НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації та управління

УДК 004.42

«До захисту допущено»
Завідувач кафедри
_____ *О.А.Павлов*
(підпис) (ініціали, прізвище)
“___”_____2019 р.

Дипломний проект
на здобуття ступеня бакалавра

з напряму підготовки *6.050101 «Комп’ютерні науки»*

на тему: «Інформаційна підтримка генерації міста на 3D-сцені»

Виконав: студент *4* курсу, групи *ІС- 52*

Трухан Григорій Олександрович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *Ст. викл. Проскура С.Л.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації *Ст. викл. Халус О.А.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент *Доц. каф. ТК, к.т.н., доц. Кисленко Ю.І.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації та управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки (програма професійного спрямування) *6.050101*

«Комп’ютерні науки» («Інформаційні управляючі системи та технології»)

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *О.А.Павлов*
(підпис) (ініціали, прізвище)

“___”_____2019 р.

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЕКТ СТУДЕНТУ

Трухану Григорію Олександровичу
(прізвище, ім’я, по батькові)

1. Тема проекту «*Інформаційна підтримка генерації міста на 3D-сцені*»

керівник проекту *Проскура Світлана Леонідівна, ст. викл.*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “*23*”*квітня* 2019 р. №1181-с

2. Термін подання студентом проекту “*03*”*червня* 2019 року

3. Вихідні дані до проекту

Технічне завдання

4. Зміст пояснювальної записки

1. Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі

2. Інформаційне забезпечення: вхідні дані, вихідні дані

3. Математичне забезпечення: змістовна та математична постановки задачі, обґрунтування та опис методу розв’язання

4. Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів

5. Технологічний розділ: керівництво користувача, методика випробувань програмного продукту

5. Перелік графічного матеріалу

1. Схема структурна діяльності

2. Схема структурна варіантів використання

3. Схема структурна фукнціональних вимог

4. Схема структурна класів програмного забезпечення

5. Схема структурна послідовності

6. Схема структурна компонентів

6. Консультанти розділів проекту

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

7. Дата видачі завдання *«15» лютого 2019 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1.	*Вивчення рекомендованої літератури*	*06.02.2019*	
2.	*Аналіз існуючих методів розв'язання задачі*	*15.04.2019*	
3.	*Постановка та формалізація задачі*	*01.05.2019*	
4.	*Розробка інформаційного забезпечення*	*04.05.2019*	
5.	*Алгоритмізація задачі*	*06.05.2019*	
6.	*Обґрунтування використовуваних технічних засобів*	*09.05.2019*	
7.	*Розробка програмного забезпечення*	*12.05.2019*	
8.	*Налагодження програми*	*20.05.2019*	
9.	*Виконання графічних документів*	*23.05.2019*	
10.	*Оформлення пояснювальної записки*	*25.05.2019*	
11.	*Подання ДП на попередній захист*	*29.05.2019*	
12.	*Подання ДП на основний захист*	*01.06.2019*	
13.	*Подання ДП рецензенту*	*05.06.2019*	

Студент _____ Г.О.Трухан
(підпис)

Керівник проекту _____ С.Л.Проскура
(підпис)

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
			Документи текстові		
1	А4	ДП ІС-5222.1181-с.ПЗ	Пояснювальна записка	85	
2	А4	ДП ІС-5222.1181-с.ТЗ	Технічне завдання	10	
3	А4	ДП ІС-5222.1181-с.ПМВ	Програма та методика випробувань	13	
			Матеріали графічні		
4		ДП ІС-5222.1181-с.ССД	Схема структурна діяльності	1	
5	А3	ДП ІС-5222.1181-с.ССВ	Схема структурна варіантів використань	1	
6	А3	ДП ІС-5222.1181-с.ССВ	Схема структурна функціональних вимог	1	
7	А3	ДП ІС-5222.1181-с.ССК	Схема структурна класів програмного	1	
			забезпечення		
8		ДП ІС-5222.1181-с.ССП	Схема структурна послідовності	1	
9		ДП ІС-5222.1181-с.ССК	Схема структурна компонентів	1	

Зм.	Арк.	ПІП	Підп.	Дата	ДП ІС-5222.1181-с.ВДП		
Розробн.		Трухан Г.О.			Відомість дипломного проекту	Літ.	Лист 1 / Листів 1
Керівн.		ПроскураС.Л.					
Консульт.						КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-52	
Н/контр.		Халус О.А.					
в.о.зав. каф		Павлов О.А.					

Пояснювальна записка
до дипломного проекту

на тему: «Інформаційна підтримка генерації міста на 3D-сцені»

Київ – 2019 року

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проекту складається з 6 розділів, містить 40 рисунків, 6 таблиць, 1 додаток.

У дипломному проекті реалізована тема «Інформаційна підтримка генерації міста на 3D-сцені» метою якої є спрощення процесу розробки оточення для відео-ігор на Unreal Engine 4 за рахунок автоматизації розміщення доріг, будинків, парків та декоративних елементів міста.

У розділі загальні положення був визначений процес діяльності для системи, варіанти використання, функціональні вимоги. Були визначені основні аналоги даної системи та наведений порівняльний аналіз. Була поставлена задача, визначені цілі та мета розробки.

У розділі з інформаційного забезпечення був визначений формат вхідних та вихідних даних, описані основні групи налаштувань генерації.

У розділі з математичного забезпечення були описані алгоритми генерації доріг методом ділення прямокутників та методом створення павутиноподібної сітки. Був наведений алгоритм розстановки будинків. Була обґрунтована доцільність використання даних алгоритмів.

У розділі з програмного та технічного забезпечення визначені засоби розробки та технічні вимоги до системи, на якій дане програмне забезпечення буде запускатись. Наведені діаграми класів, послідовності та компонентів.

У технологічному розділі наведена інструкція користувача та проведено тестування системи.

UNREAL ENGINE 4, ПРОЦЕДУРНА ГЕНЕРАЦІЯ, 3D-СЦЕНА, ACET, ІГРОВИЙ РЕДАКТОР, АКТОР

ABSTRACT

Structure and scope of work. The explanatory note of the diploma project consists of 6 sections, containing 40 figures, 6 tables, 1 application.

The diploma project implemented the theme "Information support for city generation on the 3D scene". Project aim is to implement the process of developing the environment for video games based on Unreal Engine 4 due to placing automation of roads, buildings, parks and city decorative elements.

The general provisions describe system's activity process, use cases, functional requirements. Main analogs for current system were found and comparative analysis was created. Taks, goals and objectives of the development were described.

In the information support section, input and output data were identified and main generation settings were described.

In the mathematical support section roads generation algorithms were described – rectangle dividing and net creation algorithm. Buildings placing algorithm was described. The feasibility of using these algorithms was substantiated.

In the software section main development tools and system requirements were defined. Classes, sequence and components diagrams were created.

In the technology section user's manual was described and system was tested.

UNREAL ENGINE 4, PROCEDURAL GENERATION, 3D-SCENE, ASSET, GAME ENGINE, ACTOR

ЗМІСТ

ВСТУП

За останні роки індустрія відео-ігор розвивається великими кроками і цьому є певна кількість причин:

- потужні персональні комп'ютери, які доступні стали доступні майже кожному значно підвищили попит на відео-ігри;

- торгові площадки, в першу чергу Steam, стали більш лояльними до розробників. Так, наприклад, в Steam достатньо заплатити 100 доларів за «торгове місце», викласти свою розроблену гру і отримувати прибуток з її покупок. Більш того, Steam навіть забезпечить її рекламою, якщо аналітичні алгоритми системи визнають таку гру перспективною в плані доходу;

- значно спростився процес розробки. На даний момент невеликій команді розробників не треба писати власний редактор (game engine) для початку розробки – можна скористатися готовими потужними варіантами, які розповсюджуються безкоштовно або за певний відсоток від продаж майбутньої гри (Unreal Engine 4, Unity 5 і т.д.). До того ж розробники можуть скористатися великою кількістю сайтів з безкоштовними асетами для гри (моделі, текстури, інтерфейси, звуки, музика, анімація), якщо не для створення фінальної версії контенту, то хоча б на етапі прототипування;

- величезні прибутки. За 2018 рік дохід від реалізації відео-ігр у світі в перше за історію перевищив дохід від телебачення (116 мільярдів доларів проти 106). Одна тільки GTA5 випущена в 2013 році даними на 2018 рік принесла 6 мільярдів доларів прибутку. На даний момент вона є найбільш успішним ігровим тайтлом в історії і продовжує приносити прибуток.

Але не дивлячись на потужні засоби розробки, створення великих проектів досі лишається важкою задачею, яка навіть у великих студій, де розробкою однієї гри може займатися до 500 людей, займає декілька років.

Тому важливо знайти способи автоматизації робочого процесу розробника, що дозволить зекономити його час. Адже саме зарплатні співробітникам «з'їдають» більшу частину бюджету розробки.

Таким чином розробники намагаються замовляти асети у аутсорс-компаній, а не робити самим, надавати співробітникам максимально однотипні задачі, щоб пришвидшити роботу над нами і так далі. Але одним із найбільш вигідних варіантів автоматизації є розробка процедурної генерації ігрових рівнів, які й займають більшу частину часу розробки.

Засоби процедурної генерації спрощують процес створення великих ігрових сцен, за рахунок автоматичної розстановки асетів оточення за певними алгоритмами.

Подібні засоби бувають повністю автоматизованими, де розробнику потрібно лише ввести налаштування і передати асети генерації, і напів автоматизованими, не розробник сам керує процесом генерації, але більшу частину роботи все одно виконує система.

Практичне значення одержаних результатів. Розроблена система інформаційної підтримки генерації міста на 3D-сцені.

Публікації. Результати роботи були опубліковані у вигляді тез доповіді у науковому збірнику «Секція кафедри автоматизованих систем обробки інформації і управління. Матеріали конференції» II Всеукраїнської науково-практичної конференції молодих вчених та студентів «Інформаційні системи та технології управління»

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1. Опис предметного середовища

На сьогоднішній день індустрія розробки відеоігор активно розвивається у всіх країнах світу. Завдяки створеним умовам для розробників, останні можуть створювати якісні продукти з невеликими командами та бюджетами, або взагалі без бюджету. Сучасний ігровий редактор дозволяє навіть одному програмісту, маючи на руках всі необхідні асети для гри (моделі, звуки, музику) створити готовий для продажі продукт за відносно короткий термін.

Ігровий редактор (game editor / game engine) – це програмне забезпечення для розробки відеоігор. Будь-який сучасний ігровий редактор включає в себе засоби для програмування ігрової логіки (одна або декілька мов, система графічного скриптингу, система створення акторів/префабів), рендер ігрових об’єктів, засоби для редагування 3D та 2D сцен, можливість інтегрування та використання відео, звуків, музики, анімацій різного типу і т.д. [1].

Не дивлячись на розвиток сучасних ігрових редакторів, створення великих рівнів площею в десятки квадратних кілометрів може займати в ігрових студій найвищого рівня декілька місяців, а то і років. Вирішити цю проблему можна розробивши програмне забезпечення, яке допоможе розробникам генерувати окремі частини ігрових рівнів без витрачання місяців на ручне заповнення сцен контентом.

Суть даної роботи полягає в створенні програмного розширення (плагіну) до існуючого програмного забезпечення, найпотужнішого ігрового редактора Unreal Engine 4 [2]. Дане розширення дозволяє генерувати місцевість міського типу, що включає в себе генерацію доріг за декількома алгоритмами, генерацію та розміщення модульних будинків та їх

декоративних елементів, декоративних елементів міста (дорожня розмітка, ліхтарі, світлофори), парків зі своїми доріжками та рослинністю.

Дане застосування може бути використано для розробки відеоігор на базі Unreal Engine 4. З його допомогою можна генерувати місто за бажаними налаштуваннями і передавати в якості параметрів власні 3D-моделі окремих модульних елементів. Після генерації міста всі актори (дороги, будинки, їх елементи і т.д.) будуть розміщені на 3D сцені, де розробник зможе відредагувати їх вручну (за бажанням).

1.1.1. Опис процесу діяльності

Під час створення ігрової карти (рівня) розробники проходять певну кількість робочих процесів, поки не отримають на виході готовий і повністю функціонуючий ігровий рівень.Процес створення ігрового рівня наведений на структурній схемі діяльності в графічному матеріалі.

1.1.2. Опис функціональної моделі

Представлена в даній роботі система надає користувачеві (який є розробником на UE4) набір засобів для процедурної генерації міста та його елементів. Так, в основі лежить актор-контролер генерації, з яким користувач буде взаємодіяти для налаштування та запуску генерації всього міста.

Під час процесу генерації контролер буде розміщувати на сцені інші актори генерації, такі як:

- генератор будинку;
- генератор парку;
- генератор дороги.

Користувач, в свою чергу, може змінити налаштування генератора будинку та парку та пере-генерувати їх окремо, не змінивши генерацію інших елементів міста. Це дозволяє змінювати окремі процедурні елементи згенерованого міста.

Виходячи з описаного функціоналу та можливостей, які має користувач, можна сформулювати структурну схему варіантів використання, яка наведена в графічній частині проекту.

Підсумувавши, можна сказати, що система має функціонал для генерації міста та його окремих елементів, надає користувачу доступ для зміни генерації, введення налаштувань. Виходячи з цього були сформульовані функціональні вимоги до системи. Схема структурна функціональних вимог до програмного забезпечення наведена в графічному матеріалі.

1.2. Користувачі системи

Користувачем системи може бути будь-який спеціаліст, що займається розробкою відеоігор на Unreal Engine 4. Найбільш вірогідні користувачі це:

- level-дизайнер;
- game-дизайнер;
- environment-дизайнер;
- програміст-розробник геймплею.

1.3. Огляд наявних аналогів

Існує певна кількість засобів, які спрощують створення high-end ігрових карт з сучасним містом, що доступні для покупки на Unreal Marketplace [3]. Тим паче жоден з них не включає в себе одночасно генерацію вулиць (доріг) та створення модульних будинків, шо можуть бути змінені вручну вже після генерації. Більш того, всі ці засоби потребують великої кількості роботи для розміщення та налаштування акторів. Саме тому розробка генератора міста, який вимагає мінімум зусиль від користувача для отримання результату є доцільною.

Procedural generator of the highways and roads by Andry K

Рисунок 1.1 – Приклад згенерованої ділянки дороги

Потужний засіб для розміщення доріг на сцені, включає в себе багато декоративних елементів та налаштувань. Дозволяє замінювати всі візуальні елементи на власні, вимагає ручного розміщення лінії дороги [4].

Procedural Building Lot by Ammobox Studios

Рисунок 1.2 – Процедурно згенеровані будинки

Засіб для генерації будинків згідно зі вказаними налаштуваннями. Всі візуальні елементи (елементи архітектури) можна замінити на власні, висока кількість налаштувань. Вимагає ручного розміщення та налаштування кожного будинку [5].

Procedural cities with interiors

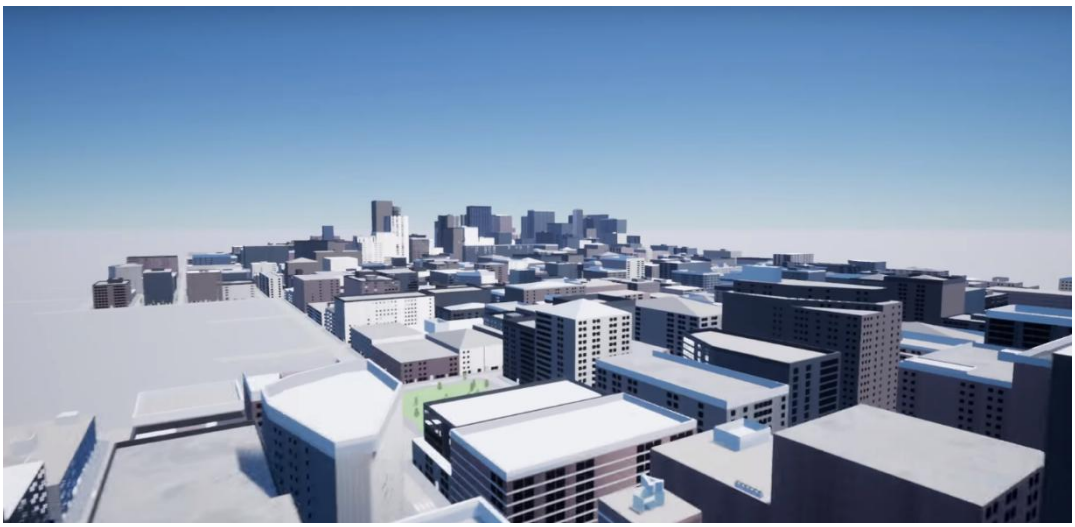


Рисунок 1.3 – Процедурно згенероване місто

Найбільш близький по функціоналу засіб для генерації міста. На вході алгоритм приймає карту висот, що дозволяє задати місця з найменшою висотою (для розміщення парків) та найбільшою (для розміщення хмарочосів). Різноманітна зовнішня форма будинків пояснюється використанням процедурно згенерованих моделей, які добре виглядають на прототипі але не дозволяють кастомізувати будинки, мають низький рівень оптимізації [6].

В таблиці 1.1 наведено порівняльний аналіз основних аспектів представлених аналогів та генерації в даній роботі.

Таблиця 1.1 - Порівняльний аналіз аналогів та даної роботи

	Дана робота	Procedural generator of the highways and roads by Andry K.	Procedural Building Lot by Ammobox Studios	Procedural cities with interiors
Кількість декоративних елементів	Висока (для міста в цілому)	Висока (для доріг)	Висока (для будинків)	Низька. Відсутні дороги, світлофори, наявний лише один асет для заповнення парку
Наявна генерація карти доріг	Так	Ні (треба вказувати положення дороги вручну)	Ні	Частково (генерується карта доріг без моделей)
Наявна генерація будинків	Так	Ні	Так	Так
Наявна генерація інтер’єру	Ні	Ні	Ні	Так

Продовження таблиці 1.1

1	2	3	4	5
Наявні типи карти доріг	Прямокутна, павутино подібна	Відсутня	Відсутня	Прямокутна
Потребує багато ручної роботи	Ні	Так	Так	Ні
Присутня можливість змінювати згенеровані об’єкти	Так	Так	Так	Ні
Кількість налаштувань генерації	Висока (для будинків та їх розміщення, карти доріг, парків, декоративних елементів)	Висока (для доріг)	Висока (для будинків)	Низька (на вхід приймається лише схематична карта (текстура, картинка) майбутнього міста
Оптимізація	Висока (оптимізовані 3д моделі, копіювання створених об’єктів)	Висока (оптимізовані 3д моделі, копіювання створених об’єктів)	Висока (оптимізовані 3д моделі, копіювання створених об’єктів)	Низька (слабка оптимізація у процедурних моделей)

1.4. Постановка задачі

1.4.1. Призначення розробки

Призначенням системи є спрощенн процесу створення великих ігрових сцен завдяки ситемі процедурної генерації міста на 3D-сцені.

1.4.2. Мета та задачі розробки

Метою системи є спрощення процесу розробки сцени (ігрового рівня) при створення прототипу для тестування або фінального ігрового рівня на Unreal Engine 4.

Для досягнення поставленої мети мають бути вирішені такі задачі:

– генерація карт доріг різних типів (клітчата та павутино-подібна);

– генерація будинків вказаної розмірності з використанням заданих 3D-моделей;

– розміщення декоративних елементів будинку;

– розміщення згенерованих будинків на 3D-сцені з урахуванням карти доріг;

– розміщення декоративних елементів міста на 3D-сцені з урахуванням карти доріг;

– генерація та розміщення на 3D-сцені парків, які включають в себе генерацію паркових доріг, розміщення декоративних елементів та рослинності.

Висновок до розділу

В даному розділі було обґрунтовано доцільність створення системи процедурної генерації міста. Були висунуті основні вимоги до системи, виявлені основні групи користувачів, для яких може бути корисною дана система. Були наведені основні аналоги – системи процедурної генерації міста або його елементів, що доступні для купівлі на Unreal Engine

Marketplace або для безкоштовного скачування. Був проведений порівняльний аналіз між існуючими засобами та засобом, що реалізується в даній роботі.

2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Вхідні дані

Вхідні дані для системи процедурної генерації міста надходять від користувача системи (розробника, який використовує процедурну генерацію міста) у вигляді наступних налаштувань:

Налаштування системи генерації доріг включають в себе такі основні групи налаштувань:

- обраний алгоритм генерації доріг;
- частота розміщення елементів доріг;
- посилання на моделі дороги та декоративних елементів міста;
- параметри генерації клітчатої карти доріг;
- параметри генерації павутиноподібної карти доріг.

Налаштування системи генерації будинків:

- діапазон ширини та довжини будинків;
- функція зміни висоти будинків від центру міста до околиць;
- мінімальна дистанція між будинками;
- посилання на моделі, які використовуються для генерації будинків

Налаштування системи генерації парків:

- кількість парків;
- параметри генерації паркових доріг;
- параметри генерації декоративних елементів парку;
- параметри генерації паркової рослинності;
- посилання на всі 3D-моделі що використовуються.

