МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. В. ДАЛЯ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ

КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ПРОГРАМУВАННЯ

До захисту допускається:
завідувач кафедри ІТП

Лифар В.О.

«____» 2023 р.

ДИПЛОМНИЙ ПРОЕКТ (РОБОТА) БАКАЛАВРА

ПОЯСНЮВАЛЬНА ЗАПИСКА

НА ТЕМУ:

Містобудівна гра

Освітній ступінь – «бакалавр»

Спеціальність 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

Керівник проекту: _____ В.О. Лифар
(підпис) (ініціали, прізвище)

Здобувач вищої освіти _____ І.В. Корабльов
(підпис) (ініціали, прізвище)

Київ 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет Інформаційних технологій та електроніки

Кафедра Інформаційних технологій та програмування

Освітній ступінь бакалавр

Спеціальність 121 Інженерія програмного забезпечення

(шифр і назва)

ЗАТВЕРДЖУЮ:

завідувач кафедри ІТП

Лифар В.О.

«____» 2023 р.

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) БАКАЛАВРА

Корабльов Ілля Віталійович

(прізвище, ім'я, по батькові)

1. Тема роботи: Містобудівна гра

Керівник проекту (роботи) Лифар Володимир Олексійович, д.т.н., доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від 26.04.2023 № 240/15.15-ОД

3. Вихідні дані до роботи матеріали переддипломної практики.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): а) аналіз предметної області;
б) вибір та обґрунтування програмних засобів розробки гри;
в) розробка ігрового додатку;
г) висновки.

5. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

6. Дата видачі завдання 24.03.2023

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Одержання завдання на виконання роботи	24.03.2023-25.03.2023	
2	Укладання і погодження з керівником плану і етапів виконання роботи	27.03.2023-28.03.2023	
3	Узагальнення даних літературних джерел, укладання розділу «Аналіз предметної галузі»	10.04.2023-11.04.2023	
4	Аналіз шляхів виконання завдання. Вибір і погодження з керівником оптимального шляху	19.04.2023-25.04.2023	
5	Укладання та тестування програмного продукту	27.04.2023-05.05.2023	
6	Укладання, оформлення та погодження пояснювальної записки з керівником	28.05.2023-30.05.2023	
7	Здача готової пояснювальної записки на кафедру	28.05.2023-30.05.2023	
8	Укладання доповіді і презентації	31.05.2023-05.06.2023	

Здобувач вищої освіти _____ І.В, Корабльов

(підпис) (ініціали, прізвище)

Керівник _____ Лифар В.О.

(підпис) (ініціали, прізвище)

РЕФЕРАТ

Робота містить: 41 сторінок основного тексту, 43 сторінок додатків, 34 рисунків, 14 використаних джерел.

Метою випускної кваліфікаційної роботи є вивчення особливостей розробки містобудівного ігрового додатку для операційної системи Windows. Даний ігровий додаток спрямований на розважальний ігровий контент.

В ході розробки містобудівельного ігрового додатку для операційної системи Windows були проаналізовані сучасні рушії для створення ігрових додатків. Був детально проаналізоване ігровий рушій Unity, була обрана мова програмування для рушія Unity, ознайомилися з середою розробки, був створений містобудівельний ігровий додаток для операційної системи Windows, було проведено тестування додатку. Вироблено опис процесу розробки. Реалізовано, а також описаний користувальницький інтерфейс, ігрові механіки та саундтрек гри. Продемонстровано результат виконаної роботи.

Система задовольняє всім вимогам, пред'явленим в технічному завданні.

ЗМІСТ

ВСТУП

Актуальність. Розробка містобудівних ігор залишається актуальною через велику базу фанатів та шанувальників. Вони надають можливість гравцям створювати та керувати власними містами, розвивати їх та задовольняти потреби віртуальних жителів. Цей жанр є популярним серед гравців різного віку та ігрових вподобань. Розробка містобудівних ігор вимагає творчого мислення та інноваційних рішень. Розробники стикаються з викликом створення реалістичних та захоплюючих ігрових світів, а також забезпечення інтерактивності та глибини геймплею.

Об’єкт дослідження: містобудівний ігровий додаток для ОС Windows.

Предмет дослідження: розробка ігрового додатка на рушії Unity.

Мета дослідження: метою даної дипломної роботи є розробка містобудівельної гри для ОС Windows.

Задачі дослідження:

1. Ознайомитися з принципом роботи сучасних ігрових рушіїв.

2. Реалізація системи будівництва та знесення доріг та будинків.

3. Реалізація системи збереження ігрової сесії та її завантаження.

4. Створення й адаптація ігрової валюти та населення до поточного проекту.

5. Тестування містобудівельної гри.

1. Аналітичний огляд

1.1 Розробка додатків для особистих комп’ютерів

Ігрові рушії - це програмне забезпечення, яке використовується для розробки комп'ютерних ігор. Існує багато видів ігрових рушіїв, кожен з яких має свої особливості та функціонал. Ось деякі з них:

Unity - це один з найпопулярніших ігрових рушіїв, який використовується для створення ігор для різних платформ, включаючи ПК, консолі, мобільні пристрої і віртуальну реальність. Unity має величезну спільноту розробників, що забезпечує підтримку, навчання та доступність безкоштовної версії для початківців.

Unreal Engine - це ігровий рушій, що використовується для створення AAA-ігор для різних платформ. Він має потужний інструментарій для розробки графіки та фізики, а також підтримує віртуальну реальність. Unreal Engine є більш складним у використанні, ніж Unity, але його можливості більш високі.

CryEngine - це ігровий рушій, який спеціалізується на створенні ігор з високоякісною графікою, особливо в області реалістичної графіки. Він має потужну підтримку геометрії, освітлення та тіней, а також можливості для розробки штучного інтелекту.

GameMaker Studio - це ігровий рушій, який спрощує процес розробки ігор для початківців. Він має графічний інтерфейс, що дозволяє розробникам зосередитися на геймплеї та історії, не витрачаючи час на складний код.

Construct - це ігровий рушій, який також спрощує процес створення ігор, зосереджуючись на графічному інтерфейсі та функціях перетягування та розміщення об'єктів. Цей рушій підходить для швидкого прототипування та розробки малих ігор.

Godot - це відкритий ігровий рушій, який має безкоштовну ліцензію та підтримує розробку ігор для різних платформ. Godot має потужний двигун графіки та фізики, а також можливості для розробки віртуальної реальності та штучного інтелекту.

Кожен з цих ігрових рушіїв має свої особливості та переваги, які залежать від потреб розробника та проекту. Для вибору найкращого ігрового рушію потрібно враховувати такі фактори:

Мета ігри: Якщо мета ігри - це розробка мобільної гри, то слід вибрати рушій, що підтримує мобільні платформи. Якщо мета - розробка ігор для консолей, то необхідно вибрати рушій, що підтримує консольні платформи.

Навички розробника: Якщо розробник має досвід роботи з певним рушієм, то це може бути важливим фактором у виборі.

Характеристики рушія: Рушій повинен мати необхідні характеристики для розробки певного типу ігор, наприклад, підтримку фізики, анімації, штучного інтелекту тощо.

Масштаб проекту: Великі проекти можуть вимагати використання більш потужних рушіїв з більшою функціональністю та можливостями для співпраці розробників.

Ліцензування та вартість: Вартість рушія та умови ліцензування можуть бути важливими факторами у виборі рушія для проекту.

Підтримка та спільнота: Рушій повинен мати досить велику спільноту розробників та бути підтримуваним розробниками для вирішення проблем і допомоги у вивченні рушія. Спеціалізовані можливості: Деякі рушії мають спеціалізовані можливості для розробки ігор у певних жанрах, наприклад, рушій для створення ігор-стратегій або ігор-рольових.

У кінцевому підсумку, вибір ігрового рушія залежить від потреб та можливостей проекту, досвіду розробника, бюджету та ліцензування рушія, підтримки спільноти розробників та інших факторів. При виборі ігрового рушія слід враховувати не тільки поточні потреби, а й майбутні перспективи розробки проекту. Важливо також розуміти, що рушії мають свої переваги та недоліки, і вибір повинен бути зроблений з урахуванням конкретних вимог проекту та потреб розробника.

1.2. Дослідження рушія Unity

Unity - це один з найпопулярніших ігрових рушіїв у світі, який дозволяє розробляти ігри для різних платформ, включаючи мобільні пристрої, комп'ютери, консолі та віртуальні реальності. Основні особливості Unity включають:

Кроссплатформеність: Unity дозволяє розробляти ігри для різних платформ, таких як Windows, MacOS, Linux, Android, iOS, Xbox та Playstation.

Візуальний редактор: Unity має візуальний редактор, який дозволяє створювати графічні об'єкти, розміщувати їх на сцені, редагувати анімацію та ефекти.

Фізика: Unity має потужний рушій фізики, який дозволяє створювати реалістичну поведінку об'єктів в ігрі.

Можливості штучного інтелекту: Unity має функціонал для розробки штучного інтелекту, що дозволяє створювати складні системи поведінки для NPC та інших об'єктів.

Мови програмування: Unity підтримує декілька мов програмування, включаючи C#, JavaScript та Boo. Однак, для розробки великих та складних проектів рекомендується використовувати C#, який є основною мовою програмування для Unity.

Unity є популярним вибором для розробки ігор завдяки своїм потужним можливостям та легкості використання. Його рушій фізики та можливості для розробки штучного інтелекту роблять його ідеальним вибором для створення ігор різного рівня складності.

1.3. Дослідження мови програмування C#

C# (C Sharp) - це мова програмування, розроблена компанією Microsoft, яка використовується для розробки різних додатків та програм, включаючи ігри.

У рушії Unity C# використовується як основна мова програмування для створення ігор. Unity має вбудовану середу розробки, яка підтримує розробку на C# та дозволяє розробникам легко створювати різноманітні функції та взаємодії між об'єктами гри.

C# має багато переваг для розробки ігор у Unity, включаючи:

Об'єктно-орієнтована мова: C# підтримує об'єктно-орієнтоване програмування, що дозволяє розробникам створювати код, який легко зрозуміти та модифікувати.

Висока продуктивність: C# компілюється до машинного коду, що дозволяє досягнути високої продуктивності гри та знизити навантаження на систему.

Легкість використання: Синтаксис C# легкий для вивчення та використання, що робить його добрим вибором для початківців.

Велика спільнота: Unity має велику спільноту розробників, яка допомагає вирішувати проблеми та надає багато ресурсів для вивчення C# та Unity.

1.4 Технічне завдання на розробку додатка

В результаті виконання даного дипломного проекту повинен бути створений ігровий додаток жанру містобудівельної гри , що дозволить користувачам відтворювати роль міського планувальника та керувати розвитком віртуального міста, забезпечуючи комфортне життя його мешканцям та розвиток міста в цілому. Для досягнення чого потребується зробити такі завдання:

- Ознайомлення з потенціалом ігрового рушія Unity
- Розробити геймплей містобудівної гри, що дозволить користувачам керувати розвитком міста. Геймплей повинен включати в себе планування доріг, інфраструктури, вибір місця будівництва
- розробити систему економіки міста, що дозволить гравцям здійснювати інвестиції та отримувати прибуток від розвитку міста. Це включає в себе управління бюджетом міста
- розробити зручний інтерфейс, який допоможе побачити користувачу загальні ресурси, інструменти по взаємодії зі світом та систему збереження

2. Модель проекту та постова задачі

Модель — це проект, інформаційне, натурна-матеріальне чи описово-макетне уявлення предмета. Об'єкт або явище, що є тотожною чи спрощеною версією модельованого об'єкта, проекту чи явища (прототипу)

2.1 Проектування

Проектування гри - це процес створення концепції та дизайну ігрового досвіду. Цей процес включає розробку геймплею, створення персонажів, рівнів, графіки та звукового супроводу.

Кожна гра, щоб бути реалізованою, потребує ідеї. Ідея – це проект всієї команди розробників, і вона може черпати натхнення хоч звідки. Генератори ідей вирішують до якого жанру буде ставитися гра, а далі вже відбувається робота відповідно. Концепція, обрана дизайнерами, може бути оригінальною ідеєю, продовженням або може бути заснована на якийсь ліцензії. Ліцензія може бути заснована на книзі, фільмі або конкретних людях і персонажах.

- Концепція гри буде заключати у розвитку свого місця на обмеженої по розмірам мапі, це буде поєднанням двох жанрів бездіяльності та містобудівельної гри.
- Розробка геймплею: в цієї грі буде реалізовано 3 головних механіки: будівництво, збереження ігрового прогресу та його завантаження та останнє механіка внутрішньо ігрової валюти, яка потребується при кожному будівництві.
- Дизайн рівня: це буде простою мапою без складнощів для гравця.
- Візуальний дизайн: він буде схожий на мінімалізм у деталізації та трохи дитячий у кольорах.
- Звуковий дизайн: буде складатися з звукових ефектів, що будуть посилювати конкретні дії гравця та музики, що буде уписуватись до концепту гри.
- Інтерфейс: буде інтуїтивно зрозумілим, не великим та не відволікати собою гравця від ігрового процесу.

2.2 Проектування механіки будівництва

Механіка будівництва – це механіка, що дозволяє гравцю розміщувати певні екземпляри об’єктів на мапі. Вона буде ґрунтуватися на системі сіток – це система яка дозволяє розміщувати екземпляри об’єктів лише по клітинкам, що дозволяє уникнути таких проблем, як великій потік даних, через те що

координати у коді будуть завжди цілочисленними тим самим зменшивши навантаження на комп’ютер гравця та проблеми «колізій», коли один об’єкт опиняється в іншому.

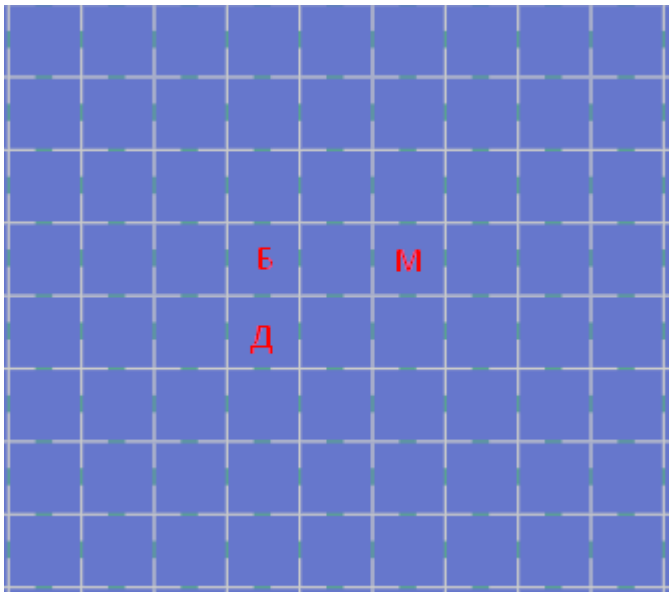

Рисунок 1. Вигляд системи будування на сітці, де кожен об’єкт може займати лише відповідну клітинку чи клітинки. Де Б – будинок, Д – дорога та М – магазин.

Як ми бачимо головним недоліком цієї системи є відсутність якоїсь логікі розміщення об’єктів: коли Б розміщується у дороги, а М – ні. Для цього приходить на допомога друга система Графів, де кожна клітинка розпізнається як вузел, а пряма, що з’єднує їх – ребро.

У такій системі можна налаштувати зв’язки вулів один з одним, що додає логіку у систему будування, тепер перевіряючи через ребра вузли сусідів можна ускладнити систему будівництва додавши умову, що у будь-якому сусідньому вузлі повинна бути дорога.

Тому якщо використовувати ці дві системи можна використовувати переваги обох систем: розміщування об’єктів по сітці, при чому використовуючи логіку та розташувати лише за виконаних умов

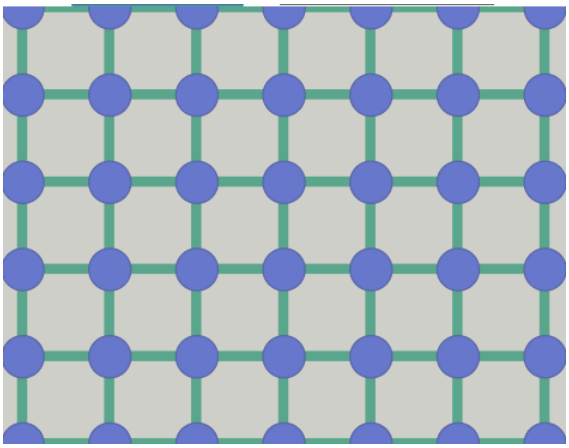

Рисунок 2. Система графів, де сині кулі – вузли та зелені лінії, що їх з’єднують – ребра.

2.3 Звукові ефекти та музика у ігрі

Відеоігри складаються з трьох компонентів: зображення, інтерактивність, звук. Якщо перші два компоненти формують фундамент гри, то звук і музика трансформують гру в щось більше, ніж просто дію.

Планування ідеального пограбування, емоційна втрата улюбленого персонажа, перегони, що захоплюють дух, – музика збільшує імерсивність гри, змушуючи гравця занурюватися з головою в іншу реальність.

Музика в іграх і навіть у кінематографі змішується з фоном і стає невіддільною частиною загальної картини. За допомогою саунд-дизайну можна "докрутити" досвід гравця і збільшити імерсивність.

