID != -1)
                {
            Gizmos.DrawLine(RaiseHeight(nodes[nodeID].position),
RaiseHeight(nodes[parentID].position));
                }
            }
            for (int i = 0 ; i < closedNodes.Count ; i++)
            {
                int nodeID = closedNodes[i];
                Gizmos.color = Color.gray;
                Gizmos.DrawSphere(nodes[nodeID].position, nodeSize);
                parentID = nodes[nodeID].parentNodeID;

          if (parentID != -1)
                {
            Gizmos.DrawLine(RaiseHeight(nodes[nodeID].position),
RaiseHeight(nodes[parentID].position));
                }
            }
            Gizmos.color = Color.green;
           Gizmos.DrawSphere(nodes[currentNodeID].position, nodeSize);
           parentID = nodes[currentNodeID].parentNodeID;
           if (parentID != -1)
           {
            Gizmos.DrawLine(RaiseHeight(nodes[currentNodeID].position),
RaiseHeight(nodes[parentID].position));
           }
      }
   }
   }
   private void DrawArrow(Vector3 position, Vector3 direction, Color color)
    {
            Vector3 right = Quaternion.LookRotation(direction) * Quaternion.Euler(0,180+30,0) *
new Vector3(0,0,1);
            Vector3 left = Quaternion.LookRotation(direction) * Quaternion.Euler(0,180-30,0) *
new Vector3(0,0,1);

            Gizmos.DrawRay(position + direction, right * (nodeSize / 2));
```

```
            Gizmos.DrawRay(position + direction, left * (nodeSize / 2));
    }
   private Vector3 RaiseHeight(Vector3 position)
    {
            position.y += 0.01f;
            return position;
    }
}
```

GraphwayConnection.cs

```
using UnityEngine;
public enum GraphwayConnectionTypes {
    Bidirectional,
   UnidirectionalAToB,
   UnidirectionalBToA
};

public class GraphwayConnection : MonoBehaviour
{
    public int nodeIDA = -1;
public int nodeIDB = -1;

    [Tooltip("Toggle connection enabled/disabled.  Disabled connections will not be used for pathfinding until enabled.")]
    public bool disabled = false;

   [Tooltip("How fast this path can be travelled relative to other paths.  This will be taken into account when searching for the fastest path.")]
   public float speedWeight = 1;

   [Tooltip("The type of connection.  Bidirectional = two way.  Unidirectional = one way only.")]
    public GraphwayConnectionTypes connectionType = GraphwayConnectionTypes.Bidirectional;

   public int editorSubnodeCounter = 0; // Keep public so value is saved upon editor close

   public void SetConnectionData(int nodeIDA, int nodeIDB, GraphwayConnectionTypes connectionType)
   {
      this.nodeIDA = nodeIDA;
      this.nodeIDB = nodeIDB;
      this.connectionType = connectionType;
   }
}
```

GraphwayNode.cs

```
using UnityEngine;

public class GraphwayNode : MonoBehaviour
{
```

```
    public int nodeID = -1;
    public void SetNodeID(int nodeID)
    {
            this.nodeID = nodeID;
    }
}
```

GwConnection.cs

```
using UnityEngine;


public class GwConnection
{
   public int connectedNodeID;
   public bool disabled;
   public float speedWeight;
   public Vector3[] subnodes;

   public GwConnection(int connectedNodeID, bool disabled, float speedWeight, Vector3[]
subnodes)
   {
      this.connectedNodeID = connectedNodeID;
      this.disabled = disabled;
      this.speedWeight = speedWeight;
      this.subnodes = subnodes;
   }
}
```

GwJob.cs

```
using System;
using UnityEngine;

public class GwJob
{
   public Vector3 origin;
   public Vector3 targetPosition;
   public Action<GwWaypoint[]> callback;
   public bool clampToEndNode;
   public bool debugMode;

   public GwJob(Vector3 origin, Vector3 targetPosition, Action<GwWaypoint[]> callback, bool
clampToEndNode, bool debugMode)
   {
      this.origin = origin;
      this.targetPosition = targetPosition;
      this.callback = callback;
      this.clampToEndNode = clampToEndNode;
      this.debugMode = debugMode;
   }
}
```

GwNode.cs