2.2. Вихідні дані

На виході (після завершення генерації) всі актори розміщуються на 3D сцені, яка одразу доступна для перегляду, редагування та збереження в Unreal Engine 4.

З точки зору користувача вихідними даними є сцена [7], з розміщеними на ній акторами (екземплярами класів) з певними налаштуваннями.

З точки зору програми вихідними даними є інформація про оновлену сцену, яка була дозаписана в бінарний файл сцени з розширенням .uasset.

2.3. Структура масивів інформації

Всі результати генерації зберігаються в бінарний файл 3D-сцени в форматі .uasset, в який записуються дані про всі розміщені об’єкти (їх позиції в координатах, налаштування та властивості які були в них на момент закінчення генерації). Цей файл може бути відкритий за допомогою ігрового редактора Unreal Engine 4, змінений та відредагований розробником через редактор.

На рисунку 2.1 наведена діаграма, яка зображує структуру 3D сцени (ігрового рівня) в Unreal Engine 4.

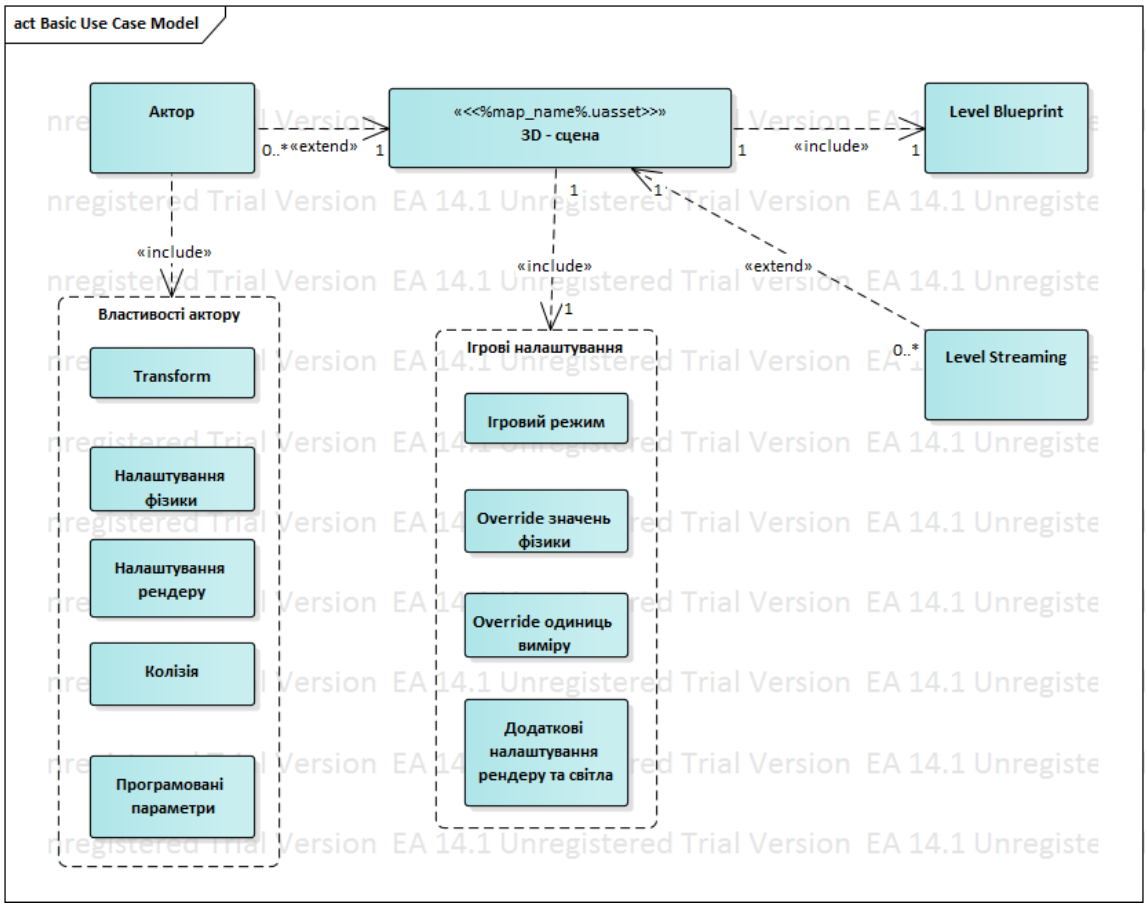


Рисунок 2.1 ─ Структура 3D сцени в UE4

Як видно з рисунку 2.1, основними компонентами сцени є актори, ігрові налаштування та level blueprint.

- *Актори.* Можуть бути присутні або відсутні на сцені. Мають перелік налаштувань, необхідних для функціонування в грі:

1) transform (позиція, кут повороту та множник розміру);

2) налаштування фізики, рендеру, колізії;

3) програмовані параметри. Сюди відносяться параметри та команди, які програміст створює та робить видимими для роботи через редактор. Наприклад, створюючи видимою певну змінну, програміст може змінювати її значення у кожного актора на сцені, а роблячи видимим event (або функцію) програміст може запускати на виконання певний функціонал актору на сцені, прямо під час роботи з редактором (до запуску самої гри).

- *Ігрові налаштування.* Сюди входять глобальні налаштування фізики, світла, одиниць виміру, дальності відрисовки (рендеру) та ін. Налаштування ігрового режиму включають в себе посилання на набір класів, які буде створювати та передавати гравцю редактор під час запуску гри. Сюди входять:

1) запрограмований клас актору, яким буде керувати гравець під час запуску гри;

2) запрограмований клас контролера, який відповідає за управління актором (або акторами) з пункту 1;

3) HUD клас [8] – графічний інтерфейс, яке буде прив’язаний до гравця та інше.

- *Level blueprint* [10] – це скрипт, програмний код, який відноситься до рівня (може існувати в єдиному екземплярі для кожного рівня). Основна відмінність від скриптів звичайних акторів в тому, що:

1) не має конструктора, так як програміст не має доступу до створення об’єкту рівня (класи рівнів створюються лише через графічний інтерфейс)
2) може мати унікальні змінні (атрибути), які одразу ініціалізуються посиланнями на актори на сцені
3) не має компонентів, не може наслідуватись або бути наслідуваним

Подібна структура дозволяє використовувати level blueprint для швидкого та безпечного написання коду, який можна одразу протестувати на створених акторах на сцені, або для створення коду, який буде працювати лише на цьому рівні (корисно при програмуванні ігрової логіки на рівні конкретної карти (3D сцени)).

Висновок до розділу

В даному розділі були описані вхідні та вихідні дані для системи процедурної генерації міста. Було детально перераховано всі налаштування генерації, з якими може працювати користувач.

Було наведено та детально описано діаграму, яка зображує структуру файлу 3D-сцени (ігрового рівня) в яку записуються всі результати генерації (актори, їх компоненти та налаштування).

3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Змістовна постановка задачі

Процедурна генерація контенту в ігровому редакторі (на прикладі Unreal Engine 4) завжди зводиться до таких ключових аспектів:

- Створення та редагування акторів в їх локальному просторі;

- Розміщення акторів на карті згідно заданих правил.

Актор в рамках UE4 це екземпляр класу AActor [11]. Даний об’єкт можна створити лише на сцені, тобто він не може існувати лише в пам’яті, не будучи присутнім на сцені. Актор має свій основний набір методів та атрибутів, які дозволяють працювати з ним на 3D-сцені (кут повороту, вектор координата, налаштування фізики і багато інших).

В даній системі можна визначити такі основні актори на сцені:

- контролер генерації;

- будинок;

- генератор парку з його декоративними елементами;

- актор-паркова доріжка;

- два генератора доріг зі своїми декоративними елементами.

При цьому, наприклад, трава, що розміщується в парку не є актором, а є компонентом актора-генератора парку.

Відповідно виникає ряд математичних задач, які треба вирішити щоб отримати правдоподібну генерацію міста.

Задача розміщення декоративних елементів є достатньо тривіальною і зводиться до простого створення та розміщення акторів на певній дистанції одне від одного. Так, наприклад, світлофори будуть розміщуватись на кожному перехресті доріг, будинки вздовж доріг або в середині кварталів (залежно від типу генерації доріг), на заданій відстані одне від одного уникаючи пересікань геометрії. Паркові лавки та ліхтарі будуть

розміщуватись вздовж паркової доріжки, а елементи рослинності будуть заповнювати парк «по сітці».

Складнішою є задача генерації доріг, де треба врахувати такі моменти, як мінімальна та максимальна довжина вулиць, бажаний розмір кварталів, реалістична інтенсивність доріг і т.д.

Ще одна не тривіальна задача – створення алгоритму генерації доріг в парку, який дозволить отримати реалістичний результат.

3.2. Математична постановка задачі

3.2.1. Задача генерації доріг методом ділення на прямокутники

Призначенням цієї задачі є побудова системи доріг в місті за американським типом. На етапі роботи алгоритмів карта доріг представлена матрицею. Дано:

- розмір прямокутника, в якому будуть згенеровані міські дороги (MxN). Одиниця даної розмірності в ігровому редакторі відповідає одиниці відстані (1.0) на сцені;
- частота розміщення елементів дороги (Freq). Даний параметр відповідає ширині та довжині елемента дороги (в даному алгоритмі елемент дороги (3D модель) має квадратну форму по осям X та Y, довільну по Z);
- мінімальна довжина вулиці L;
- коефіцієнт розбиття (k).

Ціль задачі – оперуючи даними параметрами згенерувати матрицю m цілих чисел розмірності (M/freq)x(N/freq), елементи якої відповідають розміщенню доріг в місті. Елемент матриці m може приймати значення:

- 0 – дороги немає;
- 1 - дорога є, йде під кутом 0 градусів;
- 2 – дорога є, йде під кутом 90 градусів;
- 3 – перехрестя.

3.2.2. Задача генерації доріг методом павутини

Призначенням цієї задачі є побудова системи доріг в місті за радянським типом. На етапі роботи алгоритмів карта доріг представлена набором концентричних кіл, по периметру яких розміщуються точки (майбутні перехрестя доріг). Дано:

- кількість кіл навколо центру міста;
- радіус першого кола;
- кількість точок на колах;
- коефіцієнт зміни радіусу кола;
- максимальний відступ від початкової позиції для точок на першому колі;
- коефіцієнт зміни максимального відступу.

3.2.3. Задача розміщення будинків в кварталі

Призначенням цієї задачі є розміщення будинків всередині кварталу – прямокутної зони на 3D сцені, що обмежена дорогою. Згідно з відстанню від нового будинку до центру міста ми знаємо:

- ширину будинку (W);
- довжину будинку (L);
- висоту будинку (H).

Також користувач задає параметр Offset – дистанцію, яка повинна бути між будинками що розміщуються.

Площа майбутніх будинків задається користувачем через діапазон значень ширини та довжини будинків (наприклад довжина від a до b, ширина – від c до d).

Висота будинків задається функцією, яка є окремою сутністю Unreal Engine 4 і може бути змінена через графічний інтерфейс. Функція задає залежність між відстанню від будинку до центра міста і висотою будинку.

3.3. Обґрунтування методу розв’язання

Задача 3.2.1 може бути вирішена шляхом рекурсивного розбиття прямокутника на менші прямокутники. Рекурсія закінчиться тоді, коли ми досягнемо прямокутника мінімального розміру.

При вирішенні задачі 3.2.2 можна застосувати метод побудови концентричних кіл, розстановки точок на них та з’єднання їх за алгоритмом що описаний далі.

Задача 3.2.3. може бути вирішена послідовним розміщенням будинків від центру кварталу (прямокутнику) до його країв шляхом перебору всіх розміщених в кварталі будинків для пошуку сусіда, біля якого є вільне місце. Сусід біля якого не залишилось жодних вільних місць буде видалятися із списку доступних сусідів щоб спростити складність алгоритму.

Для алгоритму генерації доріг методом павутини треба незначним чином модифікувати алгоритм. Замість початкового будинку, який розміщується всередині кварталу і біля якого починають розміщуватись інші будинки, можна розмістити цілий набір початкових будинків, розставивши їх вздовж всіх міських доріг.

3.4. Опис методів розв’язання

3.4.1. Побудова доріг діленням на прямокутники

В основу можна закласти простий, але ефективний алгоритм рекурсивного ділення прямокутника. Нехай користувач задав, що ширина та довжина міста дорівнює M та N відповідно (в одиницях довжини реального світу), при цьому Freq – ширина та довжина квадратної клітинки, яку ми обираємо за мінімальний математичний розмір одиниці площі міста. Тоді матриця S що є математичною основою майбутньої карти доріг буде мати розмірнсть m x n, де:

$$m = \frac{M}{Freq},$$

$$n = \frac{N}{Freq},$$

$$s_{i,j} \in Z,$$

$$i \in 0 \dots m-1,$$

$$j \in 0 \dots n-1$$

При цьому ми маємо параметр L – мінімальний розмір вулиці, та коефіцієнт k, який буде використаний при діленні прямокутників.

Для більш детального пояснення наведений псевдокод алгоритму.

Крок 0. Згенерувати матрицю S розмірності m×n та заповнити її нулями.

Крок 1. Обрати за поточний прямокутник всю матрицю. Поточний прямокутник заданий індексами його лівого верхнього та правого нижнього краю (i_1,j_1та i_2,j_2). Почати рекурсію.

Крок 2. ЯКШО ширина АБО довжина поточного прямокутника менша за L СТОП. ІНАКШЕ:

Розділити поточний прямокутник на чотири прямокутники розрізанням поточного прямокутника на 4 частини. Індекси матриці, через які пройде вертикальний та горизонтальний розріз визначаються таким чином:

Індекс і відповідає рядку по якому буде розрізаний прямокутник.

$$i = \frac{i_1 + i_2}{2} - Rand(-L * k, L * k)$$

Індекс j відповідає стовпцю, по якому буде розрізаний прямокутник.

$$i = \frac{j_1 + j_2}{2} - Rand(-L * k, L * k)$$

Крок 2.1. По черзі передати координати початку та кінця кожного з чотирьох прямокутників до кроку 2.

3.4.2. Побудова доріг методом «павутини»

Для початку треба згенерувати певну кількість концентричних кіл N. Маємо радіус першого кола R_1 та коефіцієнт зміни радіусу r. Таким чином радіус кожного кола можна вирахувати за формулою:

$$R_i = R_1 * r^{i-1}$$

, де i – порядковий номер поточного кола.

Тобто, маючи за коефіцієнт r число більше за одиницю, радіус кожного наступного кола буде плавно зростати. На периметрі кожного кола на однаковій відстані обирається задана користувачем кількість точок.

На виході можна отримати таку картину:

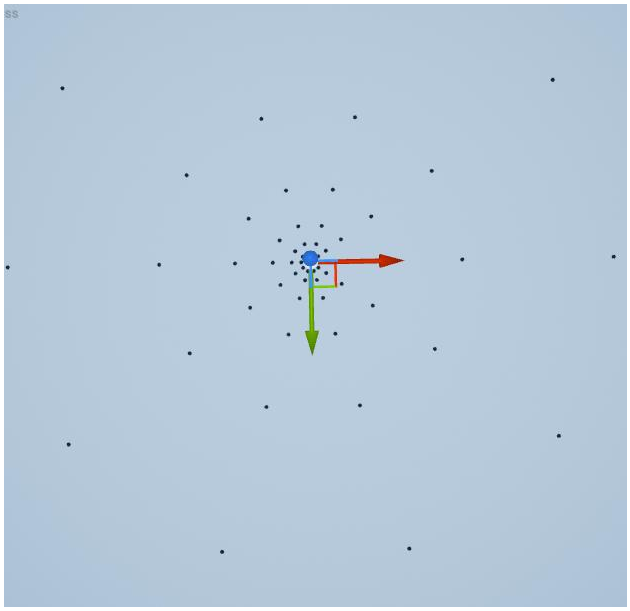


Рисунок 3.1 ─ Схематичний результат роботи алгоритму

Тепер, щоб майбутня карта доріг міста не виглядала занадто «стерильною», використовуємо параметр DotOffset та коефіцієнт k для зсуву. Таким чином, випадковий зсув для кожної точки можна розрахувати за формулою:

$$Offset = Rand\ (-DotOffset * k^{i-1}, DotOffset * k^{i-1})$$

, де Random – повертає випадкове значення в заданому діапазоні;

- i – порядковий номер поточного кола.

Тепер маємо такий прототип для майбутньої карти доріг:

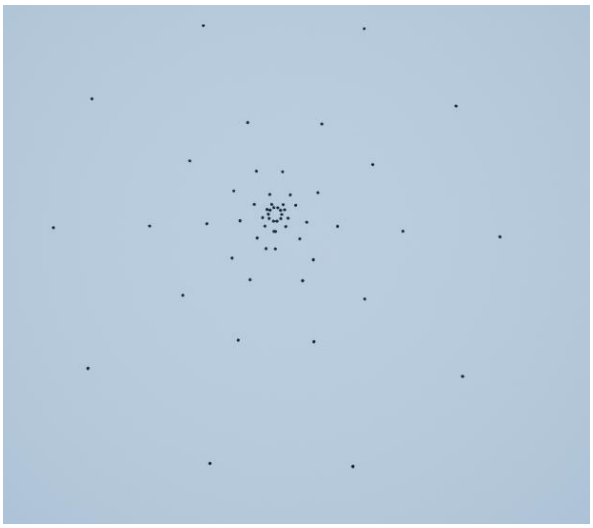


Рисунок 3.2 ─ Схематичний результат роботи алгоритму

Останній етап роботи алгоритму – з’єднати лініями кола між собою. Для цього, для кожної точки на колі і, треба знайти таку точку на колі і+1, що відстань до неї буде мінімальною. Маємо таку карту доріг:

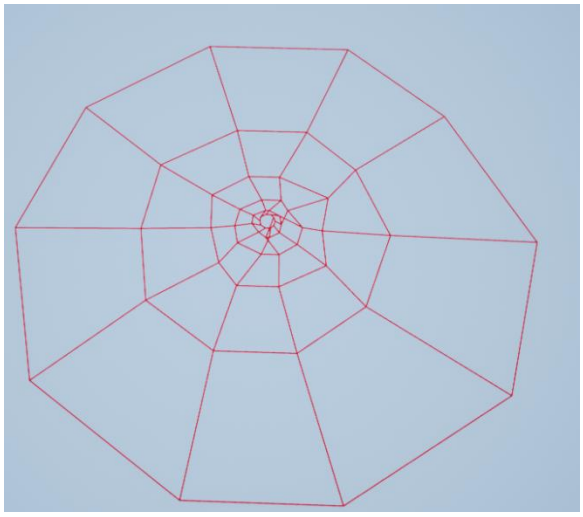


Рисунок 3.3 ─ Схематичний результат роботи алгоритму

3.4.3 Розміщення будинків в кварталі

Задачу розміщення будинків в кварталі можна вирішити наступним чином. Нехай маємо прямокутник на сцені (квартал). Тоді:

КРОК 1. Згенерувати будинок обраної ширини, довжини та висоти (параметри W, L, H що згенеровані відповідно до відстані до центру міста та налаштувань користувача), обрати його за поточний;

КРОК 2. Розмістити будинок в центрі кварталу, обрати його (будинок) за поточний. ЯКЩО геометрія будинку пересікається з геометрією дороги - видалити будинок зі сцени, СТОП. ІНАКШЕ перейти до кроку 3;

КРОК 3. Додати поточний будинок до масиву HousesArray, перемішати масив;

КРОК 4. Згенерувати будинок обраної ширини, довжини та висоти, обрати його за поточний;

КРОК 4.1. Обрати перший індекс HousesArray за поточний;

КРОК 4.2. Спробувати розмістити поточний будинок спереду, зліва, позаду та праворуч (порядок сторін - кожен раз випадковий) будинку, що відповідає поточному індексу HousesArray (при розміщенні врахувати Offset що задав користувач);

КРОК 4.3. ЯКЩО вдалося розмістити без пересікань геометрії - перейти до кроку 3. ІНАКШЕ:

КРОК 4.3.1. видалити будинок що відповідає поточному індексу із масиву (але не зі сцени);

КРОК 4.3.2. ЯКЩО залишились будинки в HousesArray, ТОДІ обрати наступний індекс масиву HousesArray за поточний індекс, перейти до кроку 4.2. ІНАКШЕ знищити будинок, СТОП.

Таким чином будинки починають генеруватися від центру кварталу до його країв, заповнюючи майже весь вільний простір. При цьому залишаються реалістичні "пробіли" між будинками, які в майбутньому можуть бути використані для генерації внутрішніх двориків, а "пробіли" від дороги до будинків - для кіосків, парковок.