З чого складається саундтрек в іграх? Ігровий саундтрек складається з декількох елементів:

- Музика;
- Звуковий інтерфейс (музична складова зворотного зв'язку);
- Фонові звуки;

Що повинен відчувати гравець, перебуваючи в глибинах підземелля? Розробник може допомогти гравцеві в дослідженні світу за допомогою звуків.

Якими вони мають бути?

- зацикленими,
- такими, що не дратують,
- відповідними до атмосфери.

Із цього робимо висновок, що музика у містобудівної грі повинна мати наступне:

1. Покровительський та веселий: в містобудівної гри зазвичай переважає позитивний настрій, тому музика повинна бути веселою. Вона повинна підіймати настрій гравцю.
2. Інструментальна музика: оскільки містобудівна гра зазвичай має фоновий характер, інструментальна музика може бути більш підходящою, ніж пісні з вокалом. Мелодійність інструментів може допомогти поглинути гравця в ігровий світ.
3. Різноманітність та пов’язаність до подій: музика може змінюватись залежно від подій в грі. Наприклад, коли гравець достягає успіху музика може підкреслити цю радісну подію.
4. Атмосферність: музика повинна відображати атмосферу гри, тобто вона повинна мати елементи, які можуть нагадувати наприклад звуки міста.
5. Циклічність: музика бути не виразна у момент переходів і не набридати при циклічному повторі.

Тому із існуючих жанрів музики більш за все підходить до містобудівної гри такі жанри музики: lo-fi та relax. Жанр музики "relax" (релакс) відповідає за створення спокійної та розслаблюючої атмосфери. Музика в цьому жанрі призначена для відпочинку, зняття стресу та сприяння загальному релаксу. Жанр музики "lo-fi" (скорочено від low fidelity) є популярним жанром, який характеризується особливою атмосферою та звуковою естетикою.

3. Створення містобудівної гри

3.1 Підготовка матеріалів до початку створювання

Для початку перед тим як почати створювати гру нам потребуються матеріали з якими ми будемо працювати. Для цього проекту мені потребуються: моделі з їх префабами, іконки, музику та звукові ефекти.

3.1.2 Модельки(будівлі, природа)

Перше що нам знадобиться це моделі і їх префаби. Префаб (prefab) – це шаблон для об'єкта в ігровому рушії Unity. За допомогою префабів можна створити "зразок" предмета з певними властивостями, а потім використовувати такі предмети на всій ігровій сцені. Якщо змінити префаб, то зміняться всі об'єкти, створені на його основі. У нашому випадку префабами будуть: дерева, кущі, каміння, споруди та дороги. Збір кількох префабів, звукових ефектів, спрайтів, механік, тощо – називається «асетом».

Є декілька варіантів знайти префаби: на офіційному веб-сайті Unity Asset Store та на інших веб-сайтах, які також як Unity створили маркетплейс орієнтований на розробников ігор. На таких веб-сайтах можна зайти як одну модель чи префаб, так і цілі збірки. І так з усім, що розташовано на маркетплейсі – трьох чи двох-вимірні моделі, спрайти(картинки), звукові ефекти, візуальні ефекти, готові збірки, механіки гри та інструменти. Там кожна одиниця може поширюватися двома способами: безкоштовно та за одноразову плату.

Різниця між офіційним маркетплейсом та іншими веб-сайтами у способі імпортування матеріалів – дерева, кущі та каміння, що я обрав у Unity Asset Store імпортуються через Packege Manager, якій зберігає усі додані асети з Unity Asset Store. Вони можуть не бути у проекту, але користувач завжди зможе їх швидко імпортувати у будь-який проект через Packege Manager. Для цього потрібно ввійти в лаунчері у свій особистий аккаунт Unity. Так я і додав

дерева і кущі. Приклад можна подивитись у пункті 3.2.1, бо для того щоб подивись, що у асеті, потрібно мати проект і дивитися у ньому після імпорту асета – це один з мінусів такого способу.

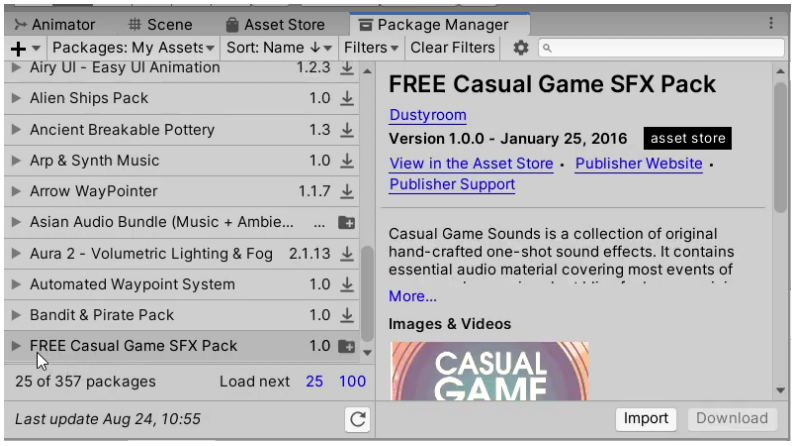

Рисунок 3. Вкладинка Packege Manager, де імпортуються асети з Unity Asset Store

Другий спосіб додати це знати інший маркетплейс орієнтований на розробників ігор, наприклад веб-сайт Kenney.nl. Це такий же веб-сайт, як і веб-сайті Unity Asset Store, але він відрізняється способом імпортування, та товари, що там розміщуються. Звідти я взяв моделі та префаби доріг та споруд. Щоб імпортувати їх, потрібно завантажити з веб-сайту архів з файлами обраного асету. Після чого коли ми забажаємо імпортувати асет, потрібно обрати у редакторі на панелі інструментів самого редактора обрати «Assets», далі у ньому знайти й обрати «Import Package», а вже ньому «Custom Package» (рис. 3). Або інший варіант просто перетягнувши з провідника Windows папку з асетом, до провідника редактора Unity.

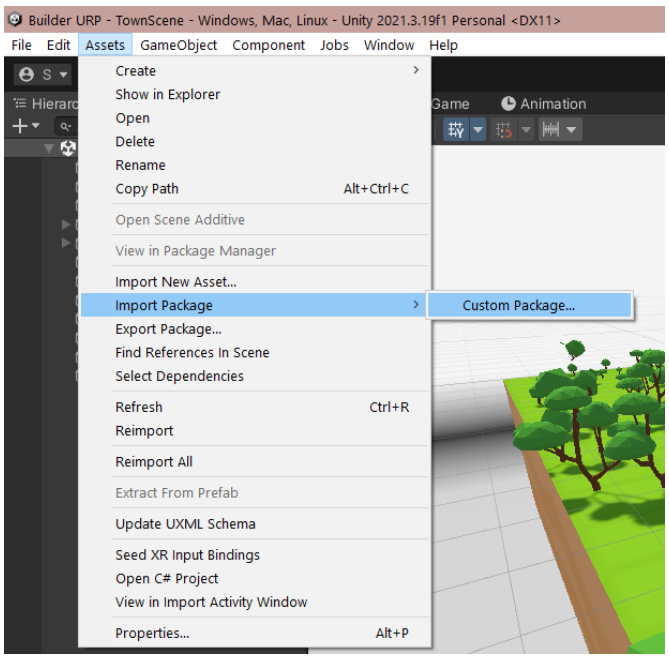

Рисунок 4. Другий шлях імпортування асету через «імпорт пакетів»

3.1.3 Іконки

Далі іконки, спрайти – звичайні малюнки, що використовуються у редакторі Unity з різних підстав. Є багато веб-сайтів загострених на розміщення й розповсюдженні іконків та спрайтів починаючи з офіційного сайту асетів Unity, до не менш популярних сайтів. Я буду використовувати веб-ресурс Flaticon.com.

Для моєї гри мені потрібно декілька іконок:

- меню
- інструменти – для вкладки будівництва
- бульдозер – для кнопки знесення
- іконка будинка
- іконка дороги
- іконка магазину/комерції
- іконки звуку для налаштування гучності

А ось деякі іконки, що підійшли до проекту:

Рисунок 5. Іконка меню

Рисунок 6. Іконка бульдозеру

Рисунок 7. Іконка будинку

Рисунок 8. Іконка магазину/комерції

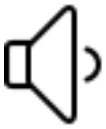

Рисунок 9. Іконка увімкненого звуку

Рисунок 10. Іконка вимкненого звуку

3.1.4 Музика та звукові ефекти

Музика в іграх відіграє дуже важливу роль, допомагаючи створювати атмосферу та посилювати емоції гравців. Вона може бути використана для підкреслення дії, створення напруги, встановлення настрою або навіть для розслаблення гравця в спокійних моментах.

В моєму проекту будуть використовуватися не великий список звукових ефектів: звук будівництва споруди, звук знесення споруди, звук надходження прибутку. У цьому мені допомогли 3 веб-ресурси: zvukipro.com – де було знайдено звук знесення споруди; pixabay.com – де було знайдене звук надходження прибутку, у моєму випадку це звук дзвін монет; Unity Asset Store – було знайдено асет «Free Casual Game SFX Pack» з збіркою звуків будівництва споруд..

Щодо музики, як я упоминав в теорії музики в іграх у пункті 2.3 до геймплею та жанру нашої гри підходять музика жанру: релакс та lo-fi. Головним веб-ресурсом музики, що я обрав став artlist.io через велику бібліотеку безкоштовної музики без заборони на копіювання.

3.2 Створення гри в рушії Unity

Перше що потребуєтеся зробити, так це завантажити потрібну версію редактора Unity. Unity надає кілько доступних версій редакторів серед яких є бета версії та релізні. Бета версії потрібні для опробування нових нововведень, але ці версії не стабільні та можуть часто викликати помилки та вимкнення редактору без збережень поточних змін, що веде до згаяного часу. Тому усі розробники ігор вибирають релізні версії – це версії, у яких нововведення відполіровані, а нові та й старі критичні помилки виправлені. Такі версії мають довготермінову підтримку, тому навіть помилки або не сумісності, що не були знайдені за бета-тестування будуть виправленні теж.

3.2.1 Створення початкової сцени та

Після встановлення редактора в лаунчері Unity з’являється можливість створити проекти. Обраємо “New Project” для того, щоб створити новий проект.

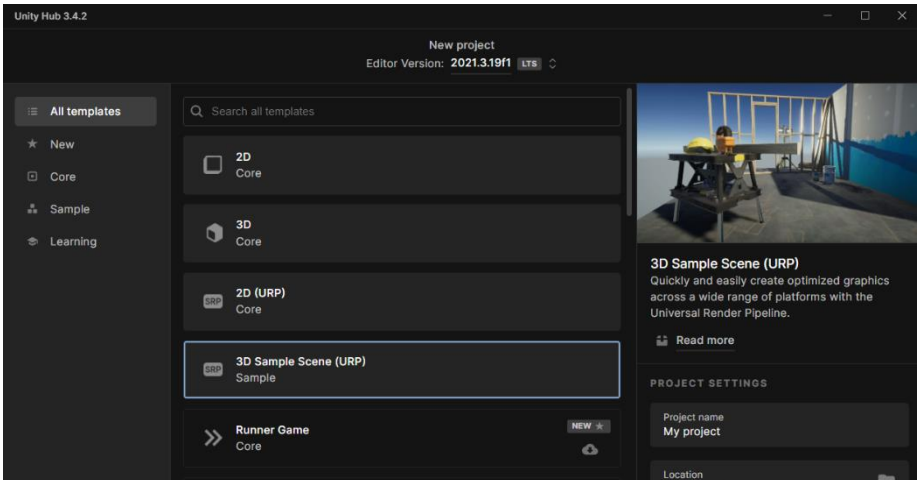

Рисунок 11. Вкладка вибору типу, назви та шляху збереження проекта

В новій вкладці серед пропонованих типів проекту обраємо «3D Sample Scene (URP)» так як він більш за усіх підходить. Називаємо проект, та обираємо розташування на особистому комп’ютері, після чого тиснімо «Create Project» та очікуємо поки редактор створить проект.

Коли завершиться створення проекту, він відчиниться у редакторі Unity, що ми в становили. Наступним кроком буде створення потрібних папок за для розташування матеріалів, скриптів, моделей та сцен та зручного знаходження потрібного.

У низу редактора Unity буде робоча область, де розташована провідник «Project» (далі провідник), консоль, а також туди ми додамо вкладку анімації. Для цього требо натиснути на «Project» правою кнопкою миші та обрати у контекстному меню «Add Tab», а далі «Animation». Вона нам понадобится на етапі створення інтерфейсу користувача.

Отже повертаємося до вкладки провідника та створюємо наступні папки: «_Script», «Script», «Animations», «Image», «Materials», «Models», «Prefabs» та «Scenes». Кладемо у папки Materials, Models, Prefabs відповідні матеріали та моделі що ми вибрали раніше. У папці Script та _ Script будуть зберігатися скріпти – тобто файли з кодом на мові C#. У Image – будуть зберігатися усі зображення та спрайти що будуть використовуватись у проекті. У папці Animations будуть зберігатися усі анімації але про це я розповім детальніше пізніше.

Останнє, що залишилось додати до нашого проекту так це ассет зі звуками, які ми будемо використовувати при будівництві чи при знесенні. Щоб їх додати потребується відкрити вкладку «Asset Store» у центральній робочій області. Відкриється магазин асетів Unity, де ми повинні знайти безкоштовний додаток «Free Casual Game SFX Pack», окриваємо його та натискаєм кнопку «Open in Unity», після чого у браузері появиться вікно з запитом на дозвіл – тиснемо «Allow». Нас перекидає до редактору, де на

центральній робочій області відчиняється вкладинка «Packege Manager», де нам потрібно знайти наш доданий асет та імпортувати його кнопкою «Import».

Тепер можна перейти до створення платформи там землі на, які ми будемо розміщувати будинка. В ієрархії натискаємо праву кнопку миші (далі ПКМ), у контекстному меню обираємо Empty GameObject та називаймо його «Terrain» у інспекторі. У ньому таким ж чином створюємо пустий об’єкт, але з назвою «Nature». Далі натискаємо ПКМ на Terrain та створюємо Plane та Cube. Cube у інспекторі називаємо Ground. На них у інспекторі накладаємо Mesh Collider та Mesh Render та робимо наступні налаштування як показано на малюнку 2 й 3. Також у параметрі Scale робимо наступні налаштування: для plane – x: 1.5; y: 1; z: 1.5; для ground – x: 15; y: 3; z: 15, а також розташування по y: -1.51;

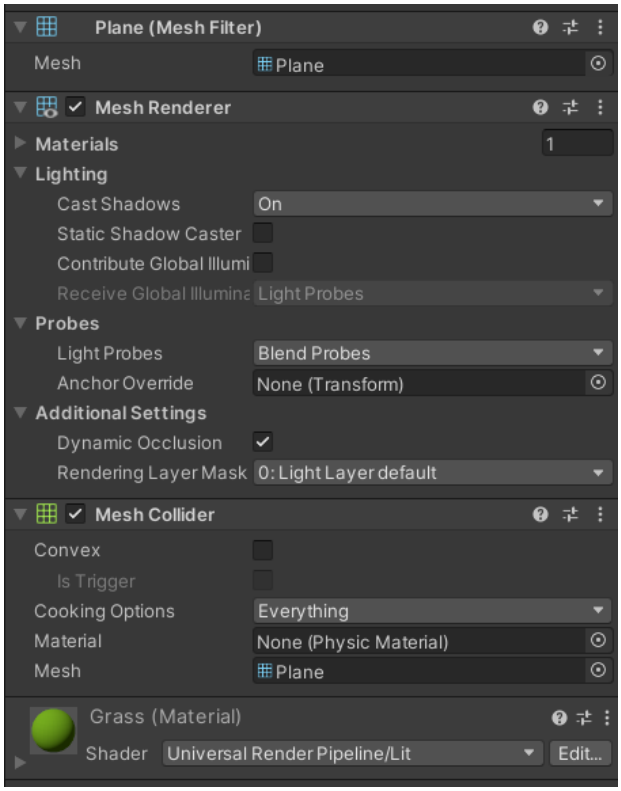

Рисунок 12

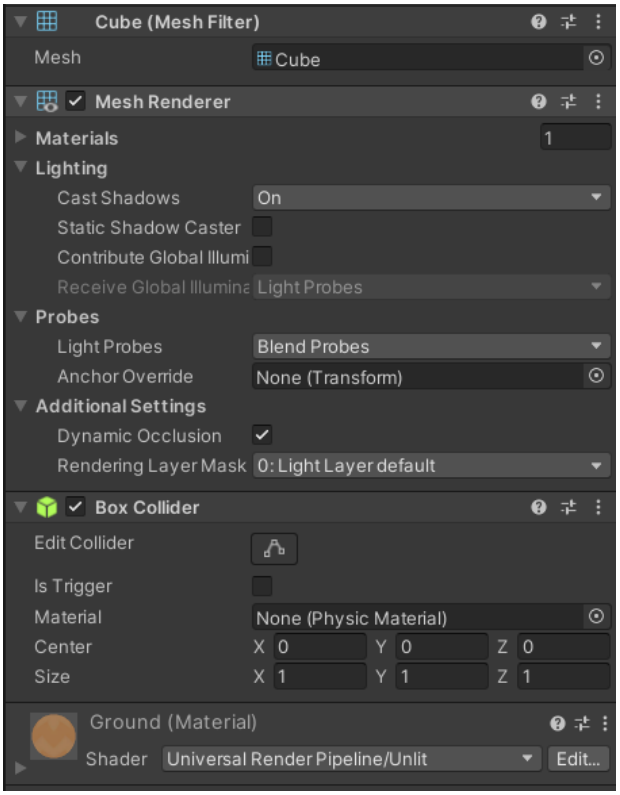

Рисунок 13

Рис. 12-13 – налаштування Mesh Render та Mesh Collider для:
12 – Plane; 13 – Ground.