```
using System.Collections.Generic;
using UnityEngine;

public class GwNode
{
   public int nodeID;
   public Vector3 position;
   public Dictionary<int, GwConnection> connections;
   public float gCost;
   public float hCost;
   public int parentNodeID;

   public GwNode(int nodeID, Vector3 position)
   {
      this.nodeID = nodeID;
      this.position = position;

      connections = new Dictionary<int, GwConnection>();
   }
   public void AddConnection(int connectedNodeID, bool disabled, float speedWeight, Vector3[]
subnodes)
   {
      if (!connections.ContainsKey(connectedNodeID))
      {
         connections[connectedNodeID] = new GwConnection(connectedNodeID, disabled,
speedWeight, subnodes);
      }
      else
      {
         UpdateConnection(connectedNodeID, disabled, speedWeight, subnodes);
      }
   }
   public void UpdateConnection(int connectedNodeID, bool isDisabled, float speedWeight,
Vector3[] subnodes)
   {
      if (connections.ContainsKey(connectedNodeID))
      {
         connections[connectedNodeID].disabled = isDisabled;
         connections[connectedNodeID].speedWeight = speedWeight;
         connections[connectedNodeID].subnodes = subnodes;
      }
   }
   public void DisableConnection(int connectedNodeID)
   {
      if (connections.ContainsKey(connectedNodeID))
      {
         connections[connectedNodeID].disabled = true;
      }
   }
   public void EnableConnection (int connectedNodeID)
```

```
    {
        if (connections.ContainsKey(connectedNodeID))
        {
            connections[connectedNodeID].disabled = false;
        }
    }
    public void SetConnectionSpeedWeight(int nodeToID, float speedWeight)
    {
        if (connections.ContainsKey(nodeToID))
        {
            connections[nodeToID].speedWeight = speedWeight;
        }
    }

    public void RemoveConnection(int connectedNodeID)
    {
        if (connections.ContainsKey(connectedNodeID))
        {
            connections.Remove(connectedNodeID);
        }
    }
    public void ResetPathfindingVars()
    {
        gCost = 0;
        hCost = 0;
        parentNodeID = -1;
    }
    public float FCost()
    {
        return gCost + hCost;
    }
}
```

GwWaypoint.cs

```
using UnityEngine;

public class GwWaypoint
{
    public Vector3 position;
    public float speed;

    public GwWaypoint(Vector3 position, float speed = 1)
    {
        this.position = position;
        this.speed = speed;
    }
}
```

AddLeftMenu.cs

```
using System.Collections;
```

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class AddLeftMenu : MonoBehaviour
{
    public GameObject PauseMenu;
    public GameObject MenuNavigation;
    public bool IfPause=false;

    void Update()
    {
        if (IfPause)
        {
            Pause();
        }
        else if (!IfPause)
        {
            Resume();
        }
    }

    public void Pause()
    {
        PauseMenu.SetActive(false);
        MenuNavigation.SetActive(true);
        IfPause = true;
        Time.timeScale = 0f;
    }

    public void Resume()
    {
        PauseMenu.SetActive(true);
        MenuNavigation.SetActive(false);
        IfPause = false;
        Time.timeScale = 1f;
    }
    public void Exit()
    {
        Application.Quit();
    }
}
```

LangDropDown.cs

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class LangDropDown : MonoBehaviour
{
    [SerializeField] string[] myLangs;
```

```
    Dropdown drp;
    int index;

    void Awake ()
    {
            drp = this.GetComponent <Dropdown> ();
            int v = PlayerPrefs.GetInt ("_language_index", 0);
            drp.value = v;

            drp.onValueChanged.AddListener (delegate {
                    index = drp.value;
                    PlayerPrefs.SetInt ("_language_index", index);
                    PlayerPrefs.SetString ("_language", myLangs [index]);
                    Debug.Log ("language changed to " + myLangs [index]);
                    //apply changes
                    ApplyLanguageChanges ();
            });
    }

    void ApplyLanguageChanges ()
    {
            SceneManager.LoadScene (1);
    }

    void OnDestroy ()
    {
            drp.onValueChanged.RemoveAllListeners ();
    }
}
```

TextTranslator.cs

```
using UnityEngine;
using UnityEngine.UI;
[RequireComponent (typeof(Text))]
public class TextTranslator : MonoBehaviour
{
    [Tooltip ("enter one of the keys that you specify in your (txt) file for all languages.\n\n# for
example: [HOME=home]\n# the key here is [HOME]")]
    [Header ("Enter your word key here.")]
    [SerializeField]
    string key;

    void Start ()
    {
            GetComponent <Text> ().text = GameMultiLang.GetTraduction (key);
    }
}
```

en.txt