Рисунок 3.4 – Результат заповнення кварталів будинками. Ортографічний вигляд

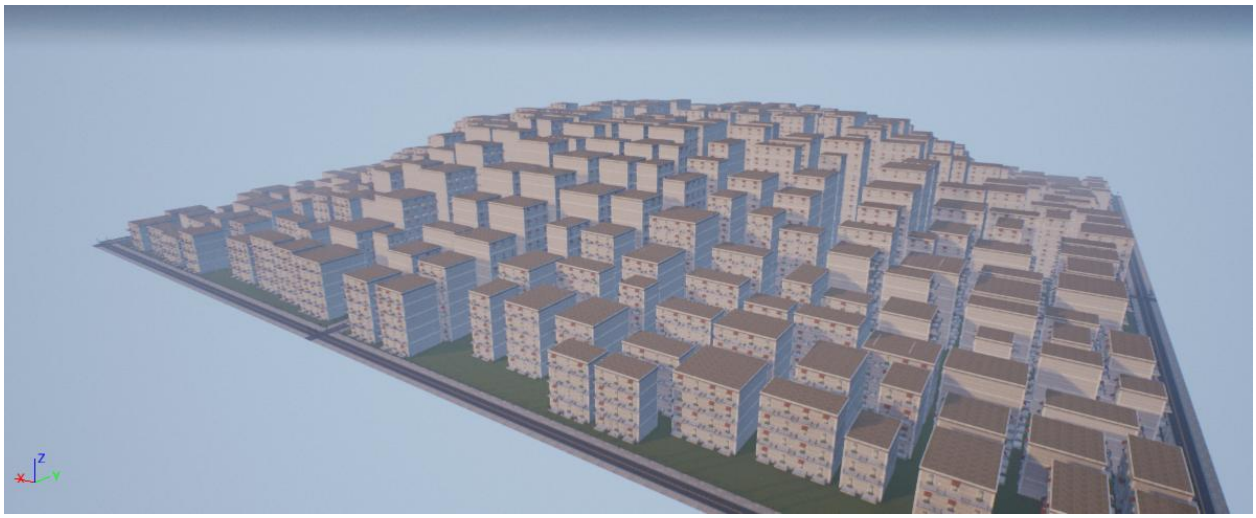


Рисунок 3.5 – Результат заповнення кварталів будинками. Перспектива

Алгоритм розміщення будинків для павутиноподібної сітки доріг дещо відрізняється. На першому кроці будинки послідовно розміщуються вздовж усіх доріг. Далі працює наведений вище псевдокод, але до нього додається додаткова умова - завершити розміщення будинків, якщо не залишилось вільних місць біля будинків, які задовольняють умові:

$$L < R0 + r\_around$$

, де L – відстань від будинку до центру міста;

- R0 – радіус останнього кола, по якому відбувалась генерація дороги;

- r_around – дистанція від останнього кола, на якій можна розміщувати будинки (околиці).

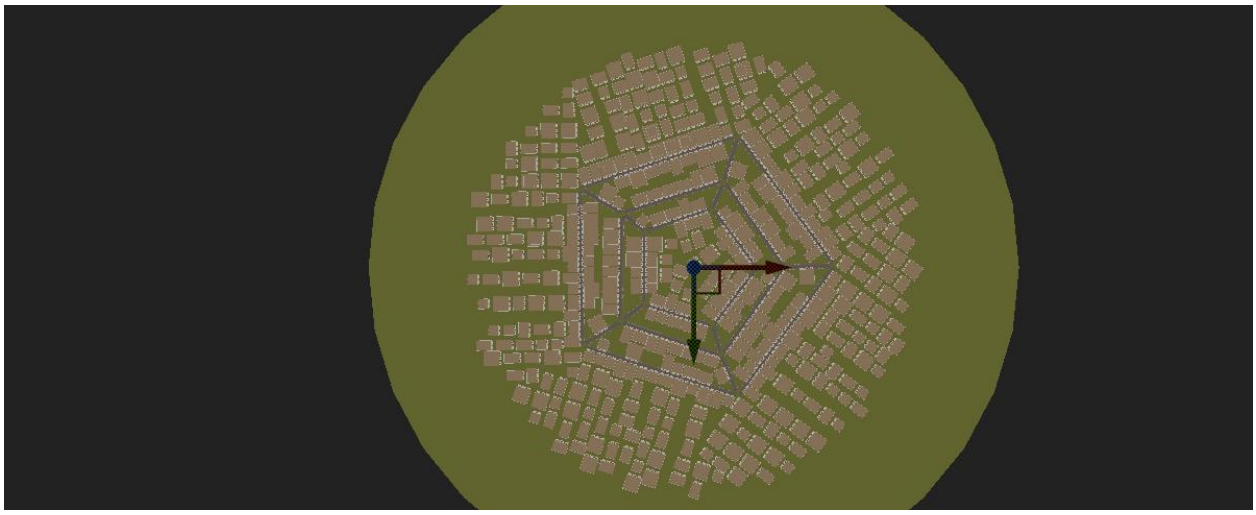


Рисунок 3.6 – Результат заповнення павутиноподібного міста будинками. Ортографічний вигляд

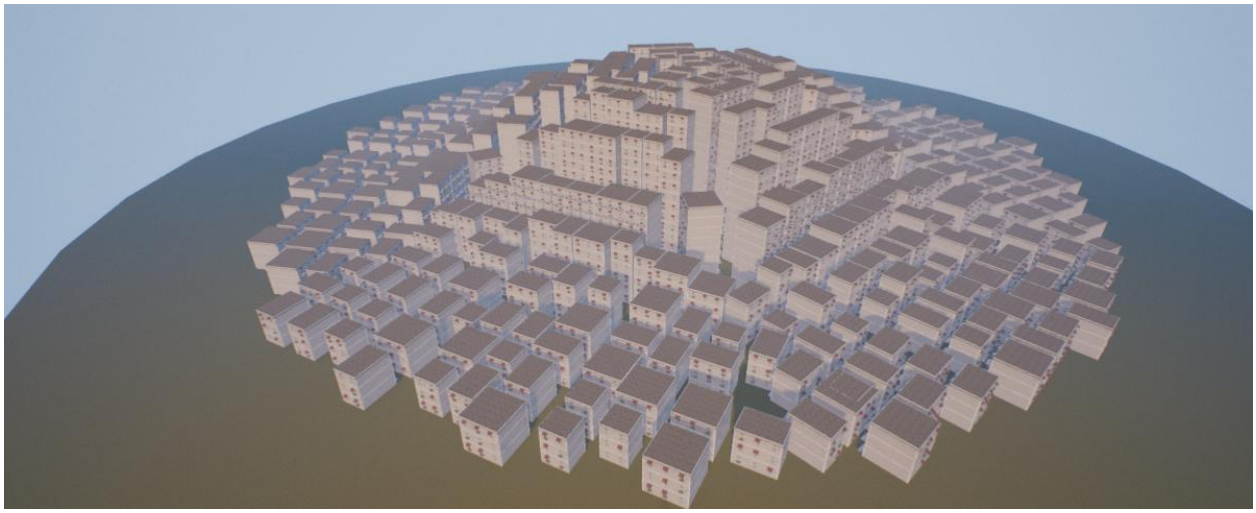


Рисунок 3.7 – Результат заповнення павутиноподібного міста будинками. Перспектива

Висновок до розділу

В даному розділі була сформульована змістовна та математична постановки задачі генерації доріг, будинків та їх розміщення з урахуванням параметрів генерації, які може змінювати користувач. Було запропоновано алгоритм генерації графа-павутини та алгоритм рекурсивного розбиття прямокутників для генерації карти доріг. Був запропонований алгоритм заповнення кварталу згенерованими будинками. Наведено детальний опис алгоритмів розв’язання поставлених задач.

4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1. Засоби розробки

В якості основного засобу розробки було обрано ігровий редактор Unreal Engine 4. Його основними перевагами є:

- відкритий програмний код;
- потужні засоби для роботи з графікою;
- можливість створювати ігри під більшість сучасних платформ: Windows, Linux, Xbox, PlayStation, iOS, Android та ін.;
- вбудована мова візуального програмування.

Головною перевагою Unreal Engine 4, яка дозволяє писати різні системи процедурної генерації – це можливість запускати скрипти на виконання прямо в редакторі, під час роботи зі сценою. Це дозволяє розробнику згенерувати певний контент, а потім приступити до роботи над ним.

В якості мов програмування були обрані C++ [12] та система Unreal Blueprints [13].

C++ - це основна і єдина класична мова програмування яка використовується в даному ігровому редакторі. Використовуючи C++ можна реалізовувати складні і ресурсо-затратні алгоритми, так як дана мова надає потужні інструменти для роботи з пам’яттю.

Unreal Blueprints – це графічна система програмування, яка використовується більшістю розробників на Unreal Engine 4. В ній реалізовані всі основні методології програмування: класи, об’єкти, компоненти, наслідування і тому інше. Для роботи з даними в blueprints реалізований функціонал для роботи з таблицями баз даних, зчитування і запис файлів.

Також в UE4 реалізована можливість об’єднувати код на C++ з системою blueprints. Так, наприклад, можна реалізувати код на C++ у вигляді

функцій, які будуть підключатись до специфічних баз даних і зчитувати інформацію, а потім використовувати ці функції в blueprints, викликати їх, передавати і отримувати значення.

4.2. Вимоги до технічного забезпечення

Реалізована система є розширенням для Unreal Engine 4, тому вимоги до апаратного забезпечення для роботи з системою відповідають вимогам до апаратного забезпечення для роботи з Unreal Engine 4. Для правильної роботи даної програми до складу технічних засобів повинні входити:

- комп’ютер з такою конфігурацією:
  1) двох ядерний процесор з тактовою частотою не нижче 2.5 ГГц;
  2) достатній об’єм оперативної пам’яті (не менше 4 ГБ);
  3) відеокарта NVIDIA GeForce 470 GTX або AMD Radeon 6870 HD series.
- додатково має бути встановлене таке програмне забезпечення:
  1) операційна система Windows 7 64-bit або Mac OS X 10.9.2;
  2) Visual Studio 2017 або новіша, відповідно до встановленої версії Unreal Engine 4;
  3) DirectX 11 або вище;
  4) Unreal Engine 4.21 або вище.
- комп’ютерна периферія, до складу якої входить:
  1) монітор;
  2) мишка або тачпад;
  3) клавіатура.

4.3. Архітектура програмного забезпечення

4.3.1. Діаграма класів

Процедурна генерація міста починається з розміщення на сцені актору CityGenerationController, метод StartGeneration якого є точкою запуску

генерації. Контролер генерації розміщує на сцені актор NetRoadsGenerator або SquareRoadsGenerator (залежно від обраного алгоритму генерації). Два останні актори відповідають за розміщення акторів класу HouseGenerator в залежності від згенерованої карти доріг. Також ці два актори відповідають за генерацію доріг та декоративних елементів міста. HouseGenerator, в свою чергу, відповідає за генерацію конкретного будинку.

У разі, якщо для генерації доріг в місті був обраний алгоритм ділення на прямокутники, буде виконана генерація парків – за кожен парк відповідає актор ParkGenerator. Він розміщується всередині кварталу, та оперуючи його шириною та довжиною починає генерацію парків. Сюди входить розміщення декоративних елементів парку, трави, дерев, кущів, альтанок та доріг між точка всередині парку.

За поєднання цих точок відповідає актор ParkRoadSplineGenerator, який поєднує дві точки плавною, процедурно деформованою дорогою, а також розміщує декоративні елементи (ліхтарі, лавки) вздовж неї.

Схема структурна класів процедурної генерації міста представлена в графічній частині роботи.

4.3.2. Діаграма послідовності

Генерація міста починається зі створення карти доріг, тому контролер генерації передає актору генерації доріг відповідне повідомлення. Після завершення генерації карти доріг контролер генерації отримає інформацію про зони в середині міста, які треба заповнити будинками або розмістити в них парки.

Генератор будинку розміщує власні блоки (моделі) відповідно до налаштувань, а генератор парку будує мережу доріг всередині себе, після чого створює актори-генератори паркових доріг та передає їм відповідні параметри. Кожен генератор паркової дороги розміщує дорогу та її

декоративні елементи, після чого генератор парку заповнює весь вільний простір зеленими насадженнями.

Схема структурна послідовності генерації міста наведена в графічній частині роботи.

4.3.3. Діаграма компонентів

В представленій системі можна виділити ключовий компонент – CityGeneratorController, що відповідає за генерацію всього міста (рисунок 4.3). Він має інтерфейс доступу до інших компонентів генерації (генератори доріг, генератор будинку, парку) через які передає налаштування та команду початку генерації.

Парк-генератор, в свою чергу, створює компоненти, що відповідають за проведення дороги між двома точками (ParkRoadGenerator).

Всі компоненти, що відповідають за генерацію міста передають згенеровані дані на 3D сцену, яка представлена файлом з розширенням .uasset.

Контролер генерації, генератори доріг та генератор парків потребують використання певних математичних алгоритмів, тому вони використовують C++ бібліотеку функцій Lib_GlobalMethods.h.

Схема структурна компонентів системи генерації міста наведена в графічній частині роботи.

4.3.4. Специфікація функцій

В даній роботі реалізована велика кількість методів різних класів та окремих бібліотечних функцій. Деякі з них виконую прості арифметичні або векторні операції, пошук по масиву і т.д. В специфікацію винесені лише найважливіші функції генерації.

В таблиці 4.1 наведена специфікація методів класу BP_CityGenerationController, який є основним класом та контролером генерації міста.

Таблиця 4.1 – Методи класу BP_CityGenerationController

Метод	Опис
CleanScene()	Очищення сцени від всіх акторів генерації
DetachSquaresForPark()	Виділяє задану в налаштуваннях кількість кварталів для генерації парків (повертає структуру-квартал)
GenerateSquarePark(Square)	Запускає генерацію парку у вказаному кварталі
LocateHouseFromPreset(Preset)	Намагається розмістити одноповерховий пресет будинку, коли це вдасться – замінює на реальний будинок з аналогічною прощею, правильною висотою і т.д.
StartGeneration()	Починає генерацію міста згідно заданих налаштувань

В таблиці 4.2 наведена специфікація методів класу BP_ParkRoadSpline, який відповідає за розміщення сплайнової дороги в парку, її деформацію та розміщення декоративних елементів.

Таблиця 4.2 – Методи класу BP_ParkRoadSpline

Метод	Опис
ConstructionScript()	Конструктор, в якому відбувається основна частина генерації паркової дороги
SpawnDeco	Розміщення декоративних елементів вздовж дороги

В таблиці 4.3 наведена специфікація методів класу BP_RoadGeneratorSquare, який відповідає за генерацію прямокутної карти міста, реалізує метод ділення прямокутників та розміщує декоративні елементи міста.

Таблиця 4.3 – Методи класу BP_RoadGeneratorSquare

Метод	Опис
Generation()	Генерація міських кварталів (розміщення доріг)
ClearInstances()	Очищення доріг та декоративних елементів міста
GenerateHousesInSquare(Square)	Розміщення будинків в кварталі
SetHouseUsingVector(Vector)	Розміщує будинок по сусідству з іншим (сусідня сторона визначається одиничним вектором)
AddRoadInstance()	Розміщує елемент дороги
GetCityCenter()	Повертає координату центра міста
AddBorderLine(Vector, Vector)	Розміщує огорожу між двома точками на площині

В таблиці 4.4 наведена специфікація методів класу BP_NetworkGenerator, який відповідає за розміщення генерацію павутиноподібної карти міста, розміщення елементів доріг, декоративних елементів міста та будинків.

Таблиця 4.4 – Методи класу BP_NetworkGenerator

Метод	Опис
Generate()	Запуск генерації
Clean()	Очищення всіх доріг та декоративних елементів міста
InCircleConnection()	Зєднаннь концентричніх кіл дорогами між собою

Продовження таблиці 4.4

Метод	Опис
ConnectCircles()	Зєденнаня точок на колі дорогами
GetLastRadius()	Повертає радіус останнього кола-дороги
IsHouseInCityRadius(House)	Перевіряє чи актор-будинок в середині міста (чи відстань до центру міста менша за останній радіус генерації)
GenerateHousesAlongTheRoads()	Розміщує будинки вздовж міських доріг

В таблиці 4.5 наведена специфікація методів класу BP_HouseGenerator, обєект якого є самостійним актором будинком та генератором самого будинку.

Таблиця 4.5 – Методи класу BP_HouseGenerator

Метод	Опис
ConstructionScript()	Конструктор та основний метод генерації будинку
CheckCollision()	Перевіряє колізію між будинком та іншими елементами містами
GenerateBuildSide(Vector)	Генерує сторону будинку основуючись на векторі-направленні
GenerateDeco()	Розміщення декоративних елементів будинку

В таблиці 4.6 наведена специфікація методів класу BP_ParkGenerator, який відповідає за генерацію парку.

Таблиця 4.6 – Методи класу BP_ParkGenerator

Метод	Опис
MiddleDotsGeneration()	Заповнення парку сіткою із точок
SelectMiddleDots()	Виділення точок в парку для альтанок
ConnectDots()	Зєднання двух точок доріжкою

Продовження таблиці 4.6

Метод	Опис
CreateNetwork()	Створення графу з точок (будуть зєднані доріжками)
FillGrass	Заповнення парку рослинністю
Generate()	Початок генерації парку

В таблиці 4.7 наведена специфікація функцій бібліотеки Lib_GlobalMethod.

Таблиця 4.7 – Функції бібліотеки Lib_GlobalMethods

Lib_GlobalMethods Бібліотека функцій	
GenerateRoadMatrix	Генерація матриці доріг (метод ділення прямокутників)
DivideSquare	Функція ділення матриці (прямокутника)
GenerateDotsOnCircle	Генерація точок на колі (метод створення павутини)
DotsConnection_CPlus	Зєднання точок на «павутині»
MiddleDotsGeneration_CPlus	Генерація точок всередині парку
AddSideDotsBetween_CPlus	Пошук бокових точок парку (виходи)
SelectDotsByRules_CPlus	Пошук точок для розміщення альтанок в парку
ShuffleVectorsArray	Перемішування масиву векторів
FindClosestVector	Знаходження найближчої вектор-координати з масиву до заданої координати

Висновок до розділу

В даному розділі були наведені засоби, які використовувались для розробки даної системи (редактор Unreal Engine 4, мова програмування C++, система графічного програмування Unreal Blueprints). Було обгрунтовано вибір засобів розробки, їх переваги для створення даної системи. Були наведені вимоги до технічного та програмного забезпечення необхідного для функціонування системи.

Було наведену детальну діаграму класів та коротко описано основний функціонал кожного класу. Було наведено та описано діаграму послідовності, яка ілюструє процес генерації міста, а також діаграму компонентів.

5. ТЕХНОЛОГІЧНИЙ РОЗДІЛ

5.1. Керівництво користувача

В процесі реалізації даної роботи було створено наступний функціонал генерації на 3D сцені в Unreal Engine 4:

- генерація доріг алгоритмом ділення прямокутників;
- генерація доріг алгоритмом побудови павутини;
- генерація будинку;
- розміщення будинків;
- генерація парку.

При цьому наявні наступні класи акторів, з якими користувач буде працювати на сцені:

- city generation controller – контролер генерації. Щоб почати генерацію потрібно витягнути його на сцену, ввести налаштування та натиснути Start Generation (деталі описані нижче);
- park generator – відповідає за генерацію парку, додатково розміщує актори Park Road Generator (паркова доріжка). Генератор парку дозволяє перезапускати генерацію конкретного парку або повністю видалити його;
- square roads generator – генерує прямокутну карту доріг;
- circle roads generator – генерує павутиноподібну карту доріг;
- house Generator – керує генерацією конкретного будинку.

5.1.1. Генерація доріг алгоритмом ділення прямокутників

Було реалізовано генерацію доріг, які ділять місто на прямокутні квартали для майбутнього розміщення парків або будинків.

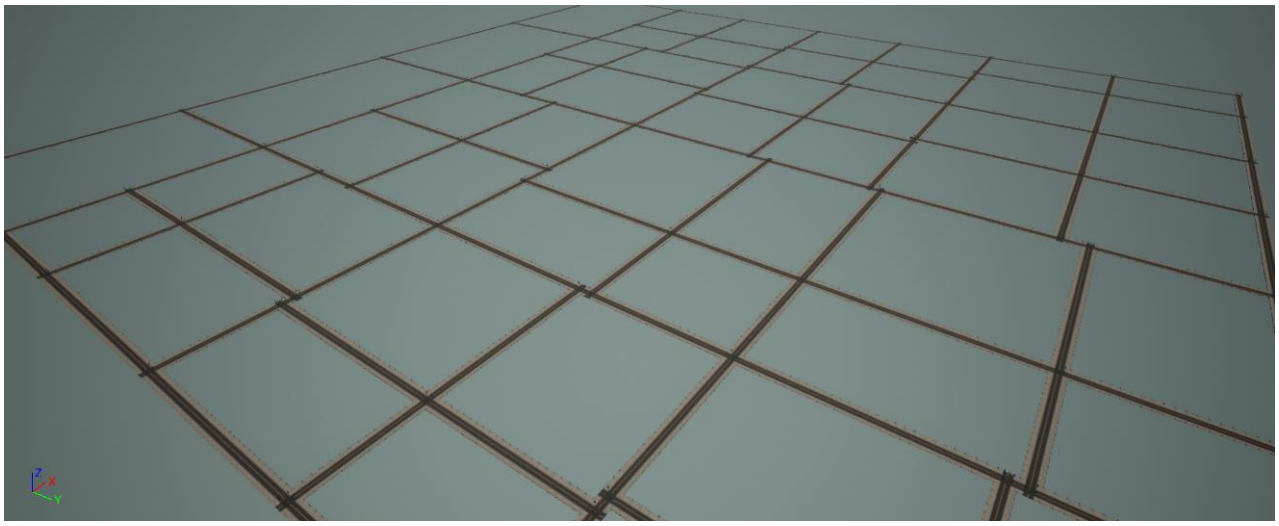


Рисунок 5.1 – Результат генерації карти доріг методом ділення прямокутників

Рисунок 5.2 – Зовнішній вигляд доріг зблизька

Як видно з рисунків, елементи дороги розставляються під коректним кутом, також зроблені окремі моделі для перехресть.