На отриманому острівці розташовуємо префаби (моделі) кущів, дерев й камінь так щоб в ієрархії вони були всередині об’єкту Nature. Результатом цього отримаєм острівець, як на рис. 14.

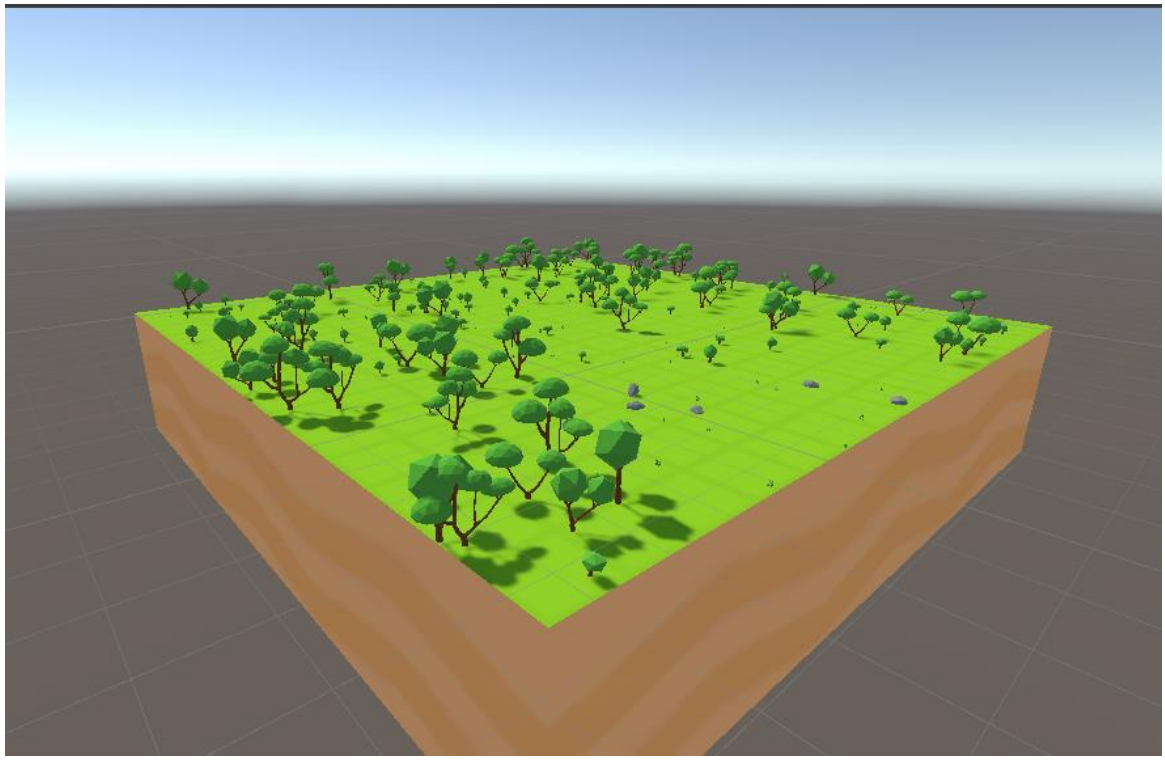

Рисунок 14. Результат розміщення дерев, кущів й каміння

Ще що потрібно зробити з початковою сценою так це видати рівні але перед тим, як їх видати, їх потрібно створити. Над вкладкою інспектора розташовується кнопка Layers. Там требо обрати Edit Layers, після чого нас перекине до налаштувань усіх рівнів де на рівень 8 требо вписати «Nature», а на 9 – «Ground».

Після того як ми створили рівні потрібно назначити які об’єкти на них будуть. Тож обираємо Plane та у інспекторі шукаємо Layers, де при натиску потрібно обрати рівень «Ground». Далі у ієрархії потрібно обрати теку-об’єкт Nature та надати йому рівень у інспекторі «Nature».

3.2.2 Створення системи керування

Механіка будівництва у цьому проекту – це велика сукупність скриптів, що складається з 10 скриптів, що виконують певну функцію. Але щоб розпочати створювати систему будівництва потрібно зробити

Почнемо з скрипта Camera Movement. Він використовує бібліотеку UnityEngine й потрібен за для налаштування руху та приближення камери, де cameraMovementSpeed – відповідає за швидкість руху, а OrthographicSize – за максимальне й мінімальне наближення камери. Реалізація скрипта:

```
        public Camera gameCamera;
        public float cameraMovementSpeed = 12;

        public float maxOrthographicSize = 5f, minOrthographicSize = 0.5f;
        public float sensitivity = 0.1f;

        private void Start()
        {
            gameCamera = GetComponent<Camera>();
        }
        public void MoveCamera(Vector3 inputVector)
        {
            var movementVector = Quaternion.Euler(0,30,0) * inputVector;
            gameCamera.transform.position += movementVector * Time.deltaTime *
cameraMovementSpeed;
        }

        private void Update()
        {
            var scrollInput = Input.GetAxis("Mouse ScrollWheel") * sensitivity;
            gameCamera.orthographicSize = Mathf.Clamp(gameCamera.orthographicSize
- scrollInput, minOrthographicSize, maxOrthographicSize);
        }
```

Цей скрип потрібно добавити до камери. Для цього потрібно обрати камеру у ієрархії, та у інспектору де кнопка додати компонент перетягнути скрипт з провідника до кнопки. А також поки ми займаємося камерою

потрібно додати компонент «Audio Listener», завдяки ньому гра буде розуміти де розташован гравець, та передавати тільки те, що може почути гравець.

Далі другий але не менш значний скрипт InputeManager, Він як й Camera Movement потрібно створити у папці Script. Він відповідає за зчитування натиснення кнопок, що використовуються у усіх механіках. Цей скрип використовує 2 бібліотеки UnityEngine й її підбіблеотеку EventSystems. Реалізація функції зчитування:

```
    public event Action<Ray> OnMouseClick, OnMouseHold;
    public event Action OnMouseUp, OnEscape;
    private Vector2 mouseMovementVector = Vector2.zero;
    public Vector2 CameraMovementVector { get => mouseMovementVector; }
    [SerializeField]
    Camera mainCamera;


    void Update()
    {
        CheckClickDownEvent();
        CheckClickHoldEvent();
        CheckClickUpEvent();
        CheckArrowInput();
        CheckEscClick();
    }

    private void CheckClickHoldEvent()
    {
        if (Input.GetMouseButton(0) &&
EventSystem.current.IsPointerOverGameObject() == false)
        {
          OnMouseClick?.Invoke(mainCamera.ScreenPointToRay(Input.mousePosition));
        }
    }

    private void CheckClickUpEvent()
    {
        if (Input.GetMouseButtonUp(0) &&
EventSystem.current.IsPointerOverGameObject() == false)
        {
            OnMouseUp?.Invoke();
        }
    }

    private void CheckClickDownEvent()
```

```
    {
        if (Input.GetMouseButtonDown(0) &&
EventSystem.current.IsPointerOverGameObject() == false)
        {
          OnMouseClick?.Invoke(mainCamera.ScreenPointToRay(Input.mousePosition));
        }
    }

    private void CheckEscClick()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            OnEscape.Invoke();
        }
    }

    private void CheckArrowInput()
    {
        mouseMovementVector = new Vector2(Input.GetAxis("Horizontal"),
Input.GetAxis("Vertical"));
    }

    public void ClearEvents()
    {
        OnMouseClick = null;
        OnMouseHold = null;
        OnEscape = null;
        OnMouseUp = null;
    }
}
```

Після написання й збереження скрипта потрібно його додати до сцени. Для цього потрібно створити GameObject та додати до нього скрипт InputManager, для цього перетягуємо скрипт з провідника до «додати компонент» у інспекторі й назвати його InputSystem.

3.2.3 Створення системи будівництва

Механіка будівництва у цьому проекті прив’язана до сітки координат і розташовані споруди мають лише цілочисельні значення у координатах. Так уникається проблема колізій, через яку споруда могла б задіти іншу, чи во всі опинитись у ній. А ще це допомагає зменшити навантаження на комп’ютер.

Ця механіка – це велика сукупність скриптів, які взаємозв’язані між собою. Механіка будівництва складається з будівництва будинків, доріг та функції видалення.

3.2.4 Функція будівництва доріг

Функція будівництва доріг у механіцизмі будівництва використовує частково загальні функції будівництва: на кшталт перевірки клітинки на доступність для будівництва чи перевірка вибраного місця чи не вийшло обране місце за межі масива (сітки).

Працює функція будівництва доріг наступним чином натиснув один раз на кнопку «Road» у грі скрип UI Controller перемикає у режим будівництва доріг – вмикається слухач натиску й відпуску лівої кнопки миші. На просте натиснення кнопки миші спрацьовує функція RoadPlacementHandler(), яка викликає функцію PlaceRoad() у скрипті RoadManager, де викликається перевірка зайнятості місця й не виходить місце за сітку, якщо усі перевірки пройдені успішно розміщається кінцева дорога

Рисунок 15. Модель кінцевої дороги на сірому фоні.

Після чого у кінці функції викликається функція FixRoadPrefabs(), яка викликає функції з скрипта RoadFixer. Суть скрипта RoadFixer у зміні розміщуваного моделі з кінцевої на ту, що відповідає логікі. Логіка розташовування тут така – в залежності від сусідніх кліток обирається моделька, яка відповідає кількості доріг-сусідів, повертається, якщо потрібно

до них та запускає туж саму перевірку на дорогах-сусідах що змінити їх під нову дорогу.

Перевірка на сусідів стала можливою у моєму проекті через «типи клітинок», які поділяються на: пусті, дома, крамниці, великі споруди(дома). І функція перевірки бере координати й дивиться значення у масиві типів-клітинок. Приклад роботи розміщення однієї дороги біля інших:

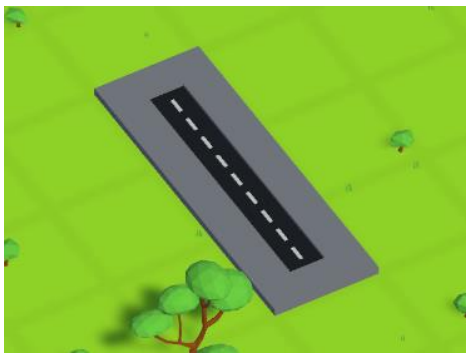

Рисунок 16. Для прикладу розташували 3 дороги у ряд.

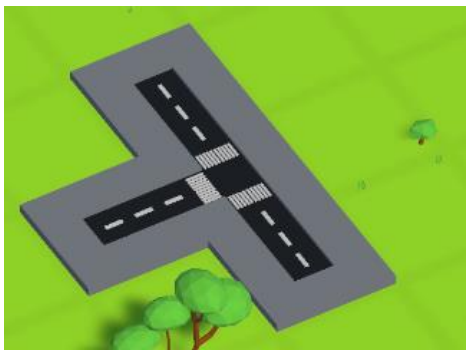

Рисунок 17. Біля центральної дороги з рис.6 розташували дорогу з ліва. Через перевірку у RoadFixer дорогу, що поставили розвернуло дорогою до права і запустило перевірку у центральній дорозі, що перетворило її з прямої дороги на т-подібну й повернуло модель у потрібну сторону.

Рисунок 18. Біля центральної дороги з рис.6 розташували дорогу з права, після зміни напрямку й розташування запустилась перевірка центральної дороги, яка перевіривши кількість сусідів змінила модель на чотиристоронню й розташував її.

У редакторі Unity скрипти RoadFixer та RoadManager зв’язані через один ігровий об’єкт на якому вони розташовані. У RoadManager прив’язане RoadFixer у слоті RoadFixer, а у RoadFixer у відповідних слотах прив’язані усі типи доріг, а саме префаби (моделі). Приклад зв'язків:

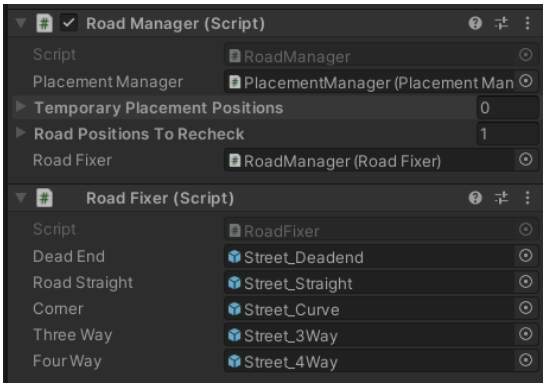

Рисунок 19. Налаштування об’єкта RoadManager

Існує і інший метод будівництва – прокладання довгої дороги. Цей метод використовує 3 функції лівої кнопки миші: OnMouseHold, OnMouseDown, OnMouseUp. Слухачем, якого э InputManager, що викликається у головному скрипті GameManager.

Прокладання доріг працює так – якщо, ліва кнопка миші(ЛКМ) затиснута, то спрацьовує 2 функції миші: на OnMouseDown – записується координати початку прокладання; на OnMouseHold – записується поточна кінцева точка прокладання доріг, при чому вона може змінюватись в залежності від зміни координат (лише цілочисельних).

Після запису 2 координат – початкових і кінцевих викликається roadManager, який прокладає шлях від одної координати до іншої, та перевіряє перешкоди на шляху, якщо вони є то він огинає перешкоди поки не зможе прокласти шлях до кінцевої координати. Після прокладання «шляху», починається розміщення на усіх його координатах тимчасових доріг. Ці тимчасові дороги розміщуються, як і в звичайному будівництві у вигляді кінцевих доріг. Далі запускається RoadFixer, який перевіряє логіку усіх розташованих тимчасових доріг і їх доріг сусідів, та замінює префаби на відповідні, розвертаючи їх по необхідності.

Усе це розміщування тимчасових доріг відбувається при кожній зміні кінцевих координат і завершуються лише при відтисканні ЛКМ (функція

OnMouseUp), тоді усі тимчасові дороги із списку тимчасових доріг видаляються і розміщуються вже постійні дороги. Приклад роботи прокладання доріг на рис.20-23.

Також потрібно зауважити, що коли розміщуються постійні і тільки постійні споруди (будинки, дороги) спрацьовує функція DestroyNatureAt(), вона використовує масив променів, які викликає до землі та записує результат зіткнення на шляху променів, де усі результати, що розташовані на рівні «Nature» видаляються. Саме тому усі дерева, кущі та каміння було записано до рівня «Nature».

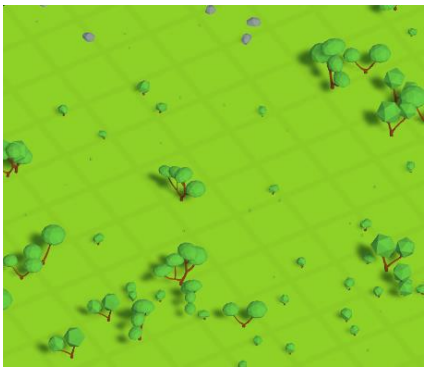

Рисунок 20. Пуста ігрова область до «прокладання доріг».

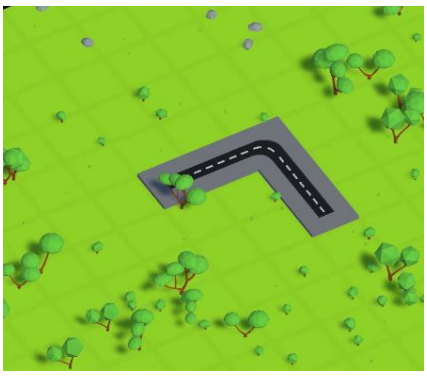

Рисунок 21. Результат прокладання тимчасових доріг після затискання ЛКМ та проведення миші у правий бік (Усі об’єкти з рівня «Nature»)

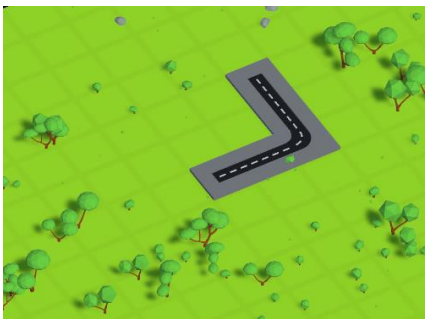

Рисунок 22. Результат продовження прокладання тимчасових доріг. На цей раз з затисненої кнопкою ЛКМ миш повели до гори та в ліво. Як ми бачимо тимчасові дороги змінились, а старі видалились.

Рисунок 23. Приклад уникнення перешкод при прокладання доріг. З червоною точки почали прокладання доріг та потягнули до синьої точки – у результаті дорога побачивши перешкоду у вигляді будинку спробувала уникнути, але побачила інший будинок й оминула його, як тільки вона його оминула від тієї точки проклався шлях до кінця.

3.2.5 Функція будівництва будинків

Функція будівництва будинків являє собою систему для розміщування копії префабів, саме копій бо кожне «будівництво гравця» - це копіювання відповідного префаба та розміщення його на мапі та у пустому об’єкті «Structure», який має свої налаштування.