```
TEXT1=Welcome to Sumy State University
TEXT2=Please to choose next information
```

```
BUILDING=University building:
AUDIENCE=Audience:
YOU=You:
SEARCHB=Search
ADDM=Resume
ADDEX=Exit
BUILD=University building
AUDI=Audience
TEXT3=Audience
TEXT4=University building
```

fr.txt

```
TEXT1=Ласкаво просимо до СумДУ
TEXT2=Будь ласка виберіть корпус та аудиторію
BUILDING=Корпус університету:
AUDIENCE=Аудиторія:
YOU=Ви:
SEARCHB=Пошук
ADDM=Продовжити
ADDEX=Вихід
ADDEX=Exit
BUILD=Корпус університету
AUDI=Аудиторія
TEXT3=Аудиторія
TEXT4=Корпус університету
```

AndroidCameraController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class AndroidCameraController : MonoBehaviour
{
   public float speed = 0.3f;
   void Update()
   {
      if (Input.touchCount > 0 && Input.GetTouch(0).phase == TouchPhase.Moved){
         Vector3 touchDeltaPos = Input.GetTouch(0).deltaPosition;
         transform.Translate (-touchDeltaPos.x * speed,-touchDeltaPos.y * speed  ,-
touchDeltaPos.z * speed);
         transform.position = new Vector3(
               Mathf.Clamp(transform.position.x, -794.0f, 1131.0f),
               Mathf.Clamp(transform.position.y, 1314.0f, 1804.0f),
               Mathf.Clamp(transform.position.z, 118.0f, 1973.0f)
            );

      }
      if (Input.touchCount == 2){
         Touch touchZero = Input.GetTouch(0);
         Touch touchOne = Input.GetTouch(1);
```

```
            Vector2 touchZeroPrevPos = touchZero.position - touchZero.deltaPosition;
            Vector2 touchOnePrevPos = touchOne.position - touchOne.deltaPosition;

            float prevTouchDeltaMag = (touchZeroPrevPos - touchOnePrevPos ).magnitude;
            float touchDeltaMag  = (touchZero.position - touchOne.position).magnitude;

            float deltaMagnitudeDiff = prevTouchDeltaMag - touchDeltaMag;

            Camera.main.fieldOfView += deltaMagnitudeDiff * 0.1f;

            Camera.main.fieldOfView = Mathf.Clamp( Camera.main.fieldOfView, 25.0f,100.0f);

        }

    }
}
```

CameraControllerRTS.txt

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraControllerRTS : MonoBehaviour
{
    public float speed;

    void Start()
    {
#if UNITY_ANDROID
        QualitySettings.vSyncCount = 0;
        Application.targetFrameRate = 60;
        QualitySettings.antiAliasing = 0;
        Screen.sleepTimeout = SleepTimeout.NeverSleep;
#endif
    }

    void Update()
    {
        if (Input.touchCount > 0 && Input.GetTouch(0).phase == TouchPhase.Moved)
        {
            Vector2 touchDeltaPosition = Input.GetTouch(0).deltaPosition;
            transform.Translate(-touchDeltaPosition.x * speed, touchDeltaPosition.y * speed, 0);

            transform.position = new Vector3(
                Mathf.Clamp(transform.position.x, 149.0f, 1353.0f),
                Mathf.Clamp(transform.position.y, -302.0f, 489.0f),
                 Mathf.Clamp(transform.position.z, 306.0f, 1106.0f)
                );
        }
        if(Input.touchCount == 2)
        {
```

```
        Touch touchZero = Input.GetTouch(0);
        Touch touchOne = Input.GetTouch(1);

        Vector2 touchZeroPrevPos = touchZero.position - touchZero.deltaPosition;
        Vector2 touchOnePrevPos = touchOne.position - touchOne.deltaPosition;

        float prevTouchDeltaMag = (touchZeroPrevPos - touchOnePrevPos).magnitude;
        float touchDeltaMag = (touchZero.position - touchOne.position).magnitude;

        float deltaMagnitudeDiff = prevTouchDeltaMag - touchDeltaMag;

        Camera.main.fieldOfView += deltaMagnitudeDiff * 0.1f;
        Camera.main.fieldOfView = Mathf.Clamp(Camera.main.fieldOfView, 10.0f, 50.0f);
    }

    }
}
```