5.1.2. Генерація доріг алгоритмом побудови павутини

Було реалізовано генерацію доріг методом створення павутиноподібної сітки. У внутрішніх секторах та навколо міста на заданій відстані будуть розміщуватись будинки.

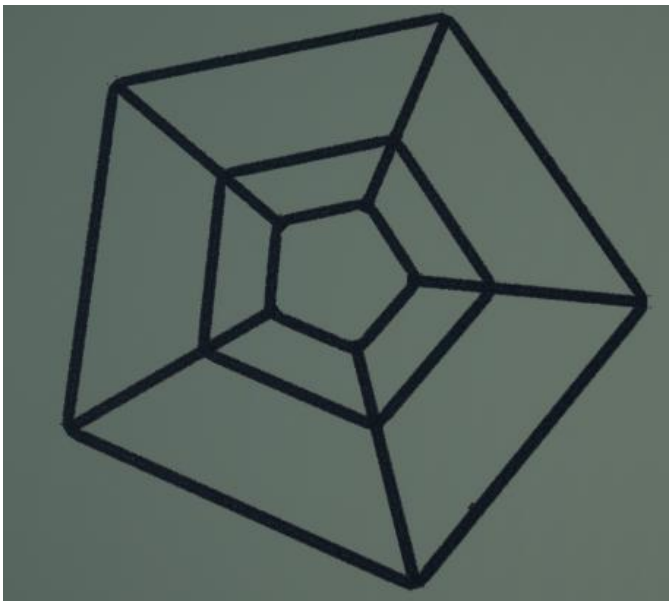


Рисунок 5.3 – Результат генерації доріг методом створення павутиноподібної сітки

Рисунок 5.4 – Результат генерації доріг зблизька

Як видно з малюнків, елементи дороги розміщуються вздовж сітки доріг та трішки деформуються біля перехресть доріг щоб плавно входити в них без виникнення пробілів або накладання моделей одна на одну.

5.1.3. Генерація будинку

Було реалізовано алгоритм генерації будинку, який зводиться до розміщення тайлів з однаковою розмірністю.

Рисунок 5.5 – Згенерований модульний будинок (вигляд зпереду та ззаду)

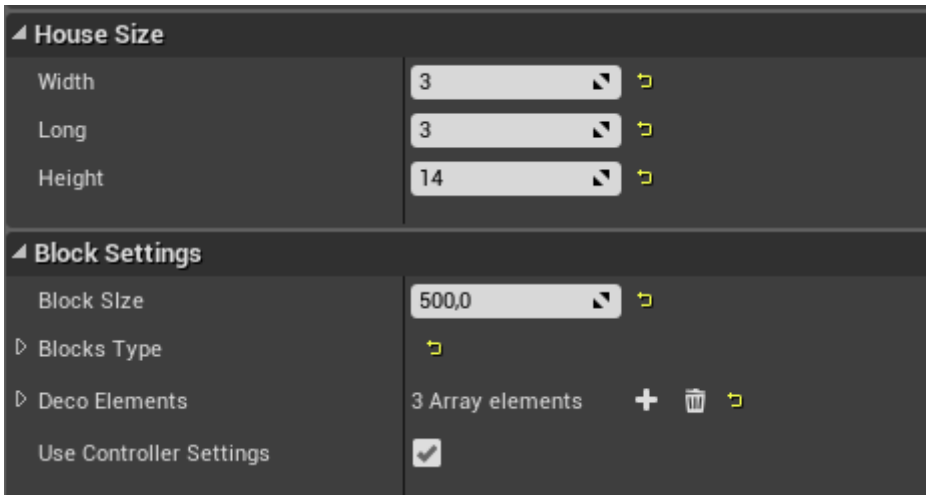


Рисунок 5.6 – Налаштування щойно згенерованого будинку

Налаштування актора будинку зображені на малюнку 5.4 дозволяють змінювати вже згенерований будинок (після генерації міста), або щойно створений будинок (вручну розміщений на сцені).

Налаштування в групі HouseSize відповідають за розмір будинку (кількість блоків по ширині, довжині та висті). Група BlockSettings включає в себе:

- BlockSize - головний параметр що відповідає за розмір блоку;

- BlocksType – масив посилань на 3D-моделі, які використовуються для генерації різних сторін будинку (рисунок 5.7);

- DecoElements – масив посилань на декоративні елементи, ймовірність їх розміщення та відносний відступ від переднього блоку будинку (рисунок 5.8).

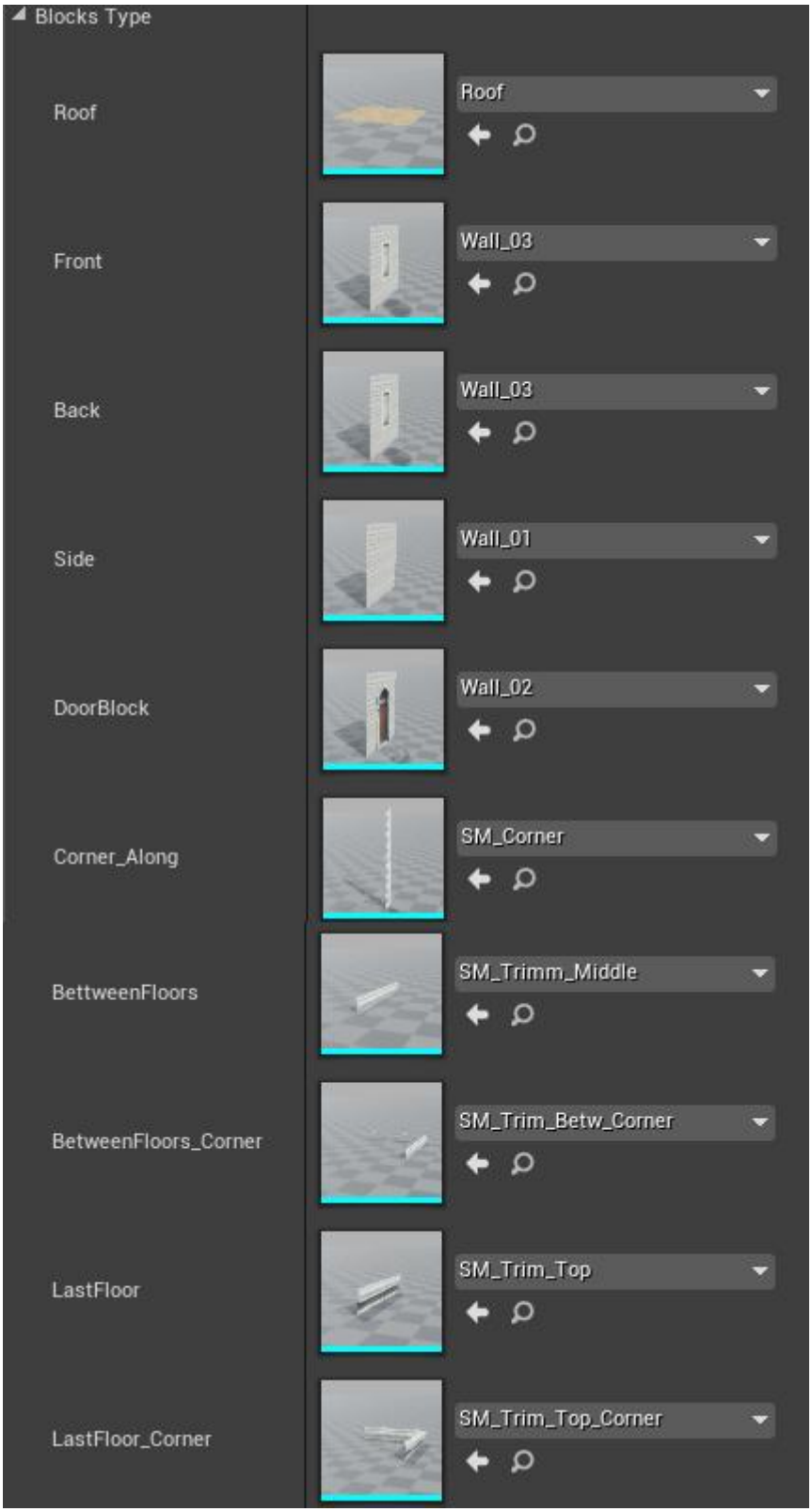


Рисунок 5.7 – Посилання на тайли будинку

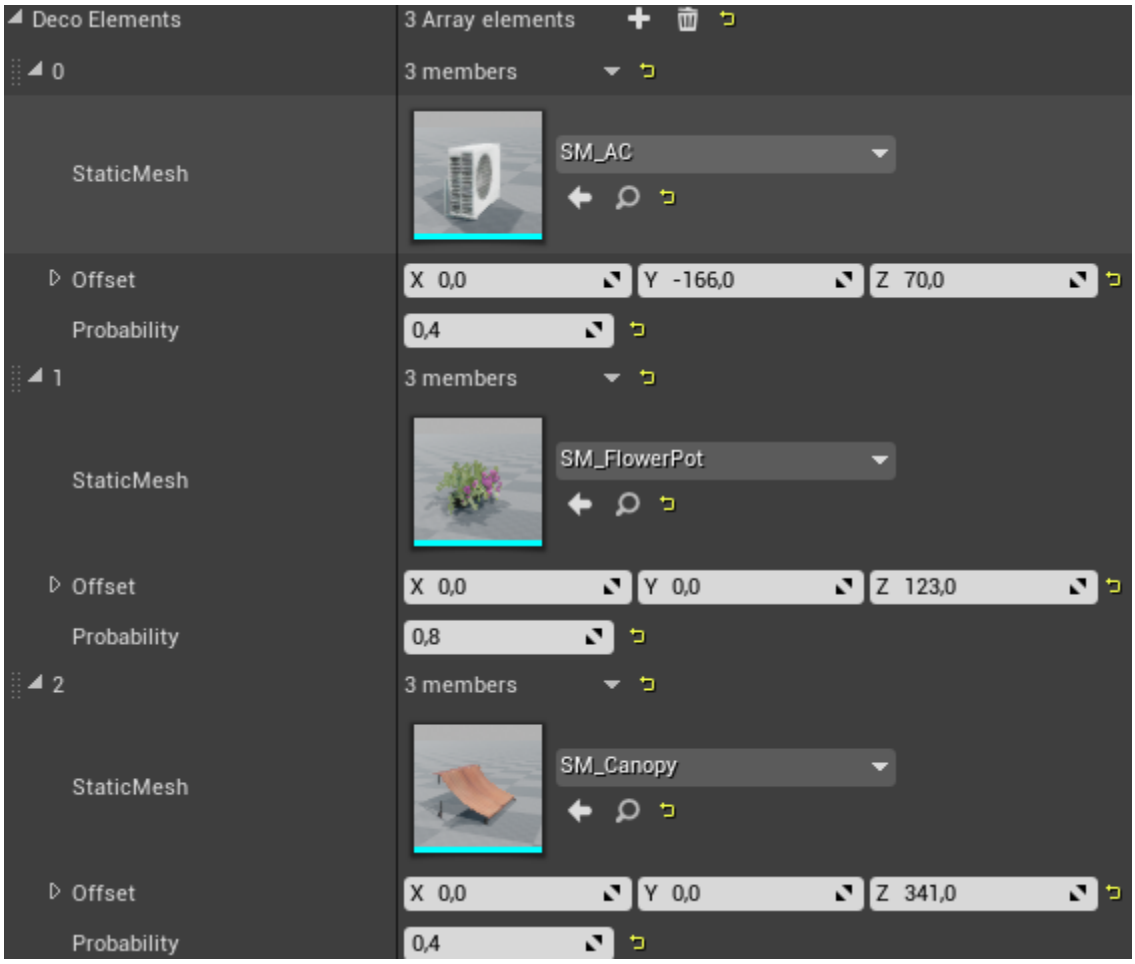
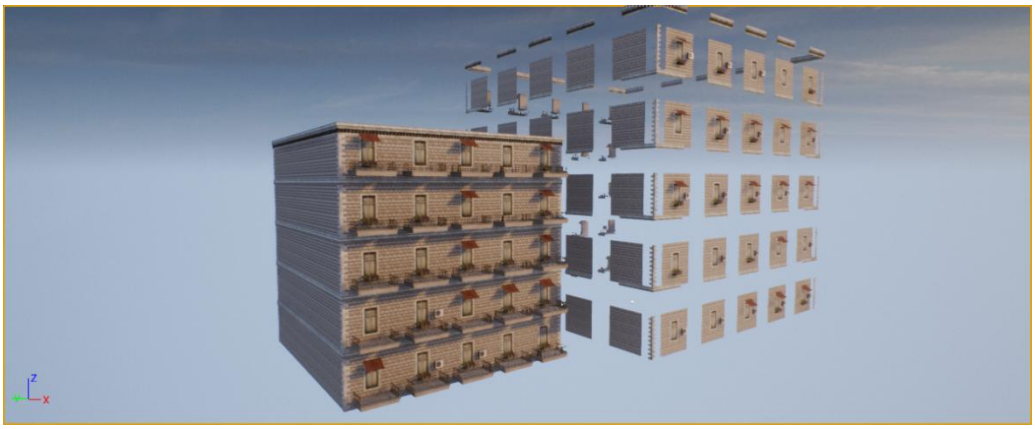


Рисунок 5.8 – Налаштування генерації декоративних елементів

Bool-змінна UseControllerSettings відповідає за джерело налаштувань. Якщо true, будинок буде використовувати налаштування задані в контролері генерації, якщо false – налаштування на самому акторі будинку.

Збільшивши параметр BlockSize таким чином, щоб він був більший за реальний розмір блоку, можна «зазирнути» всередину будинку (рисунок 5.9). Пере-генерація будинку викликається під час зміни будь-яких параметрів будинку.



Малюнок 5.9 – Будинок з нормальним розміром блоку та завищеним

5.1.4. Розміщення будинків

Розміщення будинків при генерації доріг алгоритмом ділення прямокутників виглядає наступним чином:

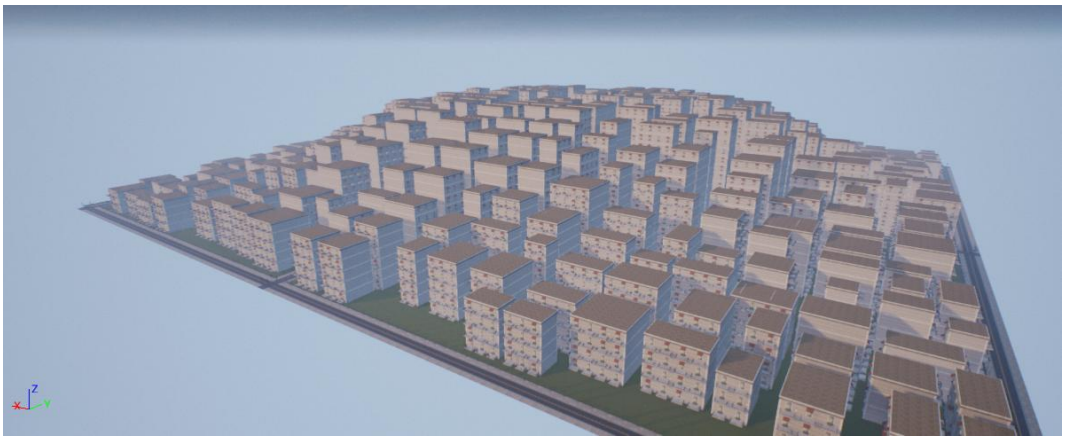


Рисунок 5.10 – Розміщення будинків в міських кварталах

Будинки починають розміщуватись від центру кварталу до його країв, по сусідству одне з одним, уникаючи колізію з іншими об’єктами (якщо наявна колізія – будинок видаляється).

У місті згенерованому за павутиноподібною схемою будинки спочатку розміщуються вздовж доріг, а вже потім по сусідству з розміщеними. Розміщення виглядає наступним чином:

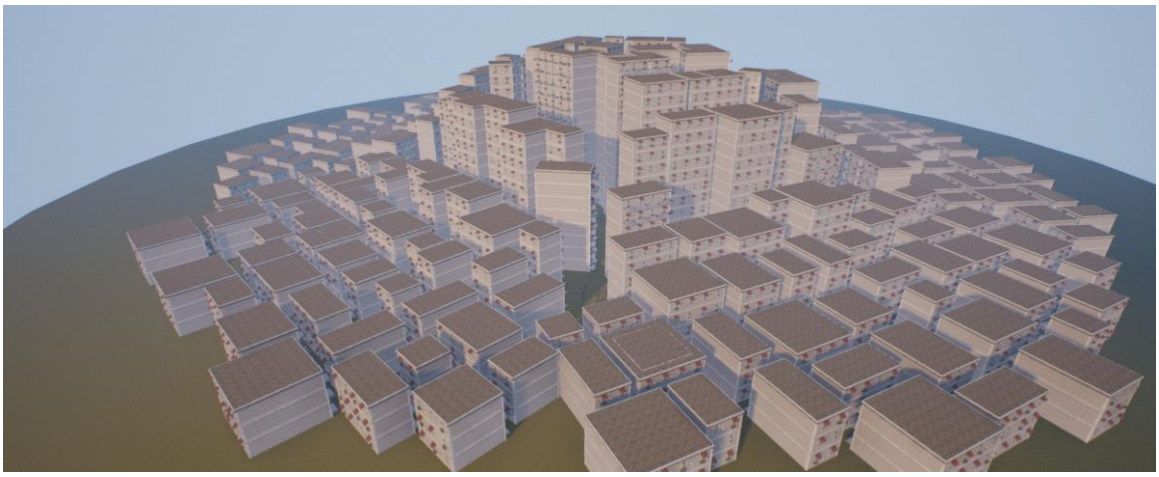


Рисунок 5.11 – Розміщення будинків в місті-павутині

5.1.5. Генерація та розміщення міських парків

Кількість парків для генерації в місті визначається згідно з вказаною користувачем кількістю в контролері генерації.

Рисунок 5.12 – Місто з одним кварталом та одним парком

Коли квартал для розміщення парку обраний, алгоритм розміщує головні точки парку на певній відстані одне від одного (альтанки).

Рисунок 5.13 – Альтанка розміщена процедурно в парку

Далі альтанки з’єднуються дорогами, вздовж доріг розміщуються лавки та ліхтарі.

Рисунок 5.14 – Згенерована дорога між альтанками

Останній етап – весь вільний простір заповнюється рослинністю.

Рисунок 5.15 – Згенеровані трава та дерева

Генератор парку є повністю модульним елементом. Тому його можна видалити (весь парк видалиться разом з ним) та перегенерувати. Для цього треба обрати актор генерації парку, перейти на вкладку details та натиснути кнопку Generate Park.

Рисунок 5.16 – Кнопка запуску генерації парку (окремо від міста)

Якщо треба змінити налаштування генерації парку, треба звернутись до категорії ParkSettings актору ParkGenerator. Налаштування парку зображені на рисунку 5.17.