Вона працює так коли ми обираємо будівництво будь-якій споруди через інтерфейс, наприклад кнопка «House», увіткається слухач натиску лівої кнопки миші і при натиску (функція OnMouseClick) запускається відповідна функція. У нашому прикладі запускається функція «HousePlacementHandler», яка виглядає наступним чином:

```
{
   ClearInputActions();

   inputManager.OnMouseClick += (pos) =>
   {
       if (money >= costHouse)
        {
             ProcessInputAndCall(structureManager.PlaceHouse, pos);
        }
   };
   inputManager.OnEscape += HandleEscape;
}
```

Функція «HousePlacementHandler» бере координати курсору на мапі через InputManager, та проходячи перевірку наявність грошей (детальніше в пункті 3.2.5) викликає функцію PlaceHouse з скрипта structureManager.

```
public void PlaceHouse(Vector3Int position)
```

```
    {
        if (CheckPositionBeforePlacement(position))
        {
            int randomIndex = GetRandomWeightedIndex(houseWeights);
            placementManager.PlaceObjectOnTheMap(position,
housesPrefabe[randomIndex].prefab, CellType.Structure,
buildingPrefabIndex:randomIndex);
            AudioPlayer.instance.PlayPlacementSound();
            gameManager.AddOneBuildingOfPopulation();
        }
    }
```

У свою чергу функція PlaceHouse вирішує випадково який префаб будинку з будинків буде використовуватись, запускає функцію PlaceObjectOnTheMap, викликає звуковий ефект будівництва споруди (детальніше у 3.2.8) та викликає функцію з головного скрипта, що додає популяцію (детальніше у 3.2.5). Але щоб усе це відбулось потребується пройти перевірку, яка перевіряє дорогу в клітці поряд, бо в моєї логікі споруди можуть будуватися лише поряд доріг, та перевірка на те чи вільно та доступне місце.

Якщо обране місце буде за масивом кліток сітки з’явиться повідомлення у консолі «This position is out of bounds». Якщо обране місце буде зайнято у консолі також з’явиться надпис «This position is not EMPTY». Та й остання перевірка дороги поряд з клітинкою будівництва, якщо її не пройдено з’являється напис у консолі «Must be placed near a road». Якщо будь-яка перевірка не буде пройдена усе розміщування обривається, та повертається з певним повідомленням у консолі.

Останнім етапом є функція PlaceObjectOnTheMap, яка є універсальною функцією для будівництва будь-якого будинку чи дороги. Вона створює пустий ігровий об’єкт з відповідною назвою (Structure чи Road), та створює у ній копію префаба, розташовує в відповідному місці, також змінює логічний тип клітинки на відповідну до типу префаба (Road, House, Special, BigStructure) та й останнє запускає функції DestroyNatureAt, яка убирає усі елементи природи.

Щоб додати префаб до певного типу ігрових споруд (дома, магазини), треба в ієрархії у пустому об’єкту StructureManager, додати скрипт StructureManager. У ньому будуть розгортаємі типи споруд , якщо требо додати споруду, наприклад до магазинів, требо розгорнути «SpecialPrefabs» та натиснути на плюс чи збільшити число напроти написав більше число. Тоді з’явиться слот з пустим місцем під префаб та налаштуванням ширини префабу. Потрібний префаб з провідника редактору потрібно перетягнути у слот з пустим значенням префабу, для в залежності від розмірів моделі зменшити її. Приклад налаштування об’єкту StructureManager:

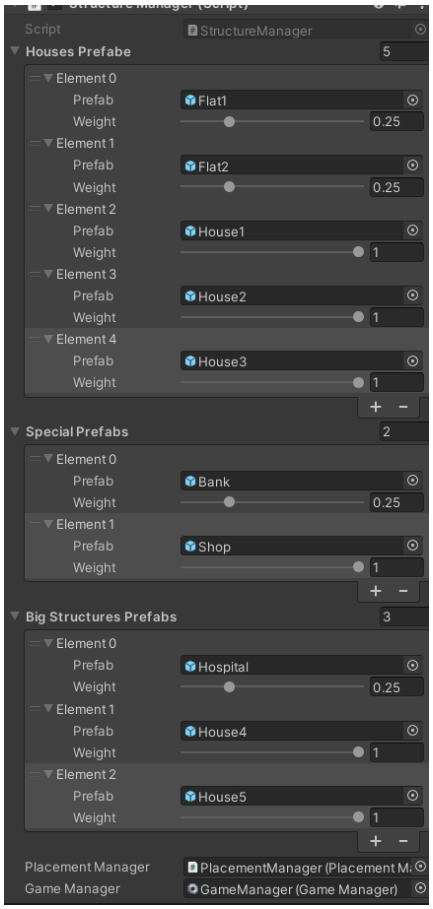

Рисунок 24. Налаштування об’єкту StructureManager:

- У House Prefabs – 5 префабів будинків
- У Special Prefabs – 2 префаба комерції
- У Big Structure Prefabs – 3 префаба великих будинків

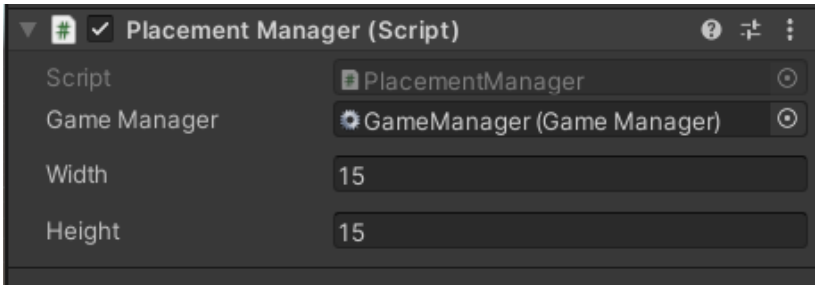

Рисунок 25. Налаштування об’єкту Placement Manager: у якому можна змінити ширину та висоту ігрової сітки. На даний час розмір сітка 15 на 15 кліток.

3.2.6 Функція видалення

Функція видалення, щоб звільнити лише одну клітку, а не усю мапу, потрібно якось знайти префаб, видалити його та змінити логічний тип клітинки на «empty».

Для цього використовується функція DestroyBuildingAt, вона викликає промені над клітинкою, до землі і якщо вона з чимось з рівня «Default» зіткнеться, це запишеться у масив. Приклад виклику променів:

```
RaycastHit[] hits = Physics.BoxCastAll(position + new Vector3(0, 0.5f, 0), new
Vector3(0.2f, 0, 0.2f), transform.up, Quaternion.identity, 0.5f, 1 <<
LayerMask.NameToLayer("Default"));
```

Де значення BoxCastAll: початок променю у 3 вимірах, кінець променю у 3 вимірах, по горизонту, направлення, довжина промені, час та рівень об’єктів, які потрібно записувати.

Після чого кожен результат перебирається у циклі, де він перетворюється у ігровий об’єкт, а об’єкту визначається об’єкт батько, якщо його ім’я підходить (Structure, Road, SpecialStructure), то виконується логічний процес системи грошей та населення, потім змінюється логічний тип клітинки на пусто, видалення ігрового об’єкту на мапі і ієрархії та видалення даних в масиві даних споруд. Приклад функції DestroyBuildingAt:

```
{
        RaycastHit[] hits = Physics.BoxCastAll(position + new Vector3(0, 0.5f,
0), new Vector3(0.2f, 0, 0.2f), transform.up, Quaternion.identity, 0.5f, 1 <<
LayerMask.NameToLayer("Default"));

        foreach (var item in hits)
        {
            var a = item.collider.gameObject;
            GameObject foo;
            foo = a.transform.parent.gameObject;
            if (a.transform.parent.name == "Structure" || a.transform.parent.name
== "Road" || a.transform.parent.name == "SpecialStructure")
            {
                switch (a.transform.parent.name)
                {
```

```
                case "Structure":
                    gameManager.RemoveOneBuildingOfPopulation();
                    break;
                case "SpecialStructure":
                    gameManager.RemoveOneBuildingOfIncome();
                    break;
                default:
                    break;
                }
                placementGrid[position.x, position.z] = CellType.Empty;
                Destroy(foo);
            }
            Debug.Log("+Name Parent: " + item.transform.parent.name);
            if (a.transform.parent.parent.name == "Road")
            {

                placementGrid[position.x, position.z] = CellType.Empty;
                foo = a.transform.parent.parent.gameObject;
                Destroy(foo);
            }
            Debug.Log("+Name PParent: " +
item.transform.parent.parent.name);
        }
        foreach (var structure in GetAllStructures())
        {
            if (structure.Key == position)
            {
                if(structureDictionary.ContainsKey(position))
                {
                    Debug.Log(position+" - "+ structure.Key);
                    structureDictionary.Remove(position);
                }
            }
        }
    }
```

Рисунок 26

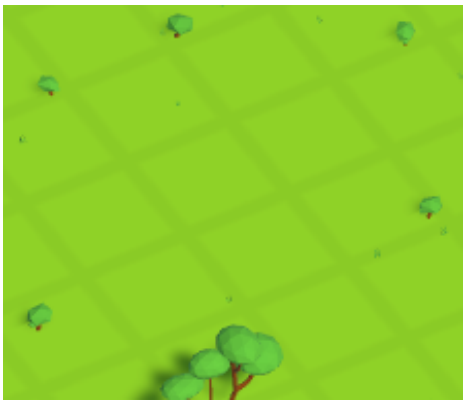

Рисунок 27

Рисунок 26-27. Приклад роботи системи видалення очима гравця: рис. 26 – гравець розташував дорогу та будинок; рис.27 – гравець видалив будинок, а потім дорогу.

Приклад роботи вибору випадкового префаба одного типу при будівництві:

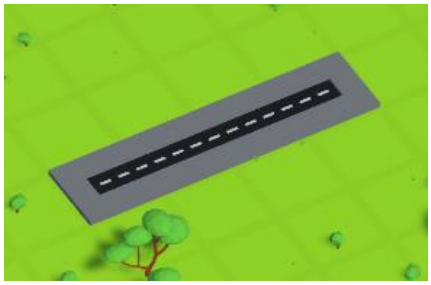

Рисунок 28. Гравець розташував у лінію 4 дороги.

Рисунок 29. Гравець розташував зверху та знизу від дороги по 4 будинка. З 5 префабів будинків використались лише 3. Це не погано, бо при вибірці лише у 8 будинків застосувалось 3 різних префаби.

Хочу зауважити що видалення споруд не повертає ігрові об’єкти природи бо ті були видалені, а не вимкнуті, що робить гру більш суворішим до бездумного будівництва.

3.2.7 Система грошей та населення

Оскільки ця гра буде сумішшю кількох жанрів, а саме: містобудівні, стратегії та бездіяльність – потрібно якось ускладнити тим самим розтягнути геймплей та зацікавити гравця. І цим ускладненням буде система грошей та населення.

Щоб гравець міг розмістити будь-яку споруду йому потрібно купити її, тим самим витратити внутрішньо ігрову валюту за основу взято валютою американський долар, але це не грає важливе значення, бо її значення дати гравцю розуміння, що позначає ця цифра лише одним символом.

На інтерфейсі який ми детальніше розглянемо далі розміщено 4 змінні на верхній панелі: з ліва поточна кількість грошей, у центрі минулий прибуток, з права 2 змінні через символ « / », з ліва поточна кількість населення, з права поточна максимальна кількість населення.

Рисунок 30. верхня панель інтерфейсу

Працює система грошей наступним чином у головному скрипті GameManager у методі Start, який запускається при запуску скрипту завантажує стандартні налаштування системи грошей та населення, оновлює тексти інтерфейсів, та запускається «карутина»– це метод, який виконує код через написану затримку. Приклад карутини з цього проекту:

```
private IEnumerator waiter()
    {
        while (true)
        {
            yield return new WaitForSecondsRealtime(7);
            UpdateMIP();
            UpdateTextMIP();
        }
    }
```

Весь код розташован у цикле while, тому код з затримкою виконується нескінченно. І якщо потрібно карутину завжди можна зупинити з іншого класу або методу.

Функція UpdateMIP запускає прорахування отриманого прибутку та збільшення населення. Працює це так, спочатку вираховується наскільки збільшиться населення чи зменшиться – заходиться 1% від показника поточної максимальної кількості населення, далі через оператор switch обирається діапазон випадкового зростання чи зменшення, Далі поточна кількість населення порівнюється з половиною поточній максимальній кількості

населення у результаті чого, вирішується тип зростання населення: в обидві сторони чи в одну сторону.

У першому випадку населення може, як збільшитися, так і зменшитись – це виправдовується динамічними змінами населення, як і в реальному світі.

У другому випадку населення може тільки збільшуватись – це водночас запобіжний засіб від першого варіанта, якщо населення дуже сильно зменшиться, так і якщо гравець збудує багато будівель – це буде нагадувати міста, що швидко розростаються, тим самим стають дуже привабливими для людей і тому на плив нових містян перебиває хоч-якесь зменшення населення. Приклад цієї системи у коді:

```
//POPULATION PART

m = (int) maxPopulation/100;
switch (m)
{
    case >= 10 :
        n = rnd.Next(0, 55);
        //Debug.Log("m>10. "+m+" n: "+n);
        break;
    case >= 5 when m < 10:
        n = rnd.Next(0, 16);
        //Debug.Log("10>m>5. "+m+" n: "+n);
        break;
    case < 5:
        n = rnd.Next(0, 5);
        //.Log("5>m. "+m+" n: "+n);
        break;
    default:
        Debug.Log("erorr in UpdateMIP");
        break;
}
m = (int) maxPopulation/2;
if (population>=(maxPopulation/2))
{
    v = rnd.Next(1, 11);
    if (v == 3 || v == 6 || v == 10)
    {
        population = population - n;
        //Debug.Log("Pop>Max -"+n+" income pop v:"+v);
    }
```

```
            if(v == 1 || v == 2 || v == 4 || v == 5 || v == 7 || v == 8 || v == 9
|| v == 11)
            {
                population = population + n;
                if (population>maxPopulation)
                {
                    population = maxPopulation;
                }
                //Debug.Log("Pop>Max +"+n+" income pop v:"+v);
            }
        }
        if(population<(maxPopulation/2))
        {
            population = population + n;
            if (population>maxPopulation)
            {
                population = maxPopulation;
            }
            //Debug.Log("Pop<Max: +"+n+" income pop v:0");
        }

        //MONEY PART

        a = (population * 100) / maxPopulation * maxIncome / 100;
        Debug.Log("a: "+a);
        income = (int) a;
        money = money + income;
```

Прибуток вже розраховується після розрахування населення, бо зміна кількості населення приводить до іншого прибутку. Прибуток розраховується так: поточна кількість населення помножується на 100, ділиться на максимальну кількість населення тим самим знаходиться 100 помножити на процент населення від максимуму, далі усе множиться на максимальний прибуток та ділиться на 100. 100 у цьому рівнянні лише задля точності розрахунків. Після чого float змінна перетворюється та записується на зміну income типу int. А вже прибуток додається до поточної кількості грошей, що утворює нову кількість грошей. У кінці запускається функція оновлення інтерфейсу верхньої панелі, після чого гравець може бачити зміни.

Коли у гравця достатньо коштів він може будувати спори. При спробі будувати першою перевіркою йде перевірка коштів, якщо вистачає будівництво йде як звичайно, а у кінці після усіх перевірок викликається

функція з головного скрипта, яка додає певну кількість максимального прибутку чи населення, також ціна споруди збільшується на 10% та запускається функція оновлення інтерфейсу гравця, за для того щоби гравець побачив зміни грошей, якщо операція пройшла успішно та інші зміни.

Якщо ж гравець почне зносити споруду він отримає 50% від старої ціни, а також зменшує ціну до минулої.

Хочу зауважити, що у кожного типу споруди свій цінник і свій додаток до показників.

3.2.8 Збереження

Система збережень цієї грі працює наступним чином використовуються 3 скрипта: перший – скрипт, який зчитує та записує данні у файл типу .json; другий – він містить шаблон збережених даних, по якому записуються данні з третього скрипта; третій – це головний скрипт GameManager, де у кінці є змінні для запису ігрових даних, чи зміні ігрових даних.

В цілому система збережень має 11 місць для збережень, а зберігає у файлах гри змінні системи грошей та населення, та усі споруди розміщенні на карті. Для цього перебирається масив даних споруд, де записано розташування та індекс. Індекс потрібен лише для будинки, дорогам не потрібне, бо будинки використовують систему будівництва з випадковим префабом будинка, тому якщо не записувати індекс, при завантаженні будівлі завжди будуть різними.

Збереження, завантаження та запуск нової гри визиваються кнопками у меню, саме меню розташовано зліва та зверху та виглядає як три паралельні лінії

Рисунок 31. Кнопка меню з зображенням меню.

Рисунок 32. Панель меню з кнопками збереження, завантаження та запуску нової гри

3.2.9 UI інтерфейс та його анімація

Інтерфейс – це відображення елементів гри спосіб взаємодії з ігровим світом. Тому він повинен бути зрозумілим, не великим, та не відволікати від гри. Тому інтерфейс у цієї гри виконано в мінімалізме.