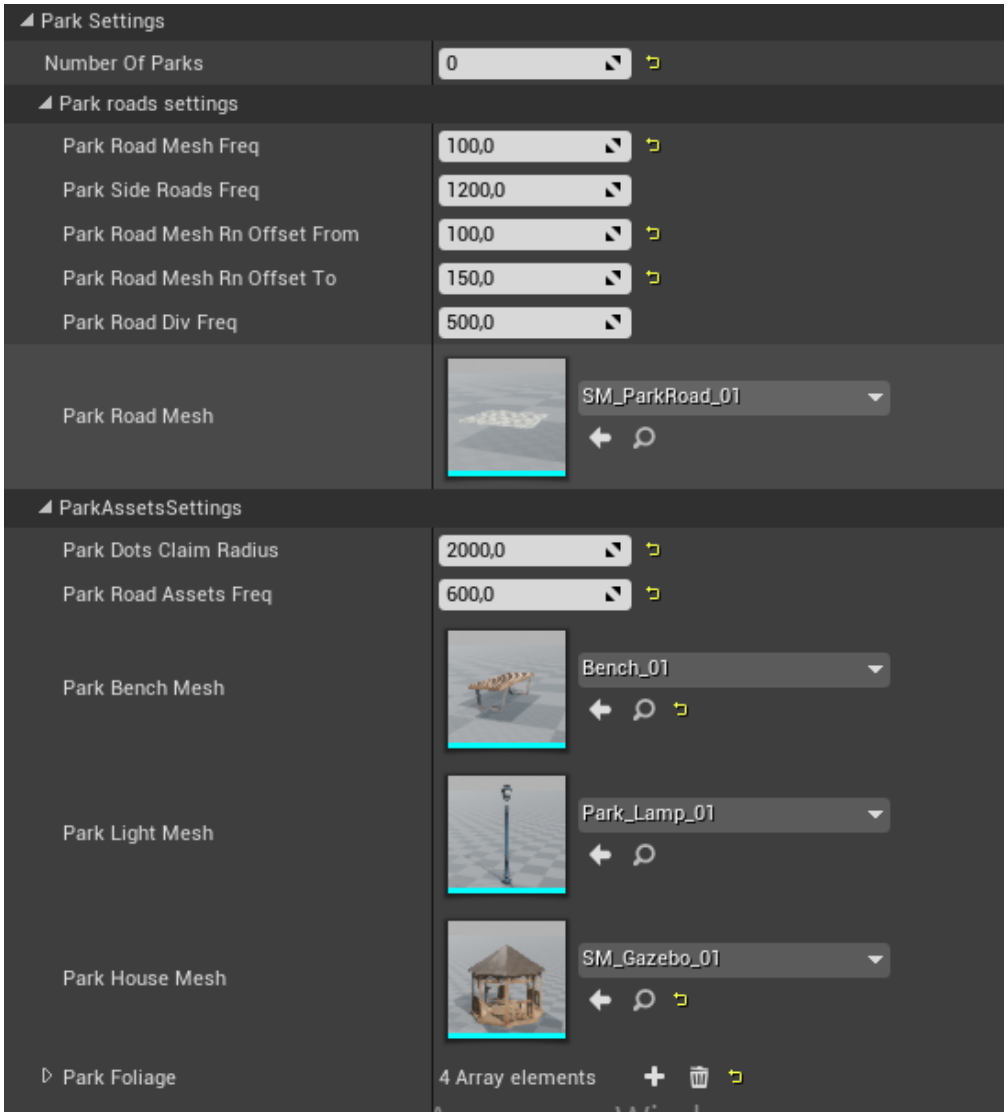


Рисунок 5.17 – Налаштування генерації парку

Маємо декілька основних налаштувань та груп налаштувань:

- number of parks – кількість парків в місті
- park roads settings:

    1) park road mesh freq – розмір тайлу паркової дороги;

    2) park road side freq – відстань між «виходами» з парку. Регулює їх частоту та кількість;

    3) park road mesh rn offset from / to – чим більші параметри, тим більше точки на парковій дорозі будуть «відхилятися» від прямої лінії. Ці два параметри задають діапазон відхилення;

    4) park road div freq – часто ділення дороги на точки, які відхилюються за попереднім налаштуванням;

    5) park road mesh – посилання на тайл паркової дороги.

- park assets setings:

    1) park dots claim radius – мінімальна відстань між альтанками;

    2) park road assets freq – частота розміщення декоративних моделей парку вздовж доріг (лавка, ліхтар);

    3) три посилання на декоративні моделі парку – лавка, ліхтар та альтанка.

- park foliage включає в себе посилання на моделі рослинності, які будуть розміщуватись в парку. Сюди включається ймовірність розміщення, частота розміщення та чи треба генерувати і перевіряти колізію (рисунок 5.18).

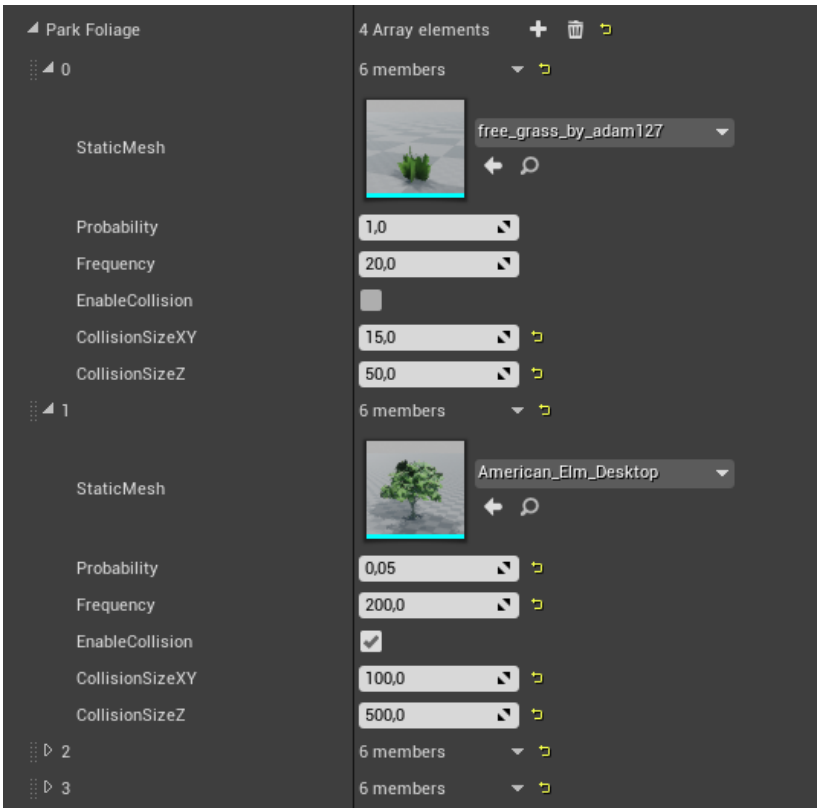


Рисунок 5.18 – Налаштування генерації рослинності

5.1.6. Контролер генерації

Користувач може використовувати актор генерації міста, щоб одразу ввести налаштування та запустити генерацію. Для цього треба витягти на сцену CityGenerationController та перейти у вкладку details. Всі налаштування генерації знаходяться в групі UserSettings, керуючі кнопки – в Defaults (рисунок 5.19).

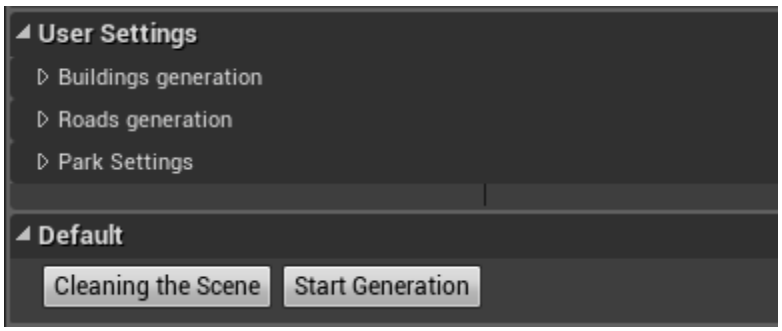


Рисунок 5.19 – Групи налаштувань генерації та керуючі кнопки

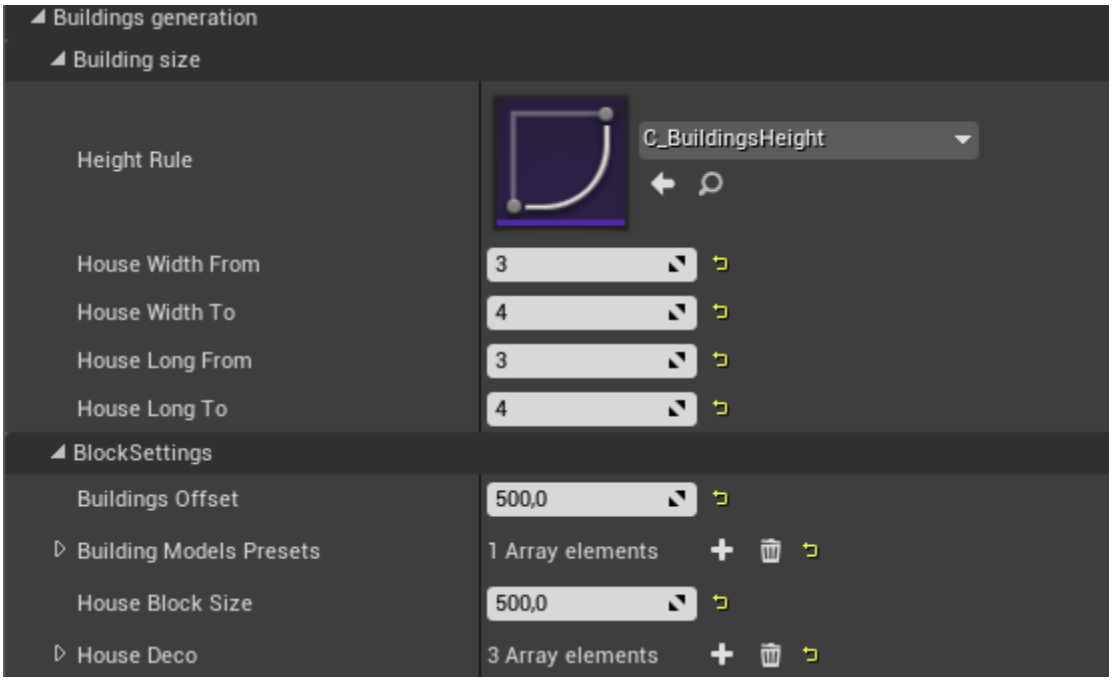


Рисунок 5.20 – Налаштування генерації будинків

Група building generation (рисунок 5.20) включає в себе:

- посилання на функцію залежності висоти будинку від відстані до центру (рисунок 5.21);

- чотири налаштування які задають діапазон, в якому буде визначатись випадкова довжина та ширина будинків;

- buildings fffset – відстань між будинками яка буде враховуватись при генерації будинків.

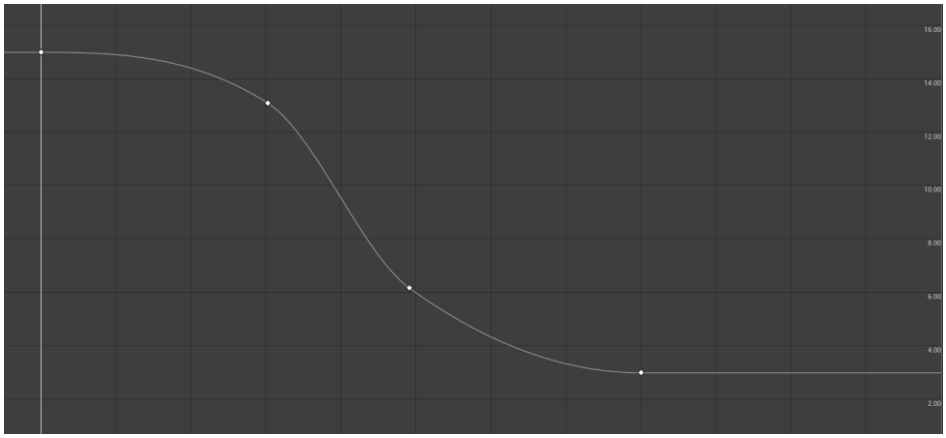


Рисунок 5.21 – Функція залежності висоти будинку від відстані до центру міста

Інші налаштування генерації будинків були описані в розділі 5.1.2.

Група RoadsGeneration включає декілька основних груп налаштувань. Перша – налаштування генерації сітки доріг (рисунок 5.22)

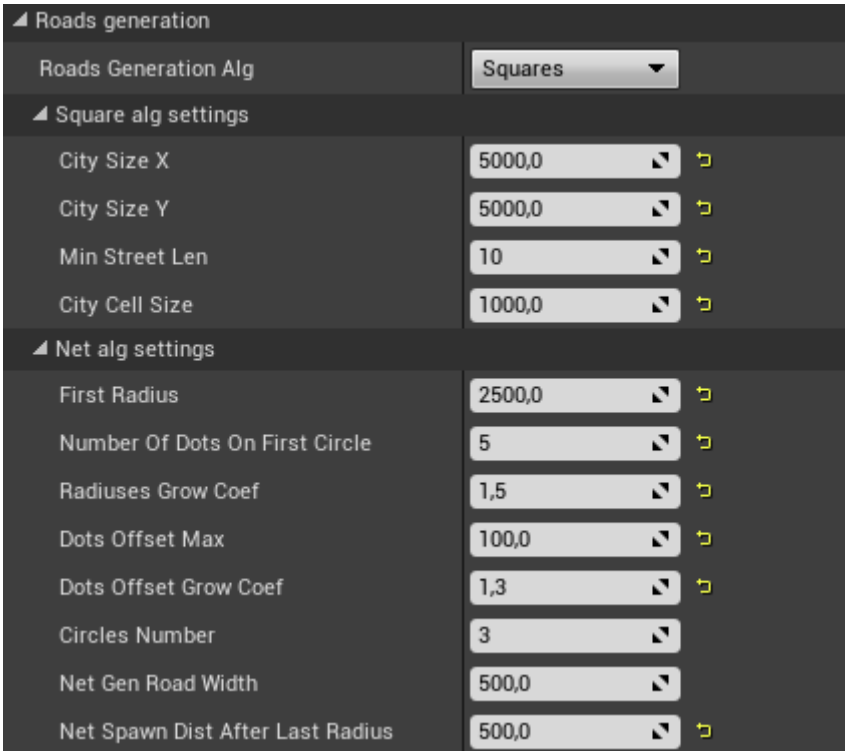


Рисунок 5.22 – Налаштування генерації сітки доріг.

Перше налаштування визначає алгоритм генерації доріг – net або squares (павутина або ділення прямокутників). Далі йдуть налаштування цих двох алгоритмів.

Група налаштувань Square alg settings включає в себе розміри міста, розмір клітинки міста (дорівнює розміру тайла дороги та мінімальну довжину вулиці (завдяки їй алгоритм знає коли закінчити ділення прямокутників).

Net alg settings включає в себе (налаштування описані по порядку згідно з рисунком 5.22):

- радіус першого кола міста;
- кількість точок на кожному колі;
- коефіцієнт зміни радіусу;
- максимальний відступ кожної точки.

- коефіцієнт зміни попереднього параметра для кожного наступного кола;
- кількість концентричних кіл в місті;
- ширина дороги (дорівнює розміру тайлу дороги);
- дистанція від останнього кола міста, на якій ще можна розміщувати будинки (околиці міста).

Кнопки, що керують генерацією розміщені в категорії defaults налаштувань актора-контролера генерації.

Рисунок 5.23 – Керування генерацією

- cleaning the scene – очищує сцену від акторів, які були згенеровані даним контролером;
- start generation – запускає генерацію міста.

Достатньо лише двух кнопок – генерація міста та очищення сцени. Редагування окремих елементів генерації користувач може проводити вже на сцені, обираючи окремий актор та змінюючи його налаштування.

5.2. Випробування програмного продукту

Основні положення, які підлягають перевірці – це коректність генерації міста при введених в контролері генерації налаштуваннях та можливість пере-генерувати окремі елементи міста. Відповідно, треба створити наступні групи тестів

- коректність генерації міста при різних комбінаціях вхідних налаштувань в акторі-контролері генерації;
- можливість пере-генерувати будинок, парк;
- можливість згенерувати будинок, парк та карту доріг окремо від міста;
- коректність розміщення будинків в місті;

- коректність генерації та розміщення парків, його елементів.

5.2.1. Мета випробувань

Метою випробувань являється перевірка відповідності функцій системи процедурної генерації міста вимогам технічного завдання.

5.2.2. Загальні положення

Випробування проводяться на основі наступних документів:

– ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;

– ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

5.2.3. Результати випробувань

В таблиці 5.1 наведені тест-кейзи генерації будинку

Таблиця 5.1 – Тестування генерації будинку

Ситуація	Очікуваний ефект
Введення від’ємних значень ширини, висоти, довжини	Будинок не згенерувався, маємо пустий актор (так як не відбулося жодної ітерації в циклах)
Нульові значення ширини, довжини або висоти	Будинок не згенерувався
Значення висоти, довжини та ширини більші за нуль	Будинок згенерувався відповідно до заданих налаштувань
Виставлення аномального куту повороту для актору	Будинок та всі блоки повернулися правильно
Встановлення параметра розміру блоків більшим, ніж реальний розмір блоків	Блоки згенеровані на відстані одне від одного

Продовження таблиці 5.1

Ситуація	Очікуваний ефект
Встановлення параметра розміру блоків меншим, ніж реальний розмір блоків	Блоки входять одне в одного
Нульовий розмір блоку	Всі блоки розмістились в одній точці
Встановлення випадкових моделей в параметри, замість блоків будинку	Будуть розміщені вказані моделі замість коректних блоків, але генерація залишається коректною
Залишити деякі параметри пустими	Відповідні компоненти будинку будуть пустими, але всі інші з’являться на сцені
Встановлення параметра розміру блоків меншим, ніж реальний розмір блоків	Блоки входять одне в одного

В таблиці 5.2 наведені тест-кейзи для генератора парку

Таблиця 5.2 – Тестування генерації парку та розміщення його компонентів

Ситуація	Очікуваний ефект
Введення від ємних чисел в поля налаштувань (SizeX, SizeY, RoadMeshFreq, RoadMeshRnOffset, Radius)	Система на дає вводити такі значення
Натискання на кнопку «Generate park»	Парк згенерувався відповідно до заданих налаштувань (попередньо згенеровані елементи були видалені)
Натискання на кнопку «Clean park»	Видалення всіх компонентів парку зі сцени

Продовження таблиці 5.2

Ситуація	Очікуваний ефект
Введення 0 в SizeX / SizeY	Парк не буде генеруватись (відповідні цикли будуть мати 0 ітерацій)
Введення 0 в RoadMeshFreq	Система не дозволяє вводити в це поле параметри менші за 50 (якщо модель буде розміщуватись через кожні 0 одиниць відстані, програма зависне).
Введення 0 в RoadMeshRnOffset	Паркові дороги будуть ідеально прямі
Залишити пусті посилання на моделі для парку	Парк згенерується, але відповідні елементи будуть відсутні
Залишити пусті посилання на моделі рослинності	Парк згенерується, але відповідні елементи рослинності не розмістяться
Введення 0 в Radius	Парк згенерується, але альтанки можуть розміщуватись занадто близько одне до одного
Введення від ємних чисел в поля налаштувань (SizeX, SizeY, RoadMeshFreq, RoadMeshRnOffset, Radius)	Система на дає вводити такі значення
Натискання на кнопку «Generate park»	Парк згенерувався відповідно до заданих налаштувань (попередньо згенеровані елементи були видалені)
Натискання на кнопку «Clean park»	Видалення всіх компонентів парку зі сцени

В таблиці 5.3 наведені тест-кейзи для алгоритму генерації доріг методом ділення на прямокутники.

Таблиця 5.3 – Тестування генерації доріг методом ділення на прямокутники

Ситуація	Очікуваний ефект
Введення відємних значень для SizeX, SizeY, Freq, MinStreetLen	Система не дає вводити такі значення
Введення 0 для SizeX, SizeY	Генерація не відбудеться
Введення 0 для Freq	Поле автоматично замінюється на 10 (для уникнення безкінечного циклу розміщення елементів доріг)
Введеня 0 для MinStreetLen	Поле автоматично замінюється на 1 (для уникнення безкінечного циклу ділення прямокутників)
Залишити пустими посилання на моделі	Відповідні елементи не будуть розміщені, але генерація відбудеться. Якщо не були вказані посилання на елементи дороги, буде відсутнє розділення кварталів (генератор не бачить, де закінчується квартал)
Вставити посилання на некоректні моделі	Генерація відбудеться, але з некоректними моделями

В таблиці 5.4 наведені тест-кейзи для алгоритму генерації павутиноподібних доріг.

Таблиця 5.4 – Тестування генерації доріг методом створення павутини доріг

Ситуація	Очікуваний ефект
Введення від’ємних значень для всіх параметрів	Система не дає вводити такі значення
Введення 0 для CirclesNumber	Генерація не відбудеться
Введення 0 або іншого занадто малого значення для FirstRadius	Генерація не відбудеться
Введеня RadiusProgressKoef менше за 1	Поле автоматично замінюється на 1
Введеня OffsetGrowK менше за 1	Поле автоматично замінюється на 1
Введення SegmentsNumber 0	Поле автоматично замінюється на 3 (мінімальна кількість точок на колі щоб створити сегмент)
StartDotOffset 0	Створюються симетричні дороги, без жодних відступів
Передати посилання на моделі-бордюри, які не підходять для сплайн-диформації	Генерація та сплайн деформація відбудеться, але зовнішній вигляд деформованих моделей може бути поганим

В таблиці 5.5 наведені тест-кейзи для контролера генерації. Так як багато його налаштувань напряму передаються попередньо описаним акторам, було наведені тести лише для унікальних властивостей цього актору.

Таблиця 5.5 – Тестування загальної генерації міста

Ситуація	Очікуваний ефект
Встановлення пустого посилання на HeightRule	Висота кожного згенерованого будинку буде 2
Встановлення BuildingsOffset менше за 0	Виправляється на 0
Обрати алгоритм генерації доріг - Squares	При натисканні на «Generate» відбудеться генерація міста з прямокутною картою доріг
Обрати алгоритм генерації доріг - Network	При натисканні на «Generate» відбудеться генерація міста з павутиноподібною картою доріг
Введення кількості парків менше за 0	Автоматично замінюється на 0
Введення кількості парків 0	Всі квартали будуть з будинками
Натискання «StartGeneration»	Проходить генерація міста згідно даних налаштувань
Натискання «CleaningTheScene»	Очищення всіх акторів генерації зі сцени

Висновок до розділу

В даному розділі було описано функціонал та приведені скріншоти результатів процедурної генерації міста. Були описані налаштування генерації (на що вони впливають та як їх задавати) та приведені зображення екранних форм для цих налаштувань. Був наведений опис кожного реалізованого актора процедурної генерації.

В підрозділі «Випробування програмного продукту» були наведені детальні таблиці тест-кейзів, які покривають функціонал кожного елемента генерації:

- контролер генерації;
- генератор будинку;
- генератор доріг та декоративних елементів міста;
- заповнення міста будинками та парками;
- генератор парку.

ЗАГАЛЬНИЙ ВИСНОВОК

У дипломній роботі були розглянуті основні процеси комплексу задачі інформаційної підтримки генерації міпроцеси комплексу задачі інформаційної підтримки генерації міста на 3D-сцені.

В розділі загальних положень було описано предметне середовище та перспективи розробки програмних засобів для сфери розробки відео-ігр. Був описаний процес діяльності при ручному створенні ігрового рівня без процедурної генерації. Були визначені основні актори системи, їх функції. Були визначені основні групи користувачів та створені функціональні вимоги дл системи. Було проведено дослідження аналогів та наведений детальний порівняльний аналіз даної роботи з аналогами. Були визначені мета, задачі та цілі розробки.

В розділі інформаційного забезпечення були визначені вхідні дані до системи генерації міста (користувацькі налаштування) та вихідні дані (розміщення акторів на сцені, сам файл сцени). Була детально описана структура ігрового рівня.

В розділі математичного забезпечення були поставлені основні задачі, які треба вирішити, та обрані алгоритми:

- алгоритм ділення прямкутників для генерації прямокутної карти доріг;
- алгоритм створення павутини для павутиноподібної карти доріг;
- алгоритм розміщення будинків "до сусіді" для заповнення кварталів будинками.

Було наведено детальне описання даних алгоритмів, псевдокод та обгрунтовано їх вибір.

У розділі програмного та технічного забезпечення описані технології, які використані в програмному продукті. Обґрунтовуються технології та їх особливості, які використовуються в даній розробці. В якості основного

оточення для розробки був обраний ігровий редактор Unreal Engine 4. В якості оточення для написання коду - Visual Studio 2017. Обрані мови програмування - Unreal Visual Blueprints (надають можливість швидкого написання коду) та C++ (дозволяє реалізовувати трудоємкі алгоритми завдяки детальним можливостям по роботі з памяттяю). Була описана архітектура розробки та наведена детальна специфікація функцій.

У технологічному розділі було наведено керівництво користувача в якому детально описуються всі налаштування генерації та наводяться приклади згенерованого міста та його елементів. Описані методи випрубуваннь та наведені тест-кейзи.

Даний програмний продукт спрощує процесс створення великих ігрових 3D-сцен міського типу завдяка автоматичному розміщенню всіх елементів міста згідно заданих налаштувань. Користувач (розробник на UE4) також має можливість передавати власні 3D-моделі до системи генерації як параметр та модифікувати її код.

ПЕРЕЛІК ПОСИЛАНЬ

1) Game engine [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Game_engine.

2) Unreal Engine 4 [Електронний ресурс] – Режим доступу до ресурсу: https://www.unrealengine.com/en-US/.

3) Unreal Marketplace [Електронний ресурс] – Режим доступу до ресурсу: https://www.unrealengine.com/marketplace/en-US/store.

4) Procedural generator of the highways and roads by Andry K [Електронний ресурс] – Режим доступу до ресурсу: https://www.unrealengine.com/marketplace/en-US/slug/procedural-generator-of-the-highways-and-roads.

5) Procedural Building Lot by Ammobox Studios [Електронний ресурс] – Режим доступу до ресурсу: https://www.unrealengine.com/marketplace/en-US/slug/procedural-building-lot.

6) Procedural cities with interiors [Електронний ресурс] – Режим доступу до ресурсу: https://forums.unrealengine.com/community/released-projects/1370283-procedural-cities-with-interiors-free.

7) Архитектурная визуализация в Unreal Engine 4 [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: https://habr.com/ru/post/253503/.

8) User Interfaces & HUDs [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unrealengine.com/en-US/Gameplay/Framework/UIAndHUD.

9) Unreal Engine 4 documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unrealengine.com/en-us.

10) Level Blueprint [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unrealengine.com/en-us/Engine/Blueprints/UserGuide/Types/LevelBlueprint

11) AActor documentation [Електронний ресурс] – Режим доступу до ресурсу: https://api.unrealengine.com/INT/API/Runtime/Engine/GameFramework/AActor/index.html.

12) C++ documentation [Електронний ресурс] – Режим доступу до ресурсу: https://devdocs.io/cpp/

13) Blueprints Visual Scripting [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unrealengine.com/en-us/Engine/Blueprints.

Додаток А

Тексти програмного коду

Інформаційна підтримка генерації міста на 3D-сцені

(Найменування програми (документа))

DVD-R

(Вид носія даних)

9 арк, 244 Кб

(Обсяг програми (документа) , арк.,) Кб)

Київ – 2019 року

LIB_GlobalMethods.cpp

```
#include "Lib_GlobalMethods.h"
#include "SquareDataStruct.h"
TArray<FSquareDataStruct> CitySquares;
TArray<FVector> ULib_GlobalMethods::GenerateRoadMatrix(int32 GridSizeX, int32 GridSizeY, int32 MinStreetLen, float DivideProbability)
{
CitySquares.Reset(0);
/*Initialize road array with zeros*/
int** roadsArr = new int*[GridSizeX];
for (int i = 0; i < GridSizeX; ++i)
	roadsArr[i] = new int[GridSizeY];
for (int i = 0; i < GridSizeX; i++)
{
	for (int j = 0; j < GridSizeY; j++)
	{
		roadsArr[i][j] = 0;
	}
}
DivideSquare(0, 0, GridSizeX - 1, GridSizeY - 1, roadsArr, 0, 9999, MinStreetLen, DivideProbability);
for (int i = 0; i < GridSizeX; i++)
{
	for (int j = 0; j < GridSizeY; j++)
	{
		/*checking edges*/
		if (i == 0 || i == (GridSizeX - 1))
		{
			roadsArr[i][j] = 4;
		}
		else if (j == 0 || j == (GridSizeY - 1))
		{
			roadsArr[i][j] = 3;
		}
	}
}
for (int i = 1; i < GridSizeX-1; i++)
{
	for (int j = 1; j < GridSizeY-1; j++)
	{
		if (roadsArr[i][j] == 0) continue; //empty (no road piece here)
			/*Testing for crossroad in center*/
			int neiborRoadsCounter = 0;
			if (roadsArr[i - 1][j] != 0)
				neiborRoadsCounter++;
			if (roadsArr[i + 1][j] != 0)
				neiborRoadsCounter++;
			if (roadsArr[i][j - 1] != 0)
				neiborRoadsCounter++;
			if (roadsArr[i][j + 1] != 0)
```

```
                                neiborRoadsCounter++;
                        if (roadsArr[i - 1][j] != 0 && roadsArr[i + 1][j] != 0)
                                {
                                        roadsArr[i][j] = 3;
                                }
                                else if (roadsArr[i][j - 1] != 0 && roadsArr[i][j + 1] != 0)
                                {
                                        roadsArr[i][j] = 4;
                                }
                }
        }
        for (int i = 0; i < CitySquares.Num(); i++)
        {
                roadsArr[CitySquares[i].X_Max][CitySquares[i].Y_Max] = 2;
                roadsArr[CitySquares[i].X_Max][CitySquares[i].Y_Min] = 2;
                roadsArr[CitySquares[i].X_Min][CitySquares[i].Y_Max] = 2;
                roadsArr[CitySquares[i].X_Min][CitySquares[i].Y_Min] = 2;
        }
        TArray<FVector> arr;
        for (int i = 0; i < GridSizeX; i++)
        {
                for (int j = 0; j < GridSizeY ; j++)
                {
                        FVector val = FVector(i,j,roadsArr[i][j]);
                        arr.Add(val);
                }
        }
        return arr;

        TArray<FSquareDataStruct> ULib_GlobalMethods::GetCitySquaresLastGenerated()
        {
        return CitySquares;

        void ULib_GlobalMethods::DivideSquare(int32 x_Start, int32 y_Start, int32 x_End,
int32 y_End, int32 **twoDimArray, int32 recStep, int32 recStepMax, int32 MinStreetLen, float
DivideProb)
        {
        if ((x_End - x_Start) <= MinStreetLen*2 || (y_End - y_Start) <= MinStreetLen*2 ||
recStep >= recStepMax)
        {
                FSquareDataStruct locSquares;
                locSquares.X_Max = x_End;
                locSquares.X_Min = x_Start;
                locSquares.Y_Max = y_End;
                locSquares.Y_Min = y_Start;
                CitySquares.Add(locSquares);
                return;
        }
        float probTest = FMath::RandRange(0.0f, 1.0f);
        if (probTest > DivideProb)
        {
```

```
		FSquareDataStruct locSquares;
		locSquares.X_Max = x_End;
		locSquares.X_Min = x_Start;
		locSquares.Y_Max = y_End;
		locSquares.Y_Min = y_Start;
		CitySquares.Add(locSquares);

		return;
	}
	float offset = 0.35f;
	int x_Middle = (x_End + x_Start) / 2 + FMath::RandRange(MinStreetLen*(-1)*offset, MinStreetLen*offset);
	int y_Middle = (y_End + y_Start) / 2 + FMath::RandRange(MinStreetLen*(-1)*offset, MinStreetLen*offset);
	int StrLen = MinStreetLen;
	for (int i = x_Start; i < x_End; i++)
	{
		twoDimArray[i][y_Middle] = 1;
	}

	for (int i = y_Start; i < y_End; i++)
	{
		twoDimArray[x_Middle][i] = 1;
	}
	int32 currentRecStep = recStep + 1;
	DivideSquare(x_Start, y_Start, x_Middle, y_Middle, twoDimArray, currentRecStep, recStepMax,StrLen, DivideProb);
	DivideSquare(x_Middle, y_Start, x_End, y_Middle, twoDimArray, currentRecStep, recStepMax, StrLen, DivideProb);
	DivideSquare(x_Start, y_Middle, x_Middle, y_End, twoDimArray, currentRecStep, recStepMax, StrLen, DivideProb);
	DivideSquare(x_Middle, y_Middle, x_End, y_End, twoDimArray, currentRecStep, recStepMax, StrLen, DivideProb);

	TArray<FVector> ULib_GlobalMethods::GenerateDotsOnCircle(float CurrentRadius, int32 CurrentCircle, int32 DotsNumber, float DotsOffset, float DotsOffsetGrowProgC, AActor * RoadGenerator)
	{
	TArray<FVector> arr;
	for (int i = 0; i < DotsNumber; i++)
	{
		float an360 = 360;
		float segments = DotsNumber;
		float currAngle = i * (an360 / segments);
		FVector ZAxisRotatedVector = RoadGenerator->GetActorForwardVector().RotateAngleAxis(currAngle, RoadGenerator->GetActorUpVector());
		FVector NonOffsetDot = ZAxisRotatedVector * CurrentRadius + RoadGenerator->GetActorLocation();
		float offsetBase = DotsOffset * (CurrentCircle * DotsOffsetGrowProgC) + DotsOffset;
```

```
			float xOffset = FMath::FRandRange(-1 * offsetBase, offsetBase);
			float yOffset = FMath::FRandRange(-1 * offsetBase, offsetBase);
			NonOffsetDot = NonOffsetDot + FVector(xOffset, yOffset, 0);
			arr.Add(NonOffsetDot);
		}

		return arr;
		}

		int32 ULib_GlobalMethods::FindClosestVector(TArray<FVector> Vectors, FVector Point)
		{
		int32 resultIndex = 0;
		float MAX = 9999999;
		for (int i = 0; i < Vectors.Num(); i++)
		{
			FVector localVector = Vectors[i] - Point;
			float len = FMath::Sqrt(localVector.X * localVector.X + localVector.Y * localVector.Y + localVector.Z * localVector.Z);
			if (len < MAX)
			{
				MAX = len;
				resultIndex = i;
			}
		}
		return resultIndex;
		}

		TArray<FVector> ULib_GlobalMethods::DotsConnection_CPlus(TArray<FVector> ArrIn_01, TArray<FVector> ArrIn_02)
		{
		TArray<FVector> FirstArr = ArrIn_01;
		TArray<FVector> SecondArr = ArrIn_02;

		TArray<FVector> PareOut_01;
		TArray<FVector> PareOut_02;

		for (int i = 0; i < FirstArr.Num(); i++)
		{
			int32 res = FindClosestVector(SecondArr, FirstArr[i]);
			PareOut_01.Add(FirstArr[i]);
			PareOut_02.Add(SecondArr[res]);
			SecondArr.RemoveAt(res);
		}

		for (int i = 0; i < PareOut_02.Num(); i++)
		{
			PareOut_01.Add(PareOut_02[i]);
		}
		return PareOut_01;
```

```
}

TArray<FVector> ULib_GlobalMethods::MiddleDotsGeneration_CPlus(int32 yMin, int32 xMin, int32 xDivide, int32 yDivide, int32 xMax, int32 yMax, float sizeX, float sizeY, AActor* ParkGenerator)
{

TArray<FVector> resultVectors;
for (int i = xMin; i < xMax; i++)
{
	for (int k = yMin; k < yMax; k++)
	{

	   float xLerpAlpha = (float)i / (float)xDivide;
		float yLerpAlpha = (float)k / (float)yDivide;

		float xValue = FMath::LerpStable((float)0, sizeX, xLerpAlpha);
		float yValue = FMath::LerpStable((float)0, sizeY, yLerpAlpha);

		FVector RightPart = (xValue - (sizeX / 2)) *ParkGenerator->GetActorRightVector();

		FVector ForwardPart = (yValue - (sizeY / 2)) *ParkGenerator->GetActorForwardVector();

		resultVectors.Add(RightPart + ForwardPart + ParkGenerator->GetActorLocation());
	}

}
return resultVectors;
}

TArray<FVector> ULib_GlobalMethods::AddSideDotsBetween_CPlus(FVector A, FVector B, float freq)
{
TArray<FVector> resultArray;
FVector diff = A - B;
float VectorLen = FMath::Sqrt(diff.X * diff.X + diff.Y * diff.Y + diff.Z * diff.Z);
int count = VectorLen / freq;

for (int i = 1; i < count; i++)
{
	float lerpAlpha = (float)i * freq / VectorLen;

	FVector newDot = FMath::Lerp(A, B, lerpAlpha);
	resultArray.Add(newDot);
}
return resultArray;
}
```

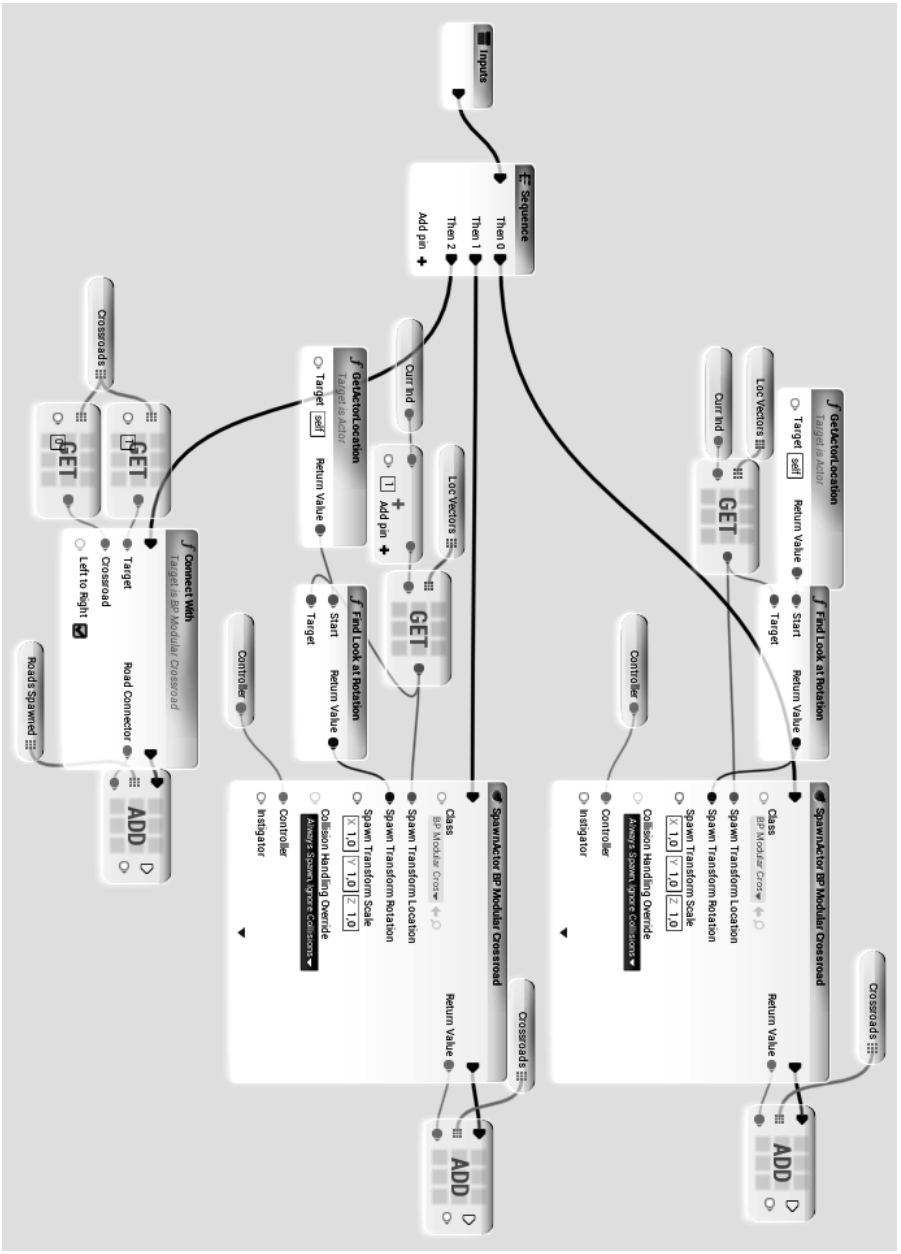
```
TArray<FVector>
ULib_GlobalMethods::FindFirstDotLongerThenRadius_CPlus(TArray<FVector>  DotsArray,
FVector Dot, float Radius)
{
TArray<FVector> LocalDotsArray = DotsArray;
TArray<FVector> ItemsToRemove;

for (int i = 0; i < LocalDotsArray.Num(); i++)
{
	FVector len = LocalDotsArray[i] - Dot;
	float f_Len = len.Size();

	if (f_Len <= Radius)
	{
		ItemsToRemove.Add(LocalDotsArray[i]);
	}
}

for (int i = 0; i < ItemsToRemove.Num(); i++)
{
	if ((ItemsToRemove[i] - Dot).Size() > 5)
	{

		for (int k = 0; k < LocalDotsArray.Num(); k++)
		{
			if ((LocalDotsArray[k] - ItemsToRemove[i]).Size() < 5)
			{
				LocalDotsArray.RemoveAt(k);
				break;
			}
		}
	}
}
return LocalDotsArray;
}

TArray<FVector>  ULib_GlobalMethods::SelectDotsByRules_CPlus(int32  DotsNumber,
TArray<FVector> DotsIn, float Radius)
{
TArray<FVector> LocalVectors = DotsIn;
TArray<FVector> FinalVectors;
for (int i = 0; i < DotsNumber; i++)
{
	LocalVectors = ShuffleVectorsArray(LocalVectors, 15);
	FinalVectors.Add(LocalVectors[0]);
	LocalVectors	=	FindFirstDotLongerThenRadius_CPlus(LocalVectors,
LocalVectors[0], Radius);
}
return FinalVectors;
}
```