Вин складається з 3 головних елементів меню, верхньої інформаційної панелі та вкладки будівництва

Спочатку скотарюємо «Canvas» - це буде нашою текою для елементів інтерфейсу, далі створюємо «Canvas» у ньому, та називаємо «BG» - це за для того щоби бачити кордони нашого інтерфейсу. Головний Canvas налаштовуємо наступним чином, як на рис. 33

Після чого створюємо «Panel», називаємо «MIPPanel», ставимо прозорість на 0% у color, а вирівнювання по центру та в середині, далі розташовуємо x: 13.70651; y: 207; z: 0; а розмір: ширина: 772.589; висота: 35. Далі у панелі MIPPanel створюємо 3 елементи типу Text, називаємо їх «TextMoney», «TextIncome» та «TextPopulation», в розділі Text усюди пишімо «SCORE», а шрифт обираємо Aria, розмір шрифту 20, прибираємо прозорість та красимо усі окрім TextIncome у чорний, а TextIncome у зелений.

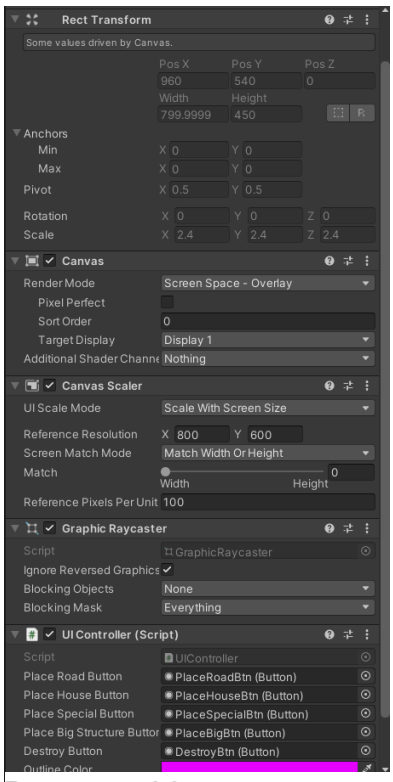

Рисунок 33.
Налаштування Canvas

Далі додаємо MIPPanel компонент «HorizontalLayout Group», у ньому ставимо усі галочки навпроти окрім налаштування «Reverse Arrangement». Результат на рис. 20. Тепер щоби текст змінних почав працювати потрібно перетягнути ці три елемента текст з ієрархії до 3 відповідних слотів в інспекторі GameManager.

Зараз переходимо до меню, створюємо кнопку, видаляємо усередині текст, розміщуємо елемент image, для нього перетягуємо іконку у слот «source», в інспекторі обираємо вирівнювання по центру, ширину та висоту робимо 20. Далі повертаємося до кнопки вирівнюємо також як і зображення, висоту та ширину робимо 35, розташовуємо по координатах x: -381.899; y: 207; z: 0. Додаємо функцію OnClick натискаючи плюс в інспекторі, перетягуємо з ієрархії GameManager у слот об’єкта, та обираємо функцію «SwitchVisabilityMenu». Результат кнопки меню на рис. 31.

Далі створюємо у Canvas «Panel», називаємо його «SaveSystemPanel», вирівнюємо, та задаємо ширину: 175; та висоту: 101.5606; розташовуємо по координатах x: -87.56 y: 50.67; z: 0;, прозорість: 40%, додаємо компонент «VerticalLayout Group».

У ньому створюємо 3 кнопки та називаємо «NewGameBtn», «SaveBtn» та «LoadBtn», робимо вирівнювання, ширину: 160; висоту: 30;, усередині змінюємо елемент Text та змінюймо текст на «New Game», «Save» та «Load». Результат можна поглянути на рис.32.

Строюємо панель та називаємо «Build Group», робимо прозорість 100%, вирівнюємо, ширину: 175; висоту: 202.8;, розташування x: -312.5y: -311.3; z: 0;, на нього добавляємо компонент «Animation».

Всередині Build Group створюємо кнопку та панель. Кнопку називаймо «BuildMoveBtn», вирівнюємо, ширину: 25; висоту: 25;, розташування x: -75; y: 91.6; z: 0;, добавляймо функцію OnClick, перетягуємо об’єкт на слот файлу «CloseOpenBuildPanel». Всередині кнопки створюємо зображення та перетягуємо «tools» з провідника на «source» у інспекторі.

Панель називаймо «BuildMoveBtn», вирівнюємо, ширину: 175; висоту: 188.4968;, розташування x: 6.10; y: -7.62;; z: 0; додаємо компонент «VerticalLayout Group». Всередині створюємо 1 елемент тексту та 5 кнопок. У тексті в тексті пишімо «Building Menu:», ширина: 160; висота: 30. Кнопки усі з шириною: 160; висотою: 30;, та наступними назвами «PlaceRoadBtn», «PlaceHouseBtn», «PlaceSpecialBtn», «PlaceBigBtn» та «DestroyBtn», й відповідно з наступним текстом «Road», «House», «Special», «Place 2x2» та «Destroy».

Після чого усі ці кнопки потрібно додати у компоненті UI скрипт, який ми додали до головного Canvas, а колір обрати «E401FF». Результат можна побачити на рис. 33 та 34.

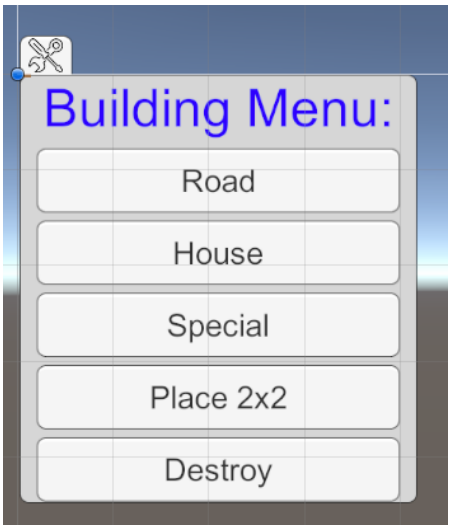

Рисунок 34. Створена панель будівництва

І останнє, що залишилося зробити, так це анімацію панелі будівництва. Вона буде вискакувати при натисканні кнопки з рисунком інструментів. Відкриваємо вкладку «Animation», тиснемо знизу «Create», зберігаємо у потрібно місці анімацію. Після чого обираємо, що ми будем змінювати – в нашому випадку це позиція та натискаємо червону кнопку запису, тягнемо часову шкалу до 0:30 там робимо наступні зміни x: -312.5; y: -124.4; z: 0, після чого тиснемо на червону кнопку, цю анімацію називаємо BuildGroupOpen.

Тепер створюємо теж саму анімацію але навпаки й називаємо його BuildGroupClose. Реалізація у коді виклику відкриття та закриття панелі будівництва:

```
switch (IndexBuildPan)
{
    case 1:
        anim.Play(“BuildGroupOpen”); //-311.3 -124.4
        IndexBuildPan = 2;
        break;
    case 2:
        anim.Play(“BuildGroupClose”);
        IndexBuildPan = 1;
        break;
    default:
        IndexBuildPan = 1;
        CloseOpenBuildPan();
        break;
}
```

Функція anim.Play(“BuildGroupClose”) в якої .play відповідає за запуск анімації, а значення у дужках вирішує, яку саме анімацію запустить на ігровому об’єкті anim, який було прив’язано до скрипта через додавання елементу на якому є компонент Animation

3.2.10 Звукові ефекти

Звукові ефекти працюють за посередництвом 2 об’єктів ієрархії: об’єкта який відтворює звук, там об’єкта який виступає слухачем. В даному випадку слухачем виступає камера на якій розташован AudioListener, А відтворювачем

звуку є об’єкт «AudioPlayer», на якому розташован компонент «Audio Source» та скрип «AudioPlayer», який викликає розміщений у його слоті звук будівництва. Приклад скрипта:

```
public class AudioPlayer : MonoBehaviour
    {
        public AudioClip placementSound;
        public AudioSource audioSource;

        public static AudioPlayer instance;

        private void Awake()
        {
            if (instance == null)
                instance = this;
            else if (instance != this)
                Destroy(this.gameObject);

        }

        public void PlayPlacementSound()
        {
            if(placementSound != null)
            {
                audioSource.PlayOneShot(placementSound);
            }
        }
        public void ChangeMute()
        {
            StartCoroutine(waiter());
        }

        private IEnumerator waiter()
        {
            audioSource.mute = !audioSource.mute;
            //Debug.Log("mute");
            yield return new WaitForSecondsRealtime(2);
            audioSource.mute = !audioSource.mute;
            //Debug.Log("Unmute");
        }
    }
```

У цьому коді функція PlayPlacementSound відповідає за виклик звукового ефекту, а функція ChangeMute виправляє звуковий баг при завантаженні ігрового прогресу шляхом приглушенім звуку цього ефекту.

За такою ж аналогією зроблені звуки знесення, прибутку, та цикл з музики.

ВИСНОВКИ

В результаті виконання даного дипломного проекту був створений ігровий додаток з жанру містобудівельна. Додаток призначений для функціонування під управлінням операційної системи Windows.

В ході розробки містобудівельного ігрового додатку до операційної системи Windows були вирішені такі задачі:

- Ознайомився з сучасними рушіями для створення ігрових додатків.
- Ознайомився з середою розробки ігрового рушія Unity, та отримав практичні навички роботи з ним.
- Було розроблене та реалізоване зручне, інтуїтивно зрозумілий інтерфейс ігрового додатку.
- Реалізовані складні механікі для ускладнення та зацікавлення гравця
- Був розроблений містобудівельний ігровий додаток
- Було реалізовано проект для операційної системи Windows.

ПЕРЕЛІК ПОСИЛАНЬ

1. Офіційна документація C#: https://docs.microsoft.com/en-us/dotnet/csharp/
2. Офіційна документація Unity: https://docs.unity3d.com/Manual/index.html
3. Unity знання: https://learn.unity.com/
4. Unity підручник на YouTube: https://www.youtube.com/channel/UCYbK_tjZ2OrIZFBvU6CCMiA
5. Джапіксе, Ф.(2017). Pro C# 7: With .NET and .NET Core / Ендрю Троєлсен – 1437 с.
6. Greene J.(2013). "Head First C#: A Learner's Guide to Real-World Programming with C#, XAML, and .NET" / Andrew Stellman – 1100 с.
7. Albahari J (2017) "C# 7.0 in a Nutshell: The Definitive Reference" / B. Albahari – 1088 с.
8. Hocking, J.(2015). "Unity in Action: Multiplatform Game Development in C#" – 352 с.
9. Ferrone, H. (2020). "Learning C# by Developing Games with Unity",(7) - 460 с.
10. Сайт з безкоштовним доступом до музики: https://artlist.io
11. Веб-ресурс іконок й спрайтів: flaticon.com
12. Веб-ресур з доступом до звукових ефектів: zvukipro.com
13. Веб-ресур з доступом до звукових ефектів: pixabay.com
14. Стаття користувача amitp про «Сітки та вузли»: https://www.redblobgames.com/pathfinding/grids/graphs.html

ДОДАТКИ

Додаток А1. GameManager

(a1 a2 a3 – каждый файл так)

```
using SVS;
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using Random = System.Random;

public class GameManager : MonoBehaviour
{
    [Header("FileImport")]
    public CameraMovement cameraMovement;
    public RoadManager roadManager;
    public InputManager inputManager;
    public UIController uiController;
    public StructureManager structureManager;
    public ObjectDetector objectDetector;
    public SaveSystem saveSystem;
    //public AnimationsManager animManager;

    [Space]
    [Header("Money System")]
    [SerializeField]public int money;
    [SerializeField]public int income;
    [SerializeField]public int maxIncome;
    [SerializeField]public int OneStructureIncome;
    [SerializeField]public int costSpecial;

    [SerializeField]public int population;
    [SerializeField]public int maxPopulation;
    [SerializeField]public int OneStructurePopulation;  // +employers
    [SerializeField]public int costHouse;

    [Space]
    //нуж текст работников и мб макс работников + рамдом.
    public Text textMoney;
    public Text textIncome;
    public Text textPopulation;

    private float a;
    private float b;
    private int IndexBuildPan = 1;
    private int m;
    private int v;
    private int n;
    private Random rnd = new Random();
```

```
[Space]
[Header("Animation")]
public Animation anim;
public GameObject saveSystemPanel;

void Start()
{
    uiController.OnRoadPlacement += RoadPlacementHandler;
    uiController.OnHousePlacement += HousePlacementHandler;
    uiController.OnSpecialPlacement += SpecialPlacementHandler;
    uiController.OnBigStructurePlacement += BigStructurePlacement;
    uiController.OnDestroy += DestroyHandler;
    inputManager.OnEscape += HandleEscape;
    textMoney.text = "";
    BeginingMoneySave();
    UpdateTextMIP();
    StartCoroutine(waiter());
}

private IEnumerator waiter()
{
    while (true)
    {
        yield return new WaitForSecondsRealtime(10);
        //UpdateMIP();
        UpdateTextMIP();
    }
}
//------------------------------MONEY----------
private void UpdateTextMIP()
{
    textMoney.text = money + "$";
    textIncome.text = income + "$";
    textPopulation.text = population+" / "+maxPopulation;
    return;
}

private void UpdateMIP()
{
    //POPULATION PART  *0.7 ne /2

    m = (int) maxPopulation/100;
    switch (m)
    {
        case >= 10 :
            n = rnd.Next(0, 55);
            //Debug.Log("m>10. "+m+" n: "+n);
            break;
        case >= 5 when m < 10:
```

```
                n = rnd.Next(0, 16);
                //Debug.Log("10>m>5. "+m+" n: "+n);
                break;
            case < 5:
                n = rnd.Next(0, 5);
                //.Log("5>m. "+m+" n: "+n);
                break;
            default:
                Debug.Log("erorr in UpdateMIP");
                break;
        }
        m = (int) maxPopulation/2;
        if (population>=(maxPopulation/2))
        {
            v = rnd.Next(1, 11);
            if (v == 3 || v == 6 || v == 10)
            {
                population = population - n;
                //Debug.Log("Pop>Max -"+n+" income pop v:"+v);
            }
            if(v == 1 || v == 2 || v == 4 || v == 5 || v == 7 || v == 8 || v == 9
|| v == 11)
            {
                population = population + n;
                if (population>maxPopulation)
                {
                    population = maxPopulation;
                }
                //Debug.Log("Pop>Max +"+n+" income pop v:"+v);
            }
        }
        if(population<(maxPopulation/2))
        {
            population = population + n;
            if (population>maxPopulation)
            {
                population = maxPopulation;
            }
            //Debug.Log("Pop<Max: +"+n+" income pop v:0");
        }

        a = (population * 100) / maxPopulation * maxIncome / 100;
        Debug.Log("a: "+a);
        income = (int) a;
        money = money + income;

        UpdateTextMIP();
    }

    public void RemoveOneBuildingOfPopulation()
```

```
{
    a = (costHouse * 0.9f * 0.5f) + money;
    money = (int) a;
    maxPopulation = maxPopulation - OneStructurePopulation;
    b = costHouse * 0.9f;
    costHouse = (int) b;
    Debug.Log("MaxPop reduction. Money+50%cost="+a);
    UpdateTextMIP();
}

public void RemoveOneBuildingOfIncome()
{
    a = (costSpecial * 0.9f * 0.5f) + money;
    money = (int) a;
    maxIncome = maxIncome - OneStructureIncome;
    b = costSpecial * 0.9f;
    costSpecial = (int) b;
    Debug.Log("MaxInc  reduction. Money+50%cost="+a);
    UpdateTextMIP();
}

public void AddOneBuildingOfPopulation()
{
    money = money - costHouse;
    maxPopulation = maxPopulation + OneStructurePopulation;
    b = costHouse * 1.1f;
    costHouse = (int) b;
    Debug.Log("MaxPop promotion");
    UpdateTextMIP();
}

public void AddOneBuildingOfIncome()
{
    money = money - costSpecial;
    maxIncome = maxIncome + OneStructureIncome;
    b = costSpecial * 1.1f;
    costSpecial = (int) b;
    Debug.Log("MaxInc promotion");
    UpdateTextMIP();
}


private int bMoney;
private int bIncome;
private int bMaxIncome;
private int bOneStructureIncome;
private int bCostSpecial;
private int bPopulation;
private int bMaxPopulation;
private int bOneStructurePopulation;
```

```
private int bCostHouse;
// save beg. settings
private void BeginingMoneySave()
{
    bMoney = money;
    bIncome = income;
    bMaxIncome = maxIncome;
    bOneStructureIncome = OneStructureIncome;
    bCostSpecial = costSpecial;
    bPopulation = population;
    bMaxPopulation = maxPopulation;
    bOneStructurePopulation = OneStructurePopulation;
    bCostHouse = costHouse;
}
// load beg. settings
public void BeginingMoneyLoad()
{
    money = bMoney;
    income = bIncome;
    maxIncome = bMaxIncome;
    OneStructureIncome = bOneStructureIncome;
    costSpecial = bCostSpecial;
    population = bPopulation;
    maxPopulation = bMaxPopulation;
    OneStructurePopulation = bOneStructurePopulation;
    costHouse = bCostHouse;
    UpdateTextMIP();
}

//--------------------------------- BUTTONS-------------

private void HandleEscape()
{
    ClearInputActions();
    uiController.ResetButtonColor();
}


private void DestroyHandler()
{
    ClearInputActions();

    inputManager.OnMouseClick += (pos) =>
    {
        ProcessInputAndCall(structureManager.Destroy, pos);
    };
    inputManager.OnEscape += HandleEscape;
}
```

```
private void BigStructurePlacement()
{
    ClearInputActions();

    inputManager.OnMouseClick += (pos) =>
    {
        ProcessInputAndCall(structureManager.PlaceBigStructure, pos);
    };
    inputManager.OnEscape += HandleEscape;
}

private void SpecialPlacementHandler()
{
    ClearInputActions();

    inputManager.OnMouseClick += (pos) =>
    {
        if (money >= costSpecial)
        {
            ProcessInputAndCall(structureManager.PlaceSpecial, pos);
        }
    };
    inputManager.OnEscape += HandleEscape;
}

private void HousePlacementHandler()
{
    ClearInputActions();

    inputManager.OnMouseClick += (pos) =>
    {
        if (money >= costHouse)
        {
            ProcessInputAndCall(structureManager.PlaceHouse, pos);
        }
    };
    inputManager.OnEscape += HandleEscape;
}

private void RoadPlacementHandler()
{
    ClearInputActions();

    inputManager.OnMouseClick += (pos) =>
    {
        ProcessInputAndCall(roadManager.PlaceRoad, pos);
    };
    inputManager.OnMouseUp += roadManager.FinishPlacingRoad;
    inputManager.OnMouseHold += (pos) =>
    {
```

```
        ProcessInputAndCall(roadManager.PlaceRoad, pos);
    };
    inputManager.OnEscape += HandleEscape;
}

private void ClearInputActions()
{
    inputManager.ClearEvents();
}

private void ProcessInputAndCall(Action<Vector3Int> callback, Ray ray)
{
    Vector3Int? result = objectDetector.RaycastGround(ray);
    if (result.HasValue)
        callback.Invoke(result.Value);
}

//--------------------- ANIMATIONS -------------------------

public void CloseOpenBuildPan()
{
    Debug.Log("index:" + IndexBuildPan);
    switch (IndexBuildPan)
    {
        case 1:
            anim.Play("BuildGroupOpen"); //-311.3 -124.4
            IndexBuildPan = 2;
            break;
        case 2:
            anim.Play("BuildGroupClose");
            IndexBuildPan = 1;
            break;
        default:
            IndexBuildPan = 1;
            CloseOpenBuildPan();
            break;
    }


}

public void SwitcchVisiabilityMenu()
{
    saveSystemPanel.SetActive(!saveSystemPanel.activeSelf);
}


private void Update()
{
```

```
        cameraMovement.MoveCamera(new
Vector3(inputManager.CameraMovementVector.x, 0,
inputManager.CameraMovementVector.y));
    }

    public void SaveGame()
    {
        SaveDataSerialization saveData = new SaveDataSerialization();
        foreach (var structureData in structureManager.GetAllStructures())
        {
            saveData.AddStructureData(structureData.Key,
structureData.Value.BuildingPrefabIndex, structureData.Value.BuildingType);
        }
        saveData.money = money;
        saveData.income = income;
        saveData.maxIncome = maxIncome;
        saveData.OneStructureIncome = OneStructureIncome;
        saveData.costSpecial = costSpecial;
        saveData.population = population;
        saveData.maxPopulation = maxPopulation;
        saveData.OneStructurePopulation = OneStructurePopulation;
        saveData.costHouse = costHouse;
        var jsonFormat = JsonUtility.ToJson(saveData);
        Debug.Log(jsonFormat);
        saveSystem.SaveData(jsonFormat);
    }

    public void LoadGame()
    {
        var jsonFormatData = saveSystem.LoadData();
        if (String.IsNullOrEmpty(jsonFormatData))
            return;
        SaveDataSerialization saveData =
JsonUtility.FromJson<SaveDataSerialization>(jsonFormatData);
        structureManager.ClearMap();
        foreach (var structureData in saveData.structuresData)
        {
            Vector3Int position =
Vector3Int.RoundToInt(structureData.position.GetValue());
            if (structureData.buildingType == CellType.Road)
            {
                roadManager.PlaceRoad(position);
                roadManager.FinishPlacingRoad();
            }
            else
            {
                structureManager.PlaceLoadedStructure(position,
structureData.buildingPrefabindex, structureData.buildingType);
            }
        }
```

```
        money = saveData.money;
        income = saveData.income;
        maxIncome = saveData.maxIncome;
        OneStructureIncome = saveData.OneStructureIncome;
        costSpecial = saveData.costSpecial;
        population = saveData.population;
        maxPopulation = saveData.maxPopulation;
        OneStructurePopulation = saveData.OneStructurePopulation;
        costHouse = saveData.costHouse;
        UpdateTextMIP();
    }
}
```

Додаток А3. Grid

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

namespace SVS
{

    public class CameraMovement : MonoBehaviour
    {
        public Camera gameCamera;
        public float cameraMovementSpeed = 5;

        public float maxOrthographicSize = 5f, minOrthographicSize = 0.5f;
        public float sensitivity = 0.1f;

        private void Start()
        {
            gameCamera = GetComponent<Camera>();
        }
        public void MoveCamera(Vector3 inputVector)
        {
            var movementVector = Quaternion.Euler(0,30,0) * inputVector;
            gameCamera.transform.position += movementVector * Time.deltaTime *
cameraMovementSpeed;
        }

        private void Update()
        {
            var scrollInput = Input.GetAxis("Mouse ScrollWheel") * sensitivity;
            gameCamera.orthographicSize = Mathf.Clamp(gameCamera.orthographicSize
- scrollInput, minOrthographicSize, maxOrthographicSize);
        }
    }
}
```

Додаток А3.

```
using System;
using System.Collections.Generic;
public class Point
{
    public int X { get; set; }
    public int Y { get; set; }

    public Point(int x, int y)
    {
        this.X = x;
        this.Y = y;
    }

    public override bool Equals(object obj)
    {
        if (obj == null)
        {
            return false;
        }
        if (obj is Point)
        {
            Point p = obj as Point;
            return this.X == p.X && this.Y == p.Y;
        }
        return false;
    }

    public override int GetHashCode()
    {
        unchecked
        {
            int hash = 6949;
            hash = hash * 7907 + X.GetHashCode();
            hash = hash * 7907 + Y.GetHashCode();
            return hash;
        }
    }

    public override string ToString()
    {
        return "P(" + this.X + ", " + this.Y + ")";
    }
}

public enum CellType
{
```

```
    Empty,
    Road,
    Structure,
    BigStructure,
    SpecialStructure,
    None
}

public class Grid
{
    private CellType[,] _grid;
    private int _width;
    public int Width { get { return _width; } }
    private int _height;
    public int Height { get { return _height; } }

    private List<Point> _roadList = new List<Point>();
    private List<Point> _specialStructure = new List<Point>();
    private List<Point> _houseStructure = new List<Point>();

    public Grid(int width, int height)
    {
        _width = width;
        _height = height;
        _grid = new CellType[width, height];
    }

    // Adding index operator to our Grid class so that we can use grid[][] to
access specific cell from our grid.
    public CellType this[int i, int j]
    {
        get
        {
            return _grid[i, j];
        }
        set
        {
            if (value == CellType.Road)
            {
                _roadList.Add(new Point(i, j));
            }
            if (value == CellType.SpecialStructure)
            {
                _specialStructure.Add(new Point(i, j));
            }
            if (value == CellType.Structure || value == CellType.BigStructure)
            {
                _houseStructure.Add(new Point(i, j));
            }
            _grid[i, j] = value;
```

```
        }
    }

    public static bool IsCellWakable(CellType cellType, bool aiAgent = false)
    {
        if (aiAgent)
        {
            return cellType == CellType.Road;
        }
        return cellType == CellType.Empty || cellType == CellType.Road;
    }

    public Point GetRandomRoadPoint()
    {
        if (_roadList.Count == 0)
        {
            return null;
        }
        return _roadList[UnityEngine.Random.Range(0, _roadList.Count)];
    }

    public Point GetRandomSpecialStructurePoint()
    {
        if (_specialStructure.Count == 0)
        {
            return null;
        }
        return _specialStructure[UnityEngine.Random.Range(0,
_specialStructure.Count)];
    }

    public Point GetRandomHouseStructurePoint()
    {
        if (_houseStructure.Count == 0)
        {
            return null;
        }
        return _houseStructure[UnityEngine.Random.Range(0,
_houseStructure.Count)];
    }

    public List<Point> GetAllHouses()
    {
        return _houseStructure;
    }

    internal List<Point> GetAllSpecialStructure()
    {
        return _specialStructure;
    }
```

```
    public List<Point> GetAdjacentCells(Point cell, bool isAgent)
    {
        return GetWakableAdjacentCells((int)cell.X, (int)cell.Y, isAgent);
    }

    public float GetCostOfEnteringCell(Point cell)
    {
        return 1;
    }

    public List<Point> GetAllAdjacentCells(int x, int y)
    {
        List<Point> adjacentCells = new List<Point>();
        if (x > 0)
        {
            adjacentCells.Add(new Point(x - 1, y));
        }
        if (x < _width - 1)
        {
            adjacentCells.Add(new Point(x + 1, y));
        }
        if (y > 0)
        {
            adjacentCells.Add(new Point(x, y - 1));
        }
        if (y < _height - 1)
        {
            adjacentCells.Add(new Point(x, y + 1));
        }
        return adjacentCells;
    }

    public List<Point> GetWakableAdjacentCells(int x, int y, bool isAgent)
    {
        List<Point> adjacentCells = GetAllAdjacentCells(x, y);
        for (int i = adjacentCells.Count - 1; i >= 0; i--)
        {
            if (IsCellWakable(_grid[adjacentCells[i].X, adjacentCells[i].Y],
isAgent) == false)
            {
                adjacentCells.RemoveAt(i);
            }
        }
        return adjacentCells;
    }

    public List<Point> GetAdjacentCellsOfType(int x, int y, CellType type)
    {
        List<Point> adjacentCells = GetAllAdjacentCells(x, y);
```

```
        for (int i = adjacentCells.Count - 1; i >= 0; i--)
        {
            if (_grid[adjacentCells[i].X, adjacentCells[i].Y] != type)
            {
                adjacentCells.RemoveAt(i);
            }
        }
        return adjacentCells;
    }

    /// <summary>
    /// Returns array [Left neighbour, Top neighbour, Right neighbour, Down
neighbour]
    /// </summary>
    /// <param name="x"></param>
    /// <param name="y"></param>
    /// <returns></returns>
    public CellType[] GetAllAdjacentCellTypes(int x, int y)
    {
        CellType[] neighbours = { CellType.None, CellType.None, CellType.None,
CellType.None };
        if (x > 0)
        {
            neighbours[0] = _grid[x - 1, y];
        }
        if (x < _width - 1)
        {
            neighbours[2] = _grid[x + 1, y];
        }
        if (y > 0)
        {
            neighbours[3] = _grid[x, y - 1];
        }
        if (y < _height - 1)
        {
            neighbours[1] = _grid[x, y + 1];
        }
        return neighbours;
    }
}
```

Додаток А4. Grid Search

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

public class GridSearch {
```

```
    public struct SearchResult
    {
        public List<Point> Path { get; set; }
    }

    public static List<Point> AStarSearch(Grid grid, Point startPosition, Point
endPosition, bool isAgent = false)
    {
        List<Point> path = new List<Point>();

        List<Point> positionsTocheck = new List<Point>();
        Dictionary<Point, float> costDictionary = new Dictionary<Point, float>();
        Dictionary<Point, float> priorityDictionary = new Dictionary<Point,
float>();
        Dictionary<Point, Point> parentsDictionary = new Dictionary<Point,
Point>();

        positionsTocheck.Add(startPosition);
        priorityDictionary.Add(startPosition, 0);
        costDictionary.Add(startPosition, 0);
        parentsDictionary.Add(startPosition, null);

        while (positionsTocheck.Count > 0)
        {
            Point current = GetClosestVertex(positionsTocheck,
priorityDictionary);
            positionsTocheck.Remove(current);
            if (current.Equals(endPosition))
            {
                path = GeneratePath(parentsDictionary, current);
                return path;
            }

            foreach (Point neighbour in grid.GetAdjacentCells(current, isAgent))
            {
                float newCost = costDictionary[current] +
grid.GetCostOfEnteringCell(neighbour);
                if (!costDictionary.ContainsKey(neighbour) || newCost <
costDictionary[neighbour])
                {
                    costDictionary[neighbour] = newCost;

                    float priority = newCost + ManhattanDiscance(endPosition,
neighbour);
                    positionsTocheck.Add(neighbour);
                    priorityDictionary[neighbour] = priority;

                    parentsDictionary[neighbour] = current;
                }
            }
```

```
        }
        return path;
    }

    private static Point GetClosestVertex(List<Point> list, Dictionary<Point,
float> distanceMap)
    {
        Point candidate = list[0];
        foreach (Point vertex in list)
        {
            if (distanceMap[vertex] < distanceMap[candidate])
            {
                candidate = vertex;
            }
        }
        return candidate;
    }

    private static float ManhattanDiscance(Point endPos, Point point)
    {
        return Math.Abs(endPos.X - point.X) + Math.Abs(endPos.Y - point.Y);
    }

    public static List<Point> GeneratePath(Dictionary<Point, Point> parentMap,
Point endState)
    {
        List<Point> path = new List<Point>();
        Point parent = endState;
        while (parent != null && parentMap.ContainsKey(parent))
        {
            path.Add(parent);
            parent = parentMap[parent];
        }
        return path;
    }
}
```

Додаток А5 INeedingRoad

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

public interface INeedingRoad
{
    Vector3Int RoadPosition { get; set; }
}
```

Додаток А6 Input Manager

```
using System;
using System.Collections;
```

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

public class InputManager : MonoBehaviour
{
    public event Action<Ray> OnMouseClick, OnMouseHold;
    public event Action OnMouseUp, OnEscape;
    private Vector2 mouseMovementVector = Vector2.zero;
    public Vector2 CameraMovementVector { get => mouseMovementVector; }
    [SerializeField]
    Camera mainCamera;


    void Update()
    {
        CheckClickDownEvent();
        CheckClickHoldEvent();
        CheckClickUpEvent();
        CheckArrowInput();
        CheckEscClick();
    }

    private void CheckClickHoldEvent()
    {
        if (Input.GetMouseButton(0) &&
EventSystem.current.IsPointerOverGameObject() == false)
        {

            OnMouseClick?.Invoke(mainCamera.ScreenPointToRay(Input.mousePosition)
);
        }
    }

    private void CheckClickUpEvent()
    {
        if (Input.GetMouseButtonUp(0) &&
EventSystem.current.IsPointerOverGameObject() == false)
        {
            OnMouseUp?.Invoke();
        }
    }

    private void CheckClickDownEvent()
    {
        if (Input.GetMouseButtonDown(0) &&
EventSystem.current.IsPointerOverGameObject() == false)
        {
            OnMouseClick?.Invoke(mainCamera.ScreenPointToRay(Input.mousePosition)
);
```

```
        }
    }

    private void CheckEscClick()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            OnEscape.Invoke();
        }
    }

    private void CheckArrowInput()
    {
        mouseMovementVector = new Vector2(Input.GetAxis("Horizontal"),
Input.GetAxis("Vertical"));
    }

    public void ClearEvents()
    {
        OnMouseClick = null;
        OnMouseHold = null;
        OnEscape = null;
        OnMouseUp = null;
    }
}
```