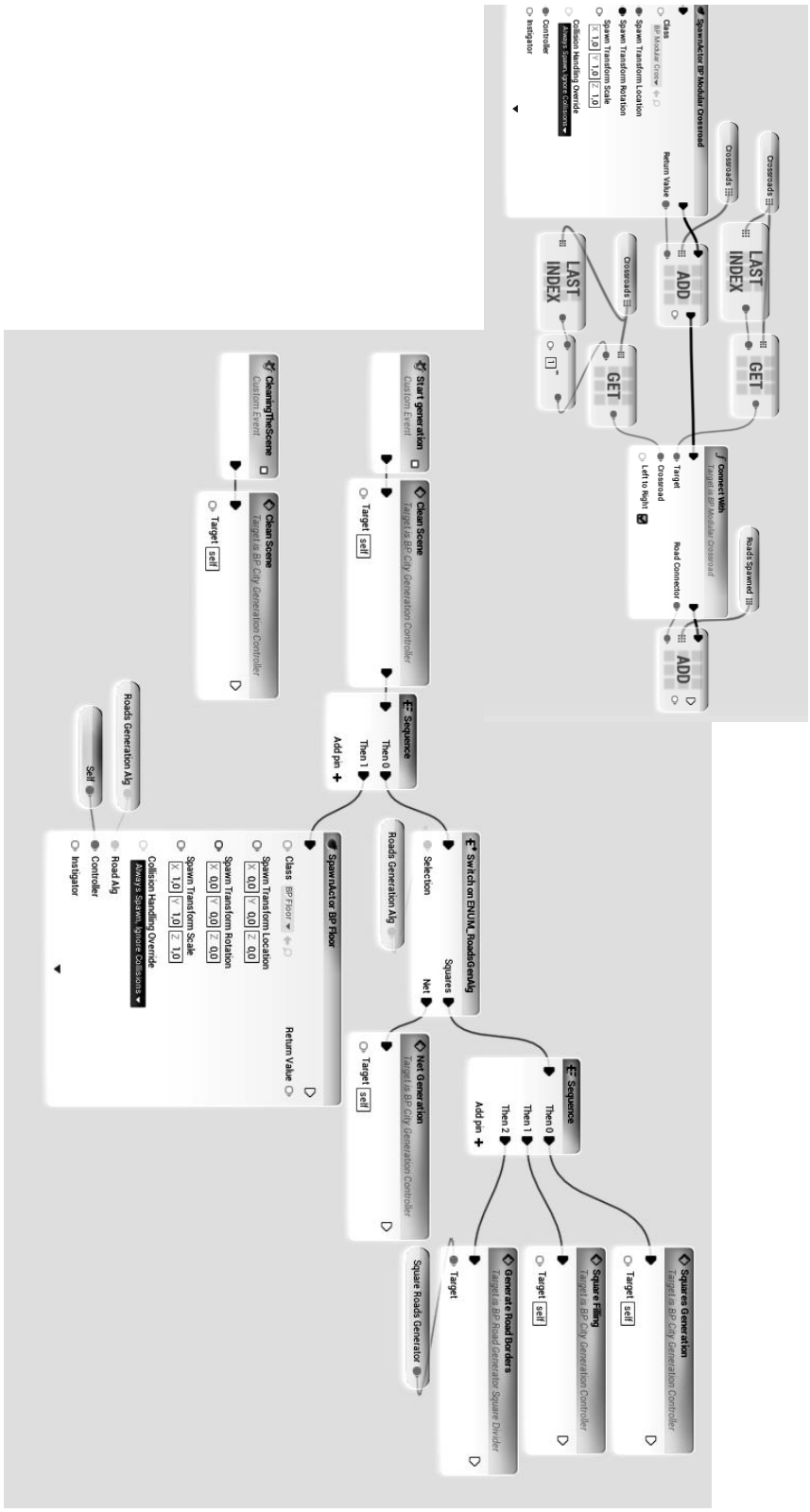
```
TArray<FVector> ULib_GlobalMethods::ShuffleVectorsArray(TArray<FVector> InArray, int32 Iterations)
{
TArray<FVector> resultArray = InArray;
for (int i = 0; i < Iterations; i++)
{
	int32 Index_01 = FMath::RandRange(0, InArray.Num() - 1);
	int32 Index_02 = FMath::RandRange(0, InArray.Num() - 1);

	FVector local = resultArray[Index_01];
	resultArray[Index_01] = resultArray[Index_02];
	resultArray[Index_02] = local;
}
return resultArray;
}

TArray<FVector> ULib_GlobalMethods::CreateSideDots_CPlus(float SideDotsRoadFreq, AActor* ParkGenerator, float sizeX, float sizeY)
{
FVector Forward = ParkGenerator->GetActorForwardVector() * (sizeY / 2);
FVector Back = Forward.RotateAngleAxis(180, ParkGenerator->GetActorUpVector());
FVector Right = ParkGenerator->GetActorRightVector() * (sizeX / 2);
FVector Left = Right.RotateAngleAxis(180, ParkGenerator->GetActorUpVector());

FVector FL_Corner = Forward + Left + ParkGenerator->GetActorLocation();
FVector FR_Corner = Right + Forward + ParkGenerator->GetActorLocation();
FVector BL_Corner = Back + Left + ParkGenerator->GetActorLocation();
FVector BR_Corner = Back + Right + ParkGenerator->GetActorLocation();
TArray<FVector> ResultVectors;
TArray<FVector> LocalResult;

LocalResult = AddSideDotsBetween_CPlus(FR_Corner, FL_Corner, SideDotsRoadFreq);
ResultVectors.Append(LocalResult);

LocalResult = AddSideDotsBetween_CPlus(BL_Corner, BR_Corner, SideDotsRoadFreq);
ResultVectors.Append(LocalResult);

LocalResult = AddSideDotsBetween_CPlus(FR_Corner, BR_Corner, SideDotsRoadFreq);
ResultVectors.Append(LocalResult);

LocalResult = AddSideDotsBetween_CPlus(FL_Corner, BL_Corner, SideDotsRoadFreq);
ResultVectors.Append(LocalResult);

return ResultVectors;
}
```

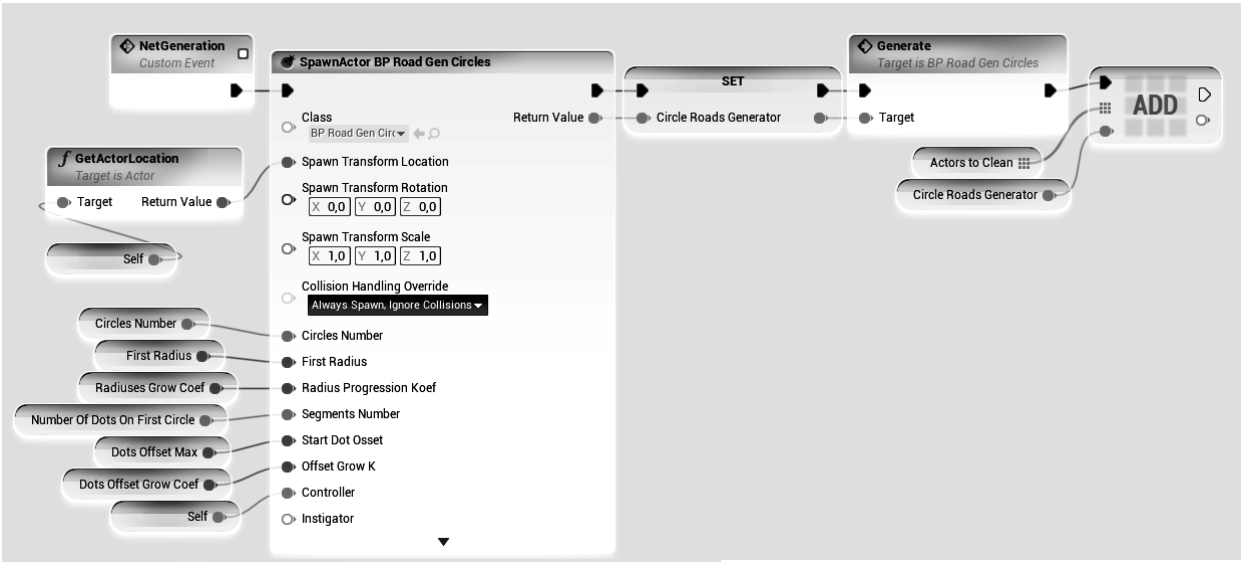
LIB_GlobalMethods.h

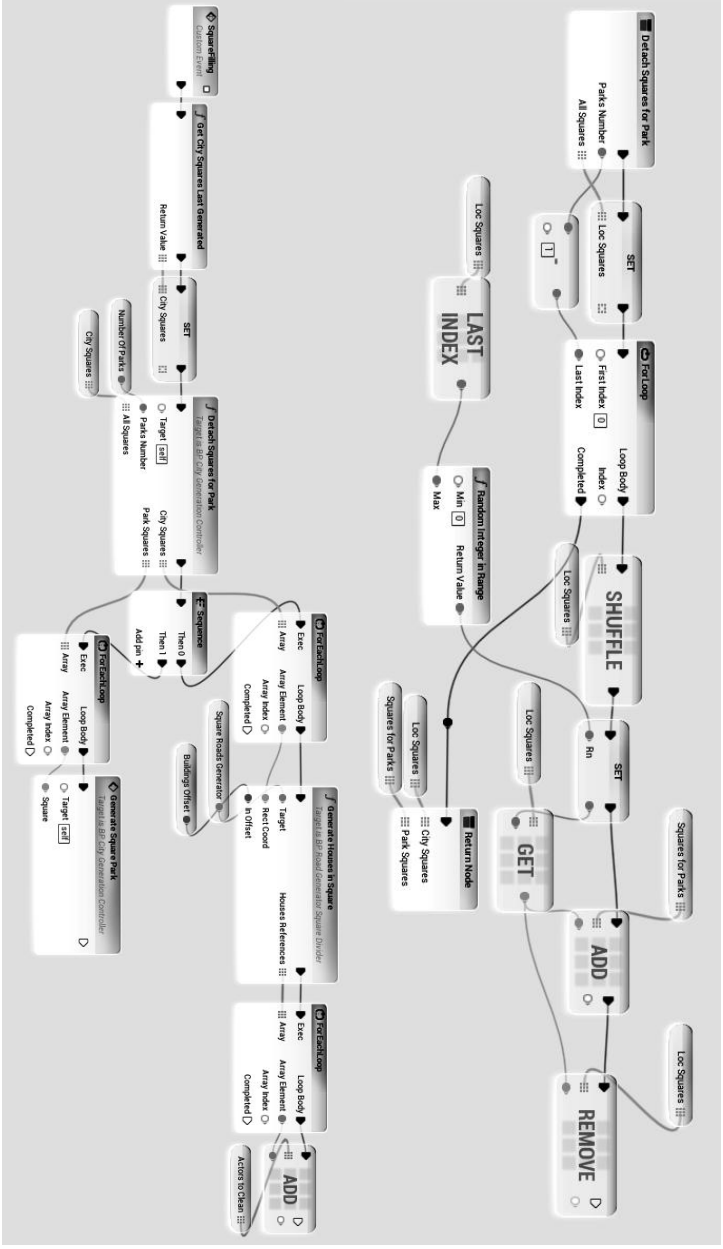
```
#pragma once
#include "CoreMinimal.h"
#include "Kismet/BlueprintFunctionLibrary.h"
#include "SquareDataStruct.h"
#include "Lib_GlobalMethods.generated.h
UCLASS()
class DIPLOMACITYGENERATOR_API ULib_GlobalMethods : public UBlueprintFunctionLibrary
{
GENERATED_BODY()
static void DivideSquare(int32 x_Start, int32 y_Start, int32 x_End, int32 y_End, int32 **twoDimArray, int32 recStep, int32 recStepMax, int32 MinStreetLen, float DivideProb);

    UFUNCTION(BlueprintCallable)
static TArray<FVector> GenerateRoadMatrix(int32 GridSizeX, int32 GridSizeY, int32 MinStreetLen, float DivideProbabylity);

    UFUNCTION(BlueprintCallable)
static TArray<FSquareDataStruct> GetCitySquaresLastGenerated();

    UFUNCTION(BlueprintCallable)
static TArray<FVector> GenerateDotsOnCircle(float CurrentRadius, int32 CurrentCircle, int32 DotsNumber, float DotsOffset, float DotsOffsetGrowProgC, AActor *RoadGenerator);

    UFUNCTION(BlueprintCallable)
static int32 FindClosestVector(TArray<FVector> Vectors, FVector Point);

    UFUNCTION(BlueprintCallable)
static TArray<FVector> DotsConnection_CPlus(TArray<FVector> ArrIn_01, TArray<FVector> ArrIn_02);

    UFUNCTION(BlueprintCallable)
static TArray<FVector> MiddleDotsGeneration_CPlus(int32 yMin, int32 xMin, int32 xDivide, int32 yDivide, int32 xMax, int32 yMax, float sizeX, float sizeY, AActor* ParkGenerator);

    UFUNCTION(BlueprintCallable)
static TArray<FVector> AddSideDotsBetween_CPlus(FVector A, FVector B, float freq);

    UFUNCTION(BlueprintCallable)
static TArray<FVector> FindFirstDotLongerThenRadius_CPlus(TArray<FVector> DotsArray, FVector Dot, float Radius);

    UFUNCTION(BlueprintCallable)
static TArray<FVector> SelectDotsByRules_CPlus(int32 DotsNumber, TArray<FVector> DotsIn, float Radius);

    UFUNCTION(BlueprintCallable)
static TArray<FVector> ShuffleVectorsArray(TArray<FVector> InArray, int32 Iterations);
```

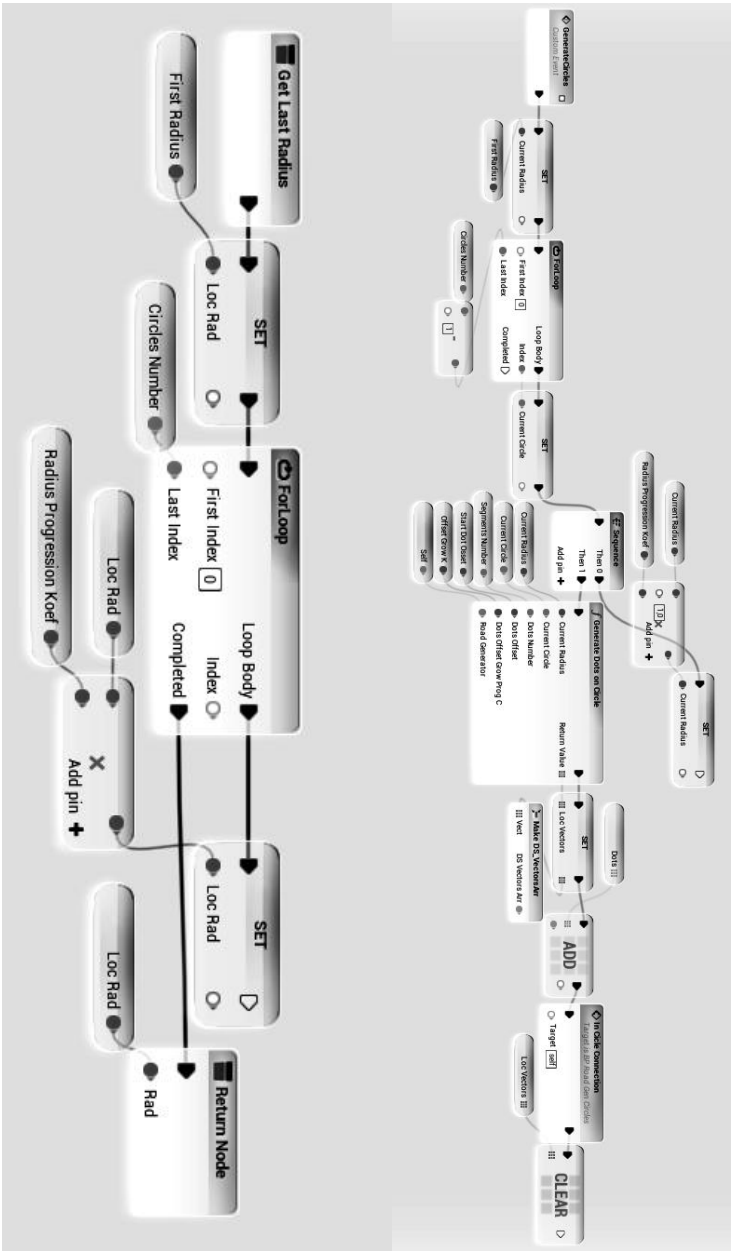
```
        UFUNCTION(BlueprintCallable)
    static TArray<FVector> CreateSideDots_CPlus(float SideDotsRoadFreq, AActor*
ParkGenerator, float sizeX, float sizeY);
    };

    SquareDataStruct.h
    #pragma once
    #include "Engine.h"
    #include "CoreMinimal.h"
    #include "SquareDataStruct.generated.h"
    USTRUCT(BlueprintType)
    struct FSquareDataStruct
    {
    GENERATED_BODY()
    public:
    UPROPERTY(BlueprintReadWrite)
        int32 X_Min;
    UPROPERTY(BlueprintReadWrite)
        int32 Y_Min;
    UPROPERTY(BlueprintReadWrite)
        int32 X_Max;
    UPROPERTY(BlueprintReadWrite)
        int32 Y_Max;
    };
```

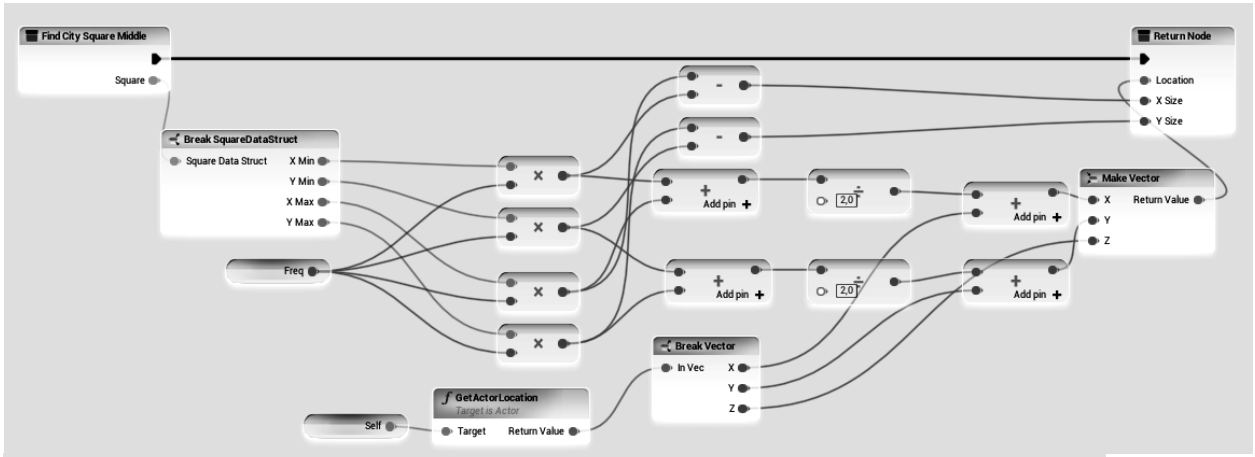

Inputs
Sequence
Then 0
Then 1
Then 2
Add pin
GetActorLocation
Target is Actor
Target self
Return Value
Loc Vectors
Curr Ind
GET
Find Look at Rotation
Start
Target
Return Value
Controller
SpawnActor BP Modular Crossroad
Class
BP Modular Crossroad
Spawn Transform Location
Spawn Transform Rotation
Spawn Transform Scale
X 1,0 Y 1,0 Z 1,0
Collision Handling Override
Always Spawn, Ignore Collisions
Instigator
Crossroads
ADD
Add pin
Connect With
Target is BP Modular Crossroad
Crossroad
Left to Right
Road Connector
Roads Spawned

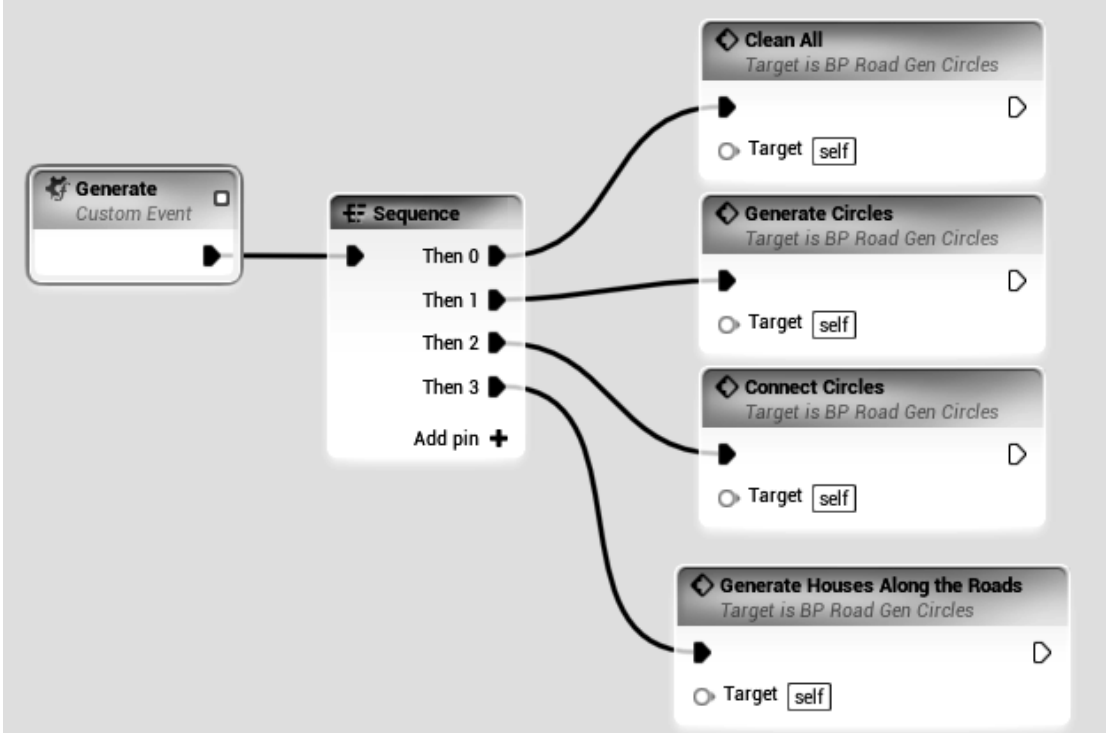

SpawnActor BP_Modular Crossroad
Class
BP_Modular Cross
Spawn Transform Location
Spawn Transform Rotation
Spawn Transform Scale
X 1.0 Y 1.0 Z 1.0
Collision Handling Override
Always Spawn, Ignore Collisions
Controller
Instigator
Return Value
Crossroads
LAST INDEX
ADD
GET
Connect With
Target is BP_Modular Crossroad
Target
Crossroad
Left to Right
Road Connector
Roads Spawned
Start generation
Custom Event
Clean Scene
Target is BP_City Generation Controller
Target
self
CleaningTheScene
Sequence
Then 0
Then 1
Add pin
Roads Generation Alg
Self
SpawnActor BP_Floor
Class
BP Floor
Spawn Transform Location
X 0.0 Y 0.0 Z 0.0
Spawn Transform Rotation
X 0.0 Y 0.0 Z 0.0
Spawn Transform Scale
X 1.0 Y 1.0 Z 1.0
Collision Handling Override
Always Spawn, Ignore Collisions
Road Alg
Controller
Instigator
Return Value
Switch on ENUM_RoadsGenAlg
Selection
Squares
Net
Net Generation
Sequence
Then 2
Squares Generation
Square Filling
Generate Road Borders
Target is BP_Road Generator Square Divider
Square Roads Generator

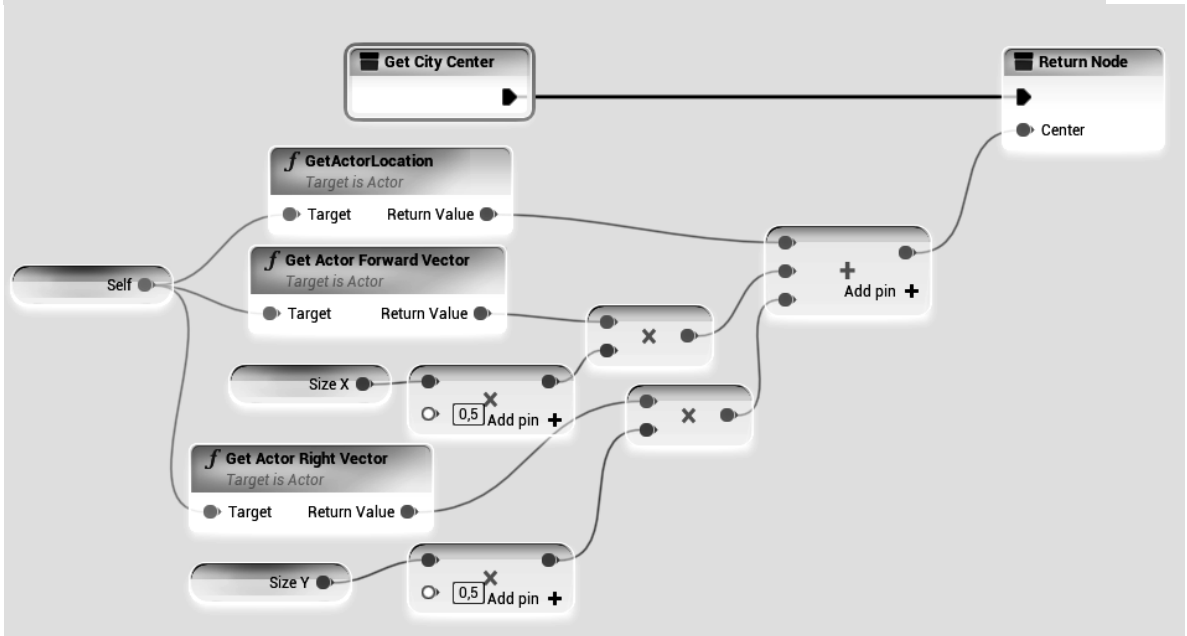

NetGeneration
Custom Event
SpawnActor BP Road Gen Circles
Class
BP Road Gen Circ
Return Value
Spawn Transform Location
Spawn Transform Rotation
X 0,0 Y 0,0 Z 0,0
Spawn Transform Scale
X 1,0 Y 1,0 Z 1,0
Collision Handling Override
Always Spawn, Ignore Collisions
Circles Number
First Radius
Radius Progression Koef
Segments Number
Start Dot Osset
Offset Grow K
Controller
Instigator
SET
Circle Roads Generator
Generate
Target is BP Road Gen Circles
Target
ADD
Actors to Clean
Circle Roads Generator
GetActorLocation
Target is Actor
Target
Return Value
Self
Circles Number
First Radius
Radiuses Grow Coef
Number Of Dots On First Circle
Dots Offset Max
Dots Offset Grow Coef
Self

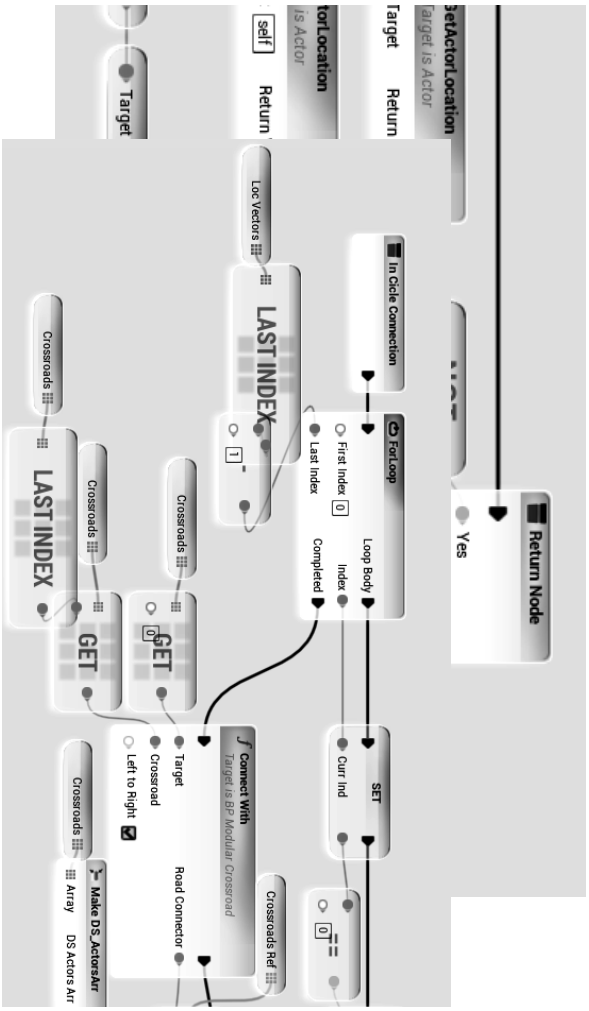

SquareFilling
Custom Event
Get City Squares Last Generated
Return Value
SET
City Squares
Number Of Parks
Detach Squares for Park
Target is BP City Generation Controller
Target
Parks Number
All Squares
City Squares
Park Squares
Sequence
Then 0
Then 1
Add pin
ForEachLoop
Exec
Array
Loop Body
Array Element
Array Index
Completed
Square Roads Generator
Buildings Offset
Square
Generate Square Park
Target is BP City Generation Controller
Target
Rect Coord
In Offset
Generate Houses in Square
Target is BP Road Generator Square Divider
Houses References
Actors to Clean
ADD
Detach Squares for Park
Parks Number
All Squares
Loc Squares
SET
FirstLoop
First Index
Last Index
Loop Body
Index
Completed
LAST INDEX
Random Integer in Range
Min
Max
Return Value
SHUFFLE
Rn
GET
Squares for Parks
Return Node
Park Squares
ADD
REMOVE

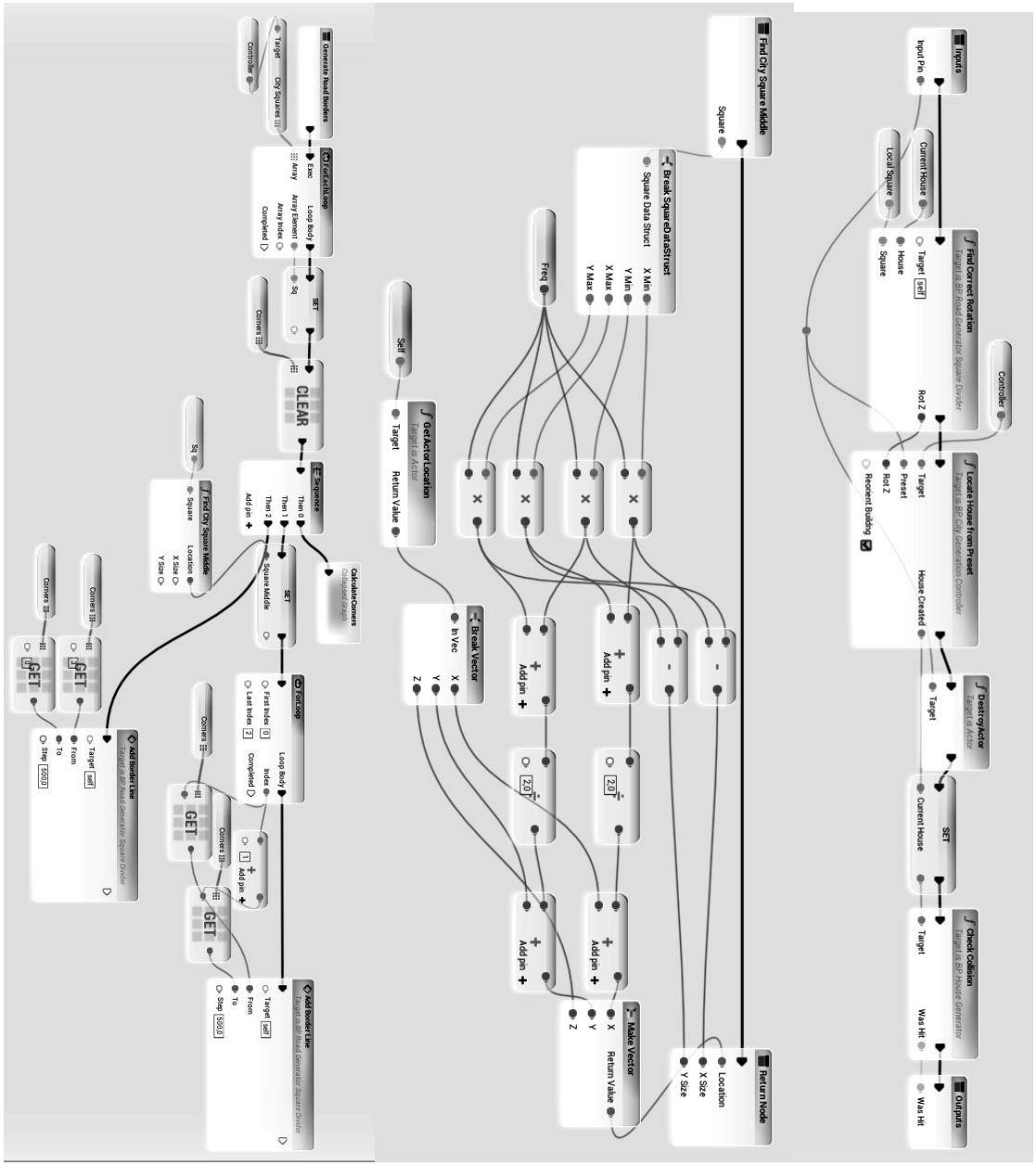

Get Last Radius
First Radius
Loc Rad
SET
Circles Number
ForLoop
First Index
Last Index
Loop Body
Index
Completed
Radius Progression Koef
Add pin
Loc Rad
SET
Loc Rad
Loc Rad
Rad
Return Node

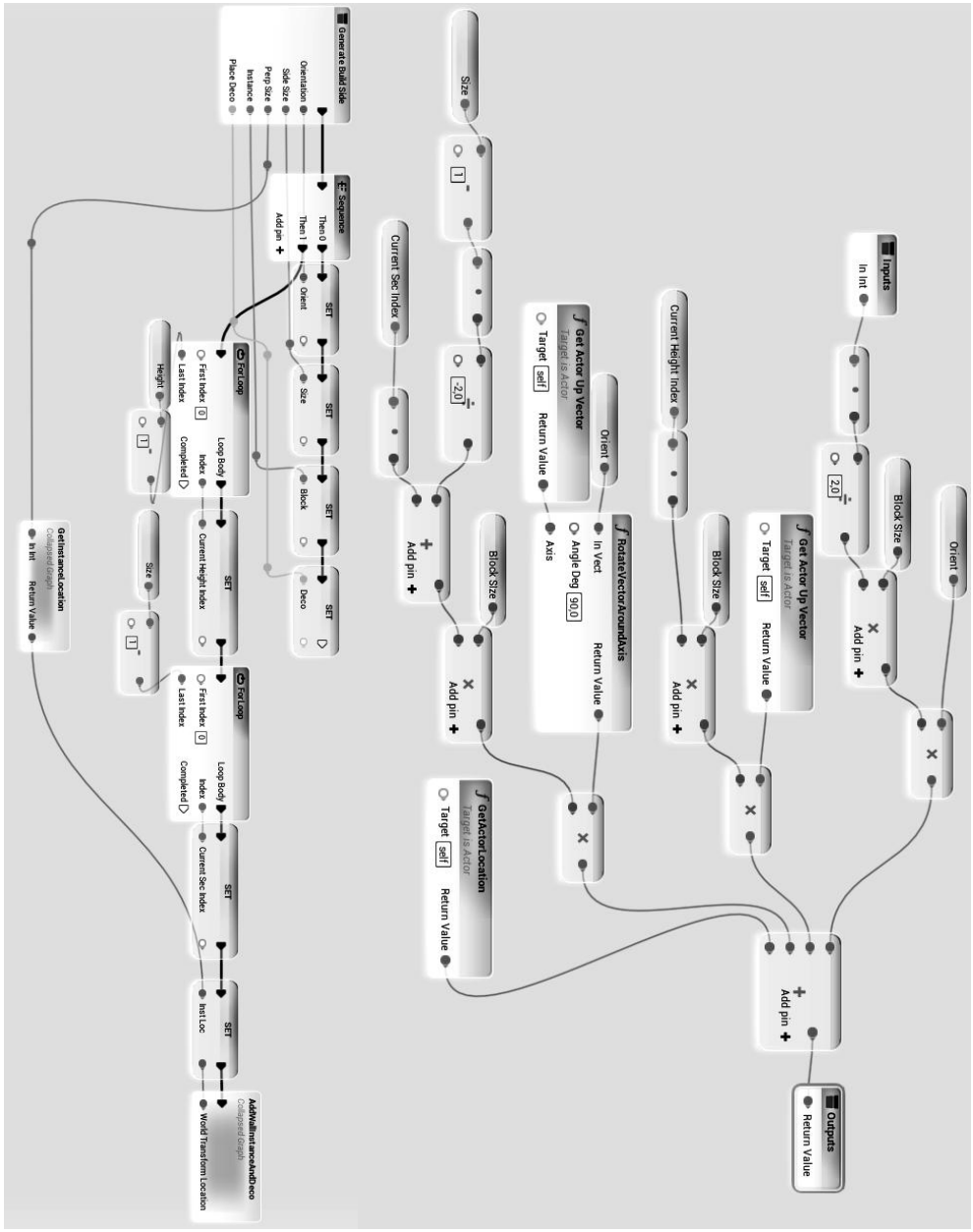

Find City Square Middle
Square
Break SquareDataStruct
Square Data Struct
X Min
Y Min
X Max
Y Max
Freq
Return Node
Location
X Size
Y Size
Make Vector
X
Y
Z
Return Value
Add pin
2,0
Break Vector
In Vec
X
Y
Z
GetActorLocation
Target is Actor
Target
Return Value
Self

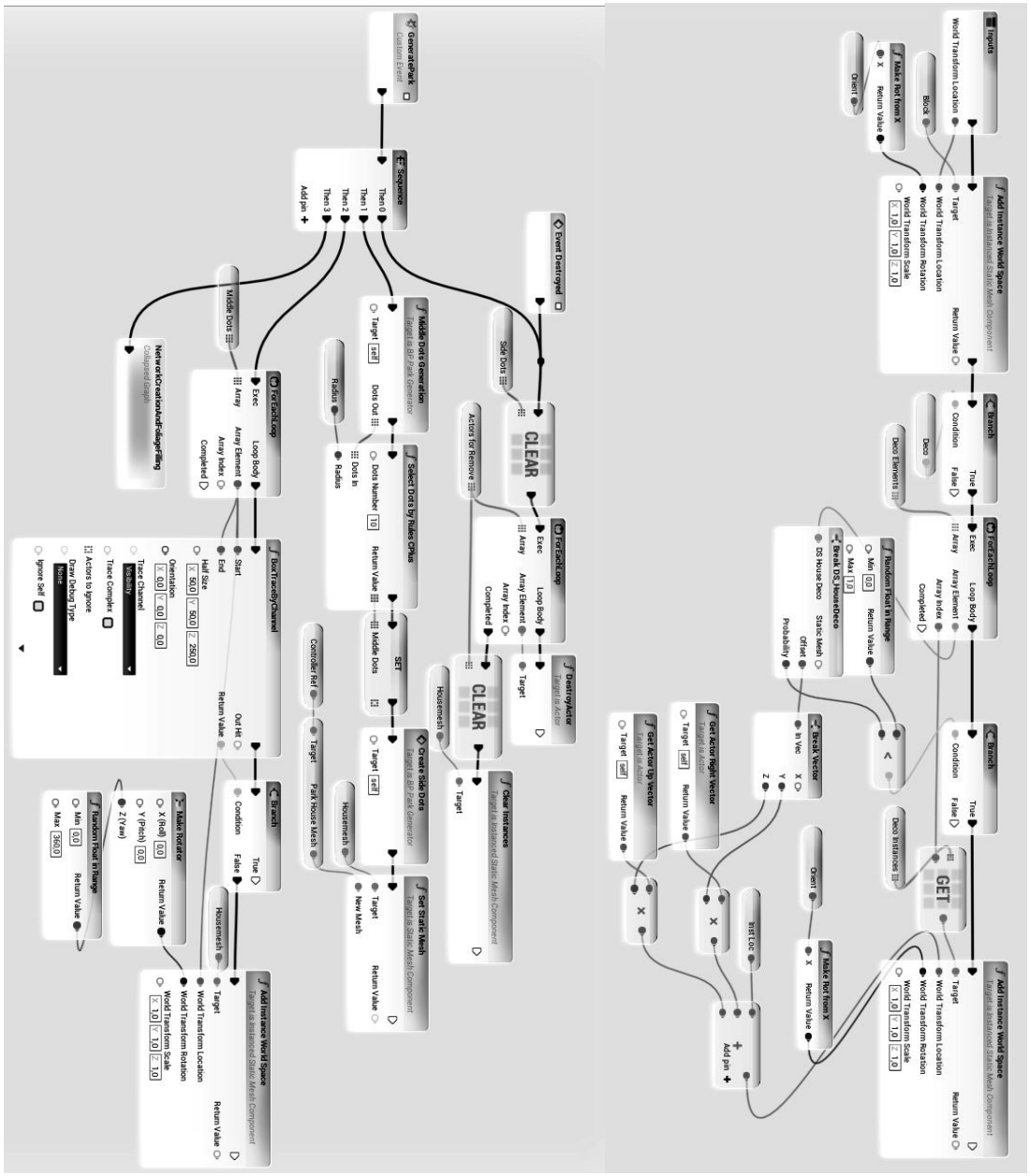

Generate
Custom Event
Sequence
Then 0
Then 1
Then 2
Then 3
Add pin
Clean All
Target is BP Road Gen Circles
Target
self
Generate Circles
Target is BP Road Gen Circles
Target
self
Connect Circles
Target is BP Road Gen Circles
Target
self
Generate Houses Along the Roads
Target is BP Road Gen Circles
Target
self


Get City Center
Return Node
Center
GetActorLocation
Target is Actor
Target
Return Value
Get Actor Forward Vector
Target is Actor
Target
Return Value
Self
Size X
Size Y