Додаток А7 Object Detector

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

public class ObjectDetector : MonoBehaviour
{
    public LayerMask groundMask;

    public Vector3Int? RaycastGround(Ray ray)
    {
        RaycastHit hit;
        if (Physics.Raycast(ray, out hit, Mathf.Infinity, groundMask))
        {
            Transform objectHit = hit.transform;
            Vector3Int positionInt = Vector3Int.RoundToInt(hit.point);
            return positionInt;
        }
        return null;
    }

    public GameObject RaycastAll(Ray ray)
    {
        RaycastHit hit;
```

```
        if (Physics.Raycast(ray, out hit, Mathf.Infinity))
        {
            return hit.transform.gameObject;
        }
        return null;
    }
}
```

Додаток А8 Placement Manager

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

public class PlacementManager : MonoBehaviour
{
    public GameManager gameManager;

    [Space]
    public int width, height;
    Grid placementGrid;

    private Dictionary<Vector3Int, StructureModel> temporaryRoadobjects = new
Dictionary<Vector3Int, StructureModel>();
    private Dictionary<Vector3Int, StructureModel> structureDictionary = new
Dictionary<Vector3Int, StructureModel>();

    private void Start()
    {
        placementGrid = new Grid(width, height);
    }

    internal CellType[] GetNeighbourTypesFor(Vector3Int position)
    {
        return placementGrid.GetAllAdjacentCellTypes(position.x, position.z);
    }

    internal bool CheckIfPositionInBound(Vector3Int position)
    {
        if (position.x >= 0 && position.x < width && position.z >= 0 &&
position.z < height)
        {
            return true;
        }
        return false;
    }

    internal void PlaceObjectOnTheMap(Vector3Int position, GameObject
structurePrefab, CellType type, int width = 1, int height = 1, int
buildingPrefabIndex = -1)
```

```
    {
        StructureModel structure = CreateANewStructureModel(position,
structurePrefab, type, buildingPrefabIndex);

        var structureNeedingRoad = structure.GetComponent<INeedingRoad>();
        if (structureNeedingRoad != null)
        {
            structureNeedingRoad.RoadPosition = GetNearestRoad(position, width,
height).Value;
            Debug.Log("My nearest road position is: " +
structureNeedingRoad.RoadPosition);
        }

        structureDictionary.Add(position, structure);
        for (int x = 0; x < width; x++)
        {
            for (int z = 0; z < height; z++)
            {
                var newPosition = position + new Vector3Int(x, 0, z);
                placementGrid[newPosition.x, newPosition.z] = type;

                DestroyNatureAt(newPosition);
            }
        }


    }

    private Vector3Int? GetNearestRoad(Vector3Int position, int width, int
height)
    {
        for (int x = 0; x < width; x++)
        {
            for (int y = 0; y < height; y++)
            {
                var newPosition = position + new Vector3Int(x, 0, y);
                var roads = GetNeighboursOfTypeFor(newPosition, CellType.Road);
                if (roads.Count > 0)
                {
                    return roads[0];
                }
            }
        }
        return null;
    }

    private void DestroyNatureAt(Vector3Int position)
    {
```

```
        RaycastHit[] hits = Physics.BoxCastAll(position + new Vector3(0, 0.5f,
0), new Vector3(0.5f, 0.5f, 0.5f), transform.up, Quaternion.identity, 1f, 1 <<
LayerMask.NameToLayer("Nature"));
        foreach (var item in hits)
        {
            Destroy(item.collider.gameObject);
        }
    }


    public void DestroyBuildingAt(Vector3Int position)
    {
        //RaycastHit[] hits = Physics.BoxCastAll(position + new Vector3(0, 0.5f,
0), new Vector3(0.3f, 0.5f, 0.3f), transform.up, Quaternion.identity, 1f, 1 <<
LayerMask.NameToLayer("Default"));
        RaycastHit[] hits = Physics.BoxCastAll(position + new Vector3(0, 0.5f,
0), new Vector3(0.2f, 0, 0.2f), transform.up, Quaternion.identity, 0.5f, 1 <<
LayerMask.NameToLayer("Default"));

        foreach (var item in hits)
        {
            var a = item.collider.gameObject;
            GameObject foo;
            foo = a.transform.parent.gameObject;
            if (a.transform.parent.name == "Structure" || a.transform.parent.name
== "Road" || a.transform.parent.name == "SpecialStructure")
            {
                switch (a.transform.parent.name)
                {
                case "Structure":
                    gameManager.RemoveOneBuildingOfPopulation();
                    break;
                case "SpecialStructure":
                    gameManager.RemoveOneBuildingOfIncome();
                    break;
                default:
                    break;
                }
                placementGrid[position.x, position.z] = CellType.Empty;
                Destroy(foo);
            }
            Debug.Log("+Name Parent: " + item.transform.parent.name);
            if (a.transform.parent.parent.name == "Road")
            {

                placementGrid[position.x, position.z] = CellType.Empty;
                foo = a.transform.parent.parent.gameObject;
                Destroy(foo);
            }
```

```
            Debug.Log("+Name PParent: " +
item.transform.parent.parent.name);
        }
        foreach (var structure in GetAllStructures())
        {
            if (structure.Key == position)
            {
                if(structureDictionary.ContainsKey(position))
                {
                    Debug.Log(position+" - "+ structure.Key);
                    structureDictionary.Remove(position);
                }
            }
        }
    }


    internal bool CheckIfPositionIsFree(Vector3Int position)
    {
        return CheckIfPositionIsOfType(position, CellType.Empty);
    }

    private bool CheckIfPositionIsOfType(Vector3Int position, CellType type)
    {
        return placementGrid[position.x, position.z] == type;
    }

    internal void PlaceTemporaryStructure(Vector3Int position, GameObject
structurePrefab, CellType type, int buildingPrefabindex = -1)
    {
        placementGrid[position.x, position.z] = type;
        StructureModel structure = CreateANewStructureModel(position,
structurePrefab, type, buildingPrefabindex);
        temporaryRoadobjects.Add(position, structure);
    }

    internal List<Vector3Int> GetNeighboursOfTypeFor(Vector3Int position,
CellType type)
    {
        var neighbourVertices = placementGrid.GetAdjacentCellsOfType(position.x,
position.z, type);
        List<Vector3Int> neighbours = new List<Vector3Int>();
        foreach (var point in neighbourVertices)
        {
            neighbours.Add(new Vector3Int(point.X, 0, point.Y));
        }
        return neighbours;
    }
```

```
    private StructureModel CreateANewStructureModel(Vector3Int position,
GameObject structurePrefab, CellType type, int buildingPrefabIndex)
    {
        GameObject structure = new GameObject(type.ToString());
        structure.transform.SetParent(transform);
        structure.transform.localPosition = position;
        var structureModel = structure.AddComponent<StructureModel>();
        structureModel.CreateModel(structurePrefab, buildingPrefabIndex, type);
        return structureModel;
    }

    internal List<Vector3Int> GetPathBetween(Vector3Int startPosition, Vector3Int
endPosition)
    {
        var resultPath = GridSearch.AStarSearch(placementGrid, new
Point(startPosition.x, startPosition.z), new Point(endPosition.x,
endPosition.z));
        List<Vector3Int> path = new List<Vector3Int>();
        foreach (Point point in resultPath)
        {
            path.Add(new Vector3Int(point.X, 0, point.Y));
        }
        return path;
    }

    internal void RemoveAllTemporaryStructures()
    {
        foreach (var structure in temporaryRoadobjects.Values)
        {
            var position = Vector3Int.RoundToInt(structure.transform.position);
            placementGrid[position.x, position.z] = CellType.Empty;
            Destroy(structure.gameObject);
        }
        temporaryRoadobjects.Clear();
    }

    internal void AddtemporaryStructuresToStructureDictionary()
    {
        foreach (var structure in temporaryRoadobjects)
        {
            structureDictionary.Add(structure.Key, structure.Value);
            DestroyNatureAt(structure.Key);
        }
        temporaryRoadobjects.Clear();
    }

    public void ModifyStructureModel(Vector3Int position, GameObject newModel,
Quaternion rotation)
    {
        if (temporaryRoadobjects.ContainsKey(position))
```

```
            temporaryRoadobjects[position].SwapModel(newModel, rotation);
        else if (structureDictionary.ContainsKey(position))
            structureDictionary[position].SwapModel(newModel, rotation);
    }

    public StructureModel GetRandomRoad()
    {
        var point = placementGrid.GetRandomRoadPoint();
        return GetStructureAt(point);
    }

    public StructureModel GetRandomSpecialStrucutre()
    {
        var point = placementGrid.GetRandomSpecialStructurePoint();
        return GetStructureAt(point);
    }

    public StructureModel GetRandomHouseStructure()
    {
        var point = placementGrid.GetRandomHouseStructurePoint();
        return GetStructureAt(point);
    }

    public List<StructureModel> GetAllHouses()
    {
        List<StructureModel> returnList = new List<StructureModel>();
        var housePositions = placementGrid.GetAllHouses();
        foreach (var point in housePositions)
        {
            returnList.Add(structureDictionary[new Vector3Int(point.X, 0,
point.Y)]);
        }
        return returnList;
    }

    internal List<StructureModel> GetAllSpecialStructures()
    {
        List<StructureModel> returnList = new List<StructureModel>();
        var housePositions = placementGrid.GetAllSpecialStructure();
        foreach (var point in housePositions)
        {
            returnList.Add(structureDictionary[new Vector3Int(point.X, 0,
point.Y)]);
        }
        return returnList;
    }


    private StructureModel GetStructureAt(Point point)
    {
```

```
        if (point != null)
        {
            return structureDictionary[new Vector3Int(point.X, 0, point.Y)];
        }
        return null;
    }

    public StructureModel GetStructureAt(Vector3Int position)
    {
        if (structureDictionary.ContainsKey(position))
        {
            return structureDictionary[position];
        }
        return null;
    }

    public Dictionary<Vector3Int, StructureModel> GetAllStructures()
    {
        return structureDictionary;
    }

    public void ClearGrid()
    {
        placementGrid = new Grid(width, height);
        foreach (var item in structureDictionary.Values)
        {
            Destroy(item.gameObject);
        }
        structureDictionary.Clear();
    }
}
```

Додаток А9 Road Fixer

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

public class RoadFixer : MonoBehaviour
{
    public GameObject deadEnd, roadStraight, corner, threeWay, fourWay;

    public void FixRoadAtPosition(PlacementManager placementManager, Vector3Int
temporaryPosition)
    {
        //[right, up, left, down]
        var result = placementManager.GetNeighbourTypesFor(temporaryPosition);
```

```
        int roadCount = 0;
        roadCount = result.Where(x => x == CellType.Road).Count();
        if(roadCount == 0 || roadCount == 1)
        {
            CreateDeadEnd(placementManager, result, temporaryPosition);
        }else if(roadCount == 2)
        {
            if (CreateStraightRoad(placementManager, result, temporaryPosition))
                return;
            CreateCorner(placementManager, result, temporaryPosition);
        }else if(roadCount == 3)
        {
            Create3Way(placementManager, result, temporaryPosition);
        }
        else
        {
            Create4Way(placementManager, result, temporaryPosition);
        }
    }

    private void Create4Way(PlacementManager placementManager, CellType[] result,
Vector3Int temporaryPosition)
    {
        placementManager.ModifyStructureModel(temporaryPosition, fourWay,
Quaternion.identity);
    }

    //[left, up, right, down]
    private void Create3Way(PlacementManager placementManager, CellType[] result,
Vector3Int temporaryPosition)
    {
        if(result[1] == CellType.Road && result[2] == CellType.Road && result[3]
== CellType.Road)
        {
            placementManager.ModifyStructureModel(temporaryPosition, threeWay,
Quaternion.identity);
        }else if (result[2] == CellType.Road && result[3] == CellType.Road &&
result[0] == CellType.Road)
        {
            placementManager.ModifyStructureModel(temporaryPosition, threeWay,
Quaternion.Euler(0,90,0));
        }
        else if (result[3] == CellType.Road && result[0] == CellType.Road &&
result[1] == CellType.Road)
        {
            placementManager.ModifyStructureModel(temporaryPosition, threeWay,
Quaternion.Euler(0, 180, 0));
        }
        else if (result[0] == CellType.Road && result[1] == CellType.Road &&
result[2] == CellType.Road)
```

```
        {
            placementManager.ModifyStructureModel(temporaryPosition, threeWay,
Quaternion.Euler(0, 270, 0));
        }

    }

    //[left, up, right, down]
    private void CreateCorner(PlacementManager placementManager, CellType[]
result, Vector3Int temporaryPosition)
    {
        if (result[1] == CellType.Road && result[2] == CellType.Road )
        {
            placementManager.ModifyStructureModel(temporaryPosition, corner,
Quaternion.Euler(0, 90, 0));
        }
        else if (result[2] == CellType.Road && result[3] == CellType.Road )
        {
            placementManager.ModifyStructureModel(temporaryPosition, corner,
Quaternion.Euler(0, 180, 0));
        }
        else if (result[3] == CellType.Road && result[0] == CellType.Road )
        {
            placementManager.ModifyStructureModel(temporaryPosition, corner,
Quaternion.Euler(0, 270, 0));
        }
        else if (result[0] == CellType.Road && result[1] == CellType.Road )
        {
            placementManager.ModifyStructureModel(temporaryPosition, corner,
Quaternion.identity);
        }
    }

    //[left, up, right, down]
    private bool CreateStraightRoad(PlacementManager placementManager, CellType[]
result, Vector3Int temporaryPosition)
    {
        if (result[0] == CellType.Road && result[2] == CellType.Road)
        {
            placementManager.ModifyStructureModel(temporaryPosition,
roadStraight, Quaternion.identity);
            return true;
        }else if (result[1] == CellType.Road && result[3] == CellType.Road)
        {
            placementManager.ModifyStructureModel(temporaryPosition,
roadStraight, Quaternion.Euler(0,90,0));
            return true;
        }
        return false;
    }
```

```
    //[left, up, right, down]
    private void CreateDeadEnd(PlacementManager placementManager, CellType[]
result, Vector3Int temporaryPosition)
    {
        if (result[1] == CellType.Road )
        {
            placementManager.ModifyStructureModel(temporaryPosition, deadEnd,
Quaternion.Euler(0, 270, 0));
        }
        else if (result[2] == CellType.Road )
        {
            placementManager.ModifyStructureModel(temporaryPosition, deadEnd,
Quaternion.identity);
        }
        else if (result[3] == CellType.Road )
        {
            placementManager.ModifyStructureModel(temporaryPosition, deadEnd,
Quaternion.Euler(0, 90, 0));
        }
        else if (result[0] == CellType.Road )
        {
            placementManager.ModifyStructureModel(temporaryPosition, deadEnd,
Quaternion.Euler(0, 180, 0));
        }
    }
}
```

Додаток А10 Road Manager

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

public class RoadManager : MonoBehaviour
{
    public PlacementManager placementManager;

    public List<Vector3Int> temporaryPlacementPositions = new List<Vector3Int>();
    public List<Vector3Int> roadPositionsToRecheck = new List<Vector3Int>();

    private Vector3Int startPosition;
    private bool placementMode = false;

    public RoadFixer roadFixer;

    private void Start()
    {
        roadFixer = GetComponent<RoadFixer>();
```

```
    }

    public void PlaceRoad(Vector3Int position)
    {
        if (placementManager.CheckIfPositionInBound(position) == false)
            return;
        if (placementManager.CheckIfPositionIsFree(position) == false)
            return;
        if (placementMode == false)
        {
            temporaryPlacementPositions.Clear();
            roadPositionsToRecheck.Clear();

            placementMode = true;
            startPosition = position;

            temporaryPlacementPositions.Add(position);
            placementManager.PlaceTemporaryStructure(position, roadFixer.deadEnd,
CellType.Road);

        }
        else
        {
            placementManager.RemoveAllTemporaryStructures();
            temporaryPlacementPositions.Clear();

            foreach (var positionsToFix in roadPositionsToRecheck)
            {
                roadFixer.FixRoadAtPosition(placementManager, positionsToFix);
            }

            roadPositionsToRecheck.Clear();

            temporaryPlacementPositions =
placementManager.GetPathBetween(startPosition, position);

            foreach (var temporaryPosition in temporaryPlacementPositions)
            {
                if (placementManager.CheckIfPositionIsFree(temporaryPosition) ==
false)
                {
                    roadPositionsToRecheck.Add(temporaryPosition);
                    continue;
                }
                placementManager.PlaceTemporaryStructure(temporaryPosition,
roadFixer.deadEnd, CellType.Road);
            }
        }

        FixRoadPrefabs();
```

```
    }

    private void FixRoadPrefabs()
    {
        foreach (var temporaryPosition in temporaryPlacementPositions)
        {
            roadFixer.FixRoadAtPosition(placementManager, temporaryPosition);
            var neighbours =
placementManager.GetNeighboursOfTypeFor(temporaryPosition, CellType.Road);
            foreach (var roadposition in neighbours)
            {
                if (roadPositionsToRecheck.Contains(roadposition)==false)
                {
                    roadPositionsToRecheck.Add(roadposition);
                }
            }
        }
        foreach (var positionToFix in roadPositionsToRecheck)
        {
            roadFixer.FixRoadAtPosition(placementManager, positionToFix);
        }
    }

    public void FinishPlacingRoad()
    {
        placementMode = false;
        placementManager.AddtemporaryStructuresToStructureDictionary();
        if (temporaryPlacementPositions.Count > 0)
        {
            AudioPlayer.instance.PlayPlacementSound();
        }
        temporaryPlacementPositions.Clear();
        startPosition = Vector3Int.zero;
    }
}
```

Додаток А11 Simple Camera Controller

```
#if ENABLE_INPUT_SYSTEM
using UnityEngine.InputSystem;
#endif

using UnityEngine;

namespace UnityTemplateProjects
{
    public class SimpleCameraController : MonoBehaviour
    {
        class CameraState
        {
```

```
            public float yaw;
            public float pitch;
            public float roll;
            public float x;
            public float y;
            public float z;

            public void SetFromTransform(Transform t)
            {
                pitch = t.eulerAngles.x;
                yaw = t.eulerAngles.y;
                roll = t.eulerAngles.z;
                x = t.position.x;
                y = t.position.y;
                z = t.position.z;
            }

            public void Translate(Vector3 translation)
            {
                Vector3 rotatedTranslation = Quaternion.Euler(pitch, yaw, roll) *
translation;

                x += rotatedTranslation.x;
                y += rotatedTranslation.y;
                z += rotatedTranslation.z;
            }

            public void LerpTowards(CameraState target, float positionLerpPct,
float rotationLerpPct)
            {
                yaw = Mathf.Lerp(yaw, target.yaw, rotationLerpPct);
                pitch = Mathf.Lerp(pitch, target.pitch, rotationLerpPct);
                roll = Mathf.Lerp(roll, target.roll, rotationLerpPct);

                x = Mathf.Lerp(x, target.x, positionLerpPct);
                y = Mathf.Lerp(y, target.y, positionLerpPct);
                z = Mathf.Lerp(z, target.z, positionLerpPct);
            }

            public void UpdateTransform(Transform t)
            {
                t.eulerAngles = new Vector3(pitch, yaw, roll);
                t.position = new Vector3(x, y, z);
            }
        }

        const float k_MouseSensitivityMultiplier = 0.01f;

        CameraState m_TargetCameraState = new CameraState();
        CameraState m_InterpolatingCameraState = new CameraState();
```

```
        [Header("Movement Settings")]
        [Tooltip("Exponential boost factor on translation, controllable by mouse
wheel.")]
        public float boost = 3.5f;

        [Tooltip("Time it takes to interpolate camera position 99% of the way to
the target."), Range(0.001f, 1f)]
        public float positionLerpTime = 0.2f;

        [Header("Rotation Settings")]
        [Tooltip("Multiplier for the sensitivity of the rotation.")]
        public float mouseSensitivity = 60.0f;

        [Tooltip("X = Change in mouse position.\nY = Multiplicative factor for
camera rotation.")]
        public AnimationCurve mouseSensitivityCurve = new AnimationCurve(new
Keyframe(0f, 0.5f, 0f, 5f), new Keyframe(1f, 2.5f, 0f, 0f));

        [Tooltip("Time it takes to interpolate camera rotation 99% of the way to
the target."), Range(0.001f, 1f)]
        public float rotationLerpTime = 0.01f;

        [Tooltip("Whether or not to invert our Y axis for mouse input to
rotation.")]
        public bool invertY = false;

#if ENABLE_INPUT_SYSTEM
        InputAction movementAction;
        InputAction verticalMovementAction;
        InputAction lookAction;
        InputAction boostFactorAction;
        bool        mouseRightButtonPressed;

        void Start()
        {
            var map = new InputActionMap("Simple Camera Controller");

            lookAction = map.AddAction("look", binding: "<Mouse>/delta");
            movementAction = map.AddAction("move", binding:
"<Gamepad>/leftStick");
            verticalMovementAction = map.AddAction("Vertical Movement");
            boostFactorAction = map.AddAction("Boost Factor", binding:
"<Mouse>/scroll");

            lookAction.AddBinding("<Gamepad>/rightStick").WithProcessor("scaleVec
tor2(x=15, y=15)");
            movementAction.AddCompositeBinding("Dpad")
                .With("Up", "<Keyboard>/w")
                .With("Up", "<Keyboard>/upArrow")
```

```
                .With("Down", "<Keyboard>/s")
                .With("Down", "<Keyboard>/downArrow")
                .With("Left", "<Keyboard>/a")
                .With("Left", "<Keyboard>/leftArrow")
                .With("Right", "<Keyboard>/d")
                .With("Right", "<Keyboard>/rightArrow");
            verticalMovementAction.AddCompositeBinding("Dpad")
                .With("Up", "<Keyboard>/pageUp")
                .With("Down", "<Keyboard>/pageDown")
                .With("Up", "<Keyboard>/e")
                .With("Down", "<Keyboard>/q")
                .With("Up", "<Gamepad>/rightshoulder")
                .With("Down", "<Gamepad>/leftshoulder");
            boostFactorAction.AddBinding("<Gamepad>/Dpad").WithProcessor("scaleVe
ctor2(x=1, y=4)");

            movementAction.Enable();
            lookAction.Enable();
            verticalMovementAction.Enable();
            boostFactorAction.Enable();
        }

#endif

        void OnEnable()
        {
            m_TargetCameraState.SetFromTransform(transform);
            m_InterpolatingCameraState.SetFromTransform(transform);
        }

        Vector3 GetInputTranslationDirection()
        {
            Vector3 direction = Vector3.zero;
#if ENABLE_INPUT_SYSTEM
            var moveDelta = movementAction.ReadValue<Vector2>();
            direction.x = moveDelta.x;
            direction.z = moveDelta.y;
            direction.y = verticalMovementAction.ReadValue<Vector2>().y;
#else
            if (Input.GetKey(KeyCode.W))
            {
                direction += Vector3.forward;
            }
            if (Input.GetKey(KeyCode.S))
            {
                direction += Vector3.back;
            }
            if (Input.GetKey(KeyCode.A))
            {
                direction += Vector3.left;
```

```
            }
            if (Input.GetKey(KeyCode.D))
            {
                direction += Vector3.right;
            }
            if (Input.GetKey(KeyCode.Q))
            {
                direction += Vector3.down;
            }
            if (Input.GetKey(KeyCode.E))
            {
                direction += Vector3.up;
            }
#endif
            return direction;
        }

        void Update()
        {
            // Exit Sample

            if (IsEscapePressed())
            {
                Application.Quit();
#if UNITY_EDITOR
                UnityEditor.EditorApplication.isPlaying = false;
#endif
            }

            // Hide and lock cursor when right mouse button pressed
            if (IsRightMouseButtonDown())
            {
                Cursor.lockState = CursorLockMode.Locked;
            }

            // Unlock and show cursor when right mouse button released
            if (IsRightMouseButtonUp())
            {
                Cursor.visible = true;
                Cursor.lockState = CursorLockMode.None;
            }

            // Rotation
            if (IsCameraRotationAllowed())
            {
                var mouseMovement = GetInputLookRotation() *
k_MouseSensitivityMultiplier * mouseSensitivity;
                if (invertY)
                    mouseMovement.y = -mouseMovement.y;
```

```
                var mouseSensitivityFactor =
mouseSensitivityCurve.Evaluate(mouseMovement.magnitude);

                m_TargetCameraState.yaw += mouseMovement.x *
mouseSensitivityFactor;
                m_TargetCameraState.pitch += mouseMovement.y *
mouseSensitivityFactor;
            }

            // Translation
            var translation = GetInputTranslationDirection() * Time.deltaTime;

            // Speed up movement when shift key held
            if (IsBoostPressed())
            {
                translation *= 10.0f;
            }

            // Modify movement by a boost factor (defined in Inspector and
modified in play mode through the mouse scroll wheel)
            boost += GetBoostFactor();
            translation *= Mathf.Pow(2.0f, boost);

            m_TargetCameraState.Translate(translation);

            // Framerate-independent interpolation
            // Calculate the lerp amount, such that we get 99% of the way to our
target in the specified time
            var positionLerpPct = 1f - Mathf.Exp((Mathf.Log(1f - 0.99f) /
positionLerpTime) * Time.deltaTime);
            var rotationLerpPct = 1f - Mathf.Exp((Mathf.Log(1f - 0.99f) /
rotationLerpTime) * Time.deltaTime);
            m_InterpolatingCameraState.LerpTowards(m_TargetCameraState,
positionLerpPct, rotationLerpPct);

            m_InterpolatingCameraState.UpdateTransform(transform);
        }

        float GetBoostFactor()
        {
#if ENABLE_INPUT_SYSTEM
            return boostFactorAction.ReadValue<Vector2>().y * 0.01f;
#else
            return Input.mouseScrollDelta.y * 0.01f;
#endif
        }

        Vector2 GetInputLookRotation()
        {
```

```
            // try to compensate the diff between the two input systems by
multiplying with empirical values
#if ENABLE_INPUT_SYSTEM
            var delta = lookAction.ReadValue<Vector2>();
            delta *= 0.5f; // Account for scaling applied directly in Windows
code by old input system.
            delta *= 0.1f; // Account for sensitivity setting on old Mouse X and
Y axes.
            return delta;
#else
            return new Vector2(Input.GetAxis("Mouse X"), Input.GetAxis("Mouse
Y"));
#endif
        }

        bool IsBoostPressed()
        {
#if ENABLE_INPUT_SYSTEM
            bool boost = Keyboard.current != null ?
Keyboard.current.leftShiftKey.isPressed : false;
            boost |= Gamepad.current != null ? Gamepad.current.xButton.isPressed
: false;
            return boost;
#else
            return Input.GetKey(KeyCode.LeftShift);
#endif
        }

        bool IsEscapePressed()
        {
#if ENABLE_INPUT_SYSTEM
            return Keyboard.current != null ?
Keyboard.current.escapeKey.isPressed : false;
#else
            return Input.GetKey(KeyCode.Escape);
#endif
        }

        bool IsCameraRotationAllowed()
        {
#if ENABLE_INPUT_SYSTEM
            bool canRotate = Mouse.current != null ?
Mouse.current.rightButton.isPressed : false;
            canRotate |= Gamepad.current != null ?
Gamepad.current.rightStick.ReadValue().magnitude > 0 : false;
            return canRotate;
#else
            return Input.GetMouseButton(1);
#endif
        }
```

```
        bool IsRightMouseButtonDown()
        {
#if ENABLE_INPUT_SYSTEM
            return Mouse.current != null ? Mouse.current.rightButton.isPressed :
false;
#else
            return Input.GetMouseButtonDown(1);
#endif
        }

        bool IsRightMouseButtonUp()
        {
#if ENABLE_INPUT_SYSTEM
            return Mouse.current != null ? !Mouse.current.rightButton.isPressed :
false;
#else
            return Input.GetMouseButtonUp(1);
#endif
        }
    }
}
```

Додаток А12 Structure Manager

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

public class StructureManager : MonoBehaviour
{
    public StructurePrefabWeighted[] housesPrefabe, specialPrefabs,
bigStructuresPrefabs;
    public PlacementManager placementManager;
    public GameManager gameManager;

    private float[] houseWeights, specialWeights, bigStructureWeights;

    private void Start()
    {
        houseWeights = housesPrefabe.Select(prefabStats =>
prefabStats.weight).ToArray();
        specialWeights = specialPrefabs.Select(prefabStats =>
prefabStats.weight).ToArray();
        bigStructureWeights = bigStructuresPrefabs.Select(prefabStats =>
prefabStats.weight).ToArray();
    }
```

```
    public void PlaceHouse(Vector3Int position)
    {
        if (CheckPositionBeforePlacement(position))
        {
            int randomIndex = GetRandomWeightedIndex(houseWeights);
            placementManager.PlaceObjectOnTheMap(position,
housesPrefabe[randomIndex].prefab, CellType.Structure,
buildingPrefabIndex:randomIndex);
            AudioPlayer.instance.PlayPlacementSound();
            gameManager.AddOneBuildingOfPopulation();
        }
    }

    internal void PlaceBigStructure(Vector3Int position)
    {
        int width = 2;
        int height = 2;
        if(CheckBigStructure(position, width , height))
        {
            int randomIndex = GetRandomWeightedIndex(bigStructureWeights);
            placementManager.PlaceObjectOnTheMap(position,
bigStructuresPrefabs[randomIndex].prefab, CellType.BigStructure, width, height,
randomIndex);
            AudioPlayer.instance.PlayPlacementSound();

        }
    }


    public void Destroy(Vector3Int position)
    {
        if (placementManager.CheckIfPositionIsFree(position) == false &&
placementManager.CheckIfPositionInBound(position) != false)
        {
            //Debug.Log("This position is not EMPTY");
            placementManager.DestroyBuildingAt(position);
        }
        //logika = typecell = null on x.y.z
        //destroy obj on input x.y.z
    }


    private bool CheckBigStructure(Vector3Int position, int width, int height)
    {
        bool nearRoad = false;
        for (int x = 0; x < width; x++)
        {
            for (int z = 0; z < height; z++)
            {
                var newPosition = position + new Vector3Int(x, 0, z);
```

```
                if (DefaultCheck(newPosition)==false)
                {
                    return false;
                }
                if (nearRoad == false)
                {
                    nearRoad = RoadCheck(newPosition);
                }
            }
        }
        return nearRoad;
    }

    public void PlaceSpecial(Vector3Int position)
    {
        if (CheckPositionBeforePlacement(position))
        {
            int randomIndex = GetRandomWeightedIndex(specialWeights);
            placementManager.PlaceObjectOnTheMap(position,
specialPrefabs[randomIndex].prefab, CellType.SpecialStructure,
buildingPrefabIndex: randomIndex);
            AudioPlayer.instance.PlayPlacementSound();
            gameManager.AddOneBuildingOfIncome();
        }
    }

    private int GetRandomWeightedIndex(float[] weights)
    {
        float sum = 0f;
        for (int i = 0; i < weights.Length; i++)
        {
            sum += weights[i];
        }

        float randomValue = UnityEngine.Random.Range(0, sum);
        float tempSum = 0;
        for (int i = 0; i < weights.Length; i++)
        {
            //0->weihg[0] weight[0]->weight[1]
            if(randomValue >= tempSum && randomValue < tempSum + weights[i])
            {
                return i;
            }
            tempSum += weights[i];
        }
        return 0;
    }

    private bool CheckPositionBeforePlacement(Vector3Int position)
```

```
    {
        if (DefaultCheck(position) == false)
        {
            return false;
        }

        if (RoadCheck(position) == false)
            return false;

        return true;
    }

    private bool RoadCheck(Vector3Int position)
    {
        if (placementManager.GetNeighboursOfTypeFor(position,
CellType.Road).Count <= 0)
        {
            Debug.Log("Must be placed near a road");
            return false;
        }
        return true;
    }

    private bool DefaultCheck(Vector3Int position)
    {
        if (placementManager.CheckIfPositionInBound(position) == false)
        {
            //Debug.Log("This position is out of bounds");
            return false;
        }
        if (placementManager.CheckIfPositionIsFree(position) == false)
        {
            //Debug.Log("This position is not EMPTY");
            return false;
        }
        return true;
    }

    internal void PlaceLoadedStructure(Vector3Int position, int
buildingPrefabindex, CellType buildingType)
    {
        switch (buildingType)
        {
            case CellType.Structure:
                placementManager.PlaceObjectOnTheMap(position,
housesPrefabe[buildingPrefabindex].prefab, CellType.Structure,
buildingPrefabIndex: buildingPrefabindex);
                break;
            case CellType.BigStructure:
```

```
                placementManager.PlaceObjectOnTheMap(position,
bigStructuresPrefabs[buildingPrefabindex].prefab, CellType.BigStructure, 2, 2,
buildingPrefabindex);
                break;
            case CellType.SpecialStructure:
                placementManager.PlaceObjectOnTheMap(position,
specialPrefabs[buildingPrefabindex].prefab, CellType.SpecialStructure,
buildingPrefabIndex: buildingPrefabindex);
                break;
            default:
                break;
        }
    }

    public Dictionary<Vector3Int, StructureModel> GetAllStructures()
    {
        return placementManager.GetAllStructures();
    }

    public void ClearMap()
    {
        placementManager.ClearGrid();
    }
}

[Serializable]
public struct StructurePrefabWeighted
{
    public GameObject prefab;
    [Range(0,1)]
    public float weight;
}
```

Додаток А13 Structure Model

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

public class StructureModel : MonoBehaviour, INeedingRoad
{
    float yHeight = 0;

    public Vector3Int RoadPosition { get; set; }

    [field: SerializeField]
    public CellType BuildingType { get; private set; }
    [field: SerializeField]
    public int BuildingPrefabIndex { get; private set; }
```

```
    public void CreateModel(GameObject model, int buildingPrefabIndex, CellType
buildingType)
    {
        var structure = Instantiate(model, transform);
        yHeight = structure.transform.position.y;
        BuildingType = buildingType;
        BuildingPrefabIndex = buildingPrefabIndex;
    }

    public void SwapModel(GameObject model, Quaternion rotation)
    {
        foreach (Transform child in transform)
        {
            Destroy(child.gameObject);
        }
        var structure = Instantiate(model, transform);
        structure.transform.localPosition = new Vector3(0, yHeight, 0);
        structure.transform.localRotation = rotation;
    }
}
```

Додаток А13 UI Controller

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

public class UIController : MonoBehaviour
{
    public Action OnRoadPlacement, OnHousePlacement, OnSpecialPlacement,
OnBigStructurePlacement, OnDestroy;
    public Button placeRoadButton, placeHouseButton, placeSpecialButton,
placeBigStructureButton, destroyButton;

    public Color outlineColor;
    List<Button> buttonList;

    private void Start()
    {
        buttonList = new List<Button> { placeHouseButton, placeRoadButton,
placeSpecialButton, placeBigStructureButton, destroyButton };

        placeRoadButton.onClick.AddListener(() =>
        {
            ResetButtonColor();
            ModifyOutline(placeRoadButton);
            OnRoadPlacement?.Invoke();

        });
```

```
        placeHouseButton.onClick.AddListener(() =>
        {
            ResetButtonColor();
            ModifyOutline(placeHouseButton);
            OnHousePlacement?.Invoke();

        });
        placeSpecialButton.onClick.AddListener(() =>
        {
            ResetButtonColor();
            ModifyOutline(placeSpecialButton);
            OnSpecialPlacement?.Invoke();

        });
        placeBigStructureButton.onClick.AddListener(() =>
        {
            ResetButtonColor();
            ModifyOutline(placeBigStructureButton);
            OnBigStructurePlacement?.Invoke();

        });

        destroyButton.onClick.AddListener(() =>
        {
            ResetButtonColor();
            ModifyOutline(destroyButton);
            OnDestroy?.Invoke();

        });

    }

    private void ModifyOutline(Button button)
    {
        var outline = button.GetComponent<Outline>();
        outline.effectColor = outlineColor;
        outline.enabled = true;
    }

    public void ResetButtonColor()
    {
        foreach (Button button in buttonList)
        {
            button.GetComponent<Outline>().enabled = false;
        }
    }
}
```

Додаток А14 AudioPlayer

```
using System.Collections;
```

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

namespace SVS
{

    public class AudioPlayer : MonoBehaviour
    {
        public AudioClip placementSound;
        public AudioSource audioSource;

        public static AudioPlayer instance;

        private void Awake()
        {
            if (instance == null)
                instance = this;
            else if (instance != this)
                Destroy(this.gameObject);

        }

        public void PlayPlacementSound()
        {
            if(placementSound != null)
            {
                audioSource.PlayOneShot(placementSound);
            }
        }
        public void ChangeMute()
        {
            StartCoroutine(waiter());
        }

        private IEnumerator waiter()
        {
            audioSource.mute = !audioSource.mute;
            //Debug.Log("mute");
            yield return new WaitForSecondsRealtime(2);
            audioSource.mute = !audioSource.mute;
            //Debug.Log("Unmute");
        }
    }
}
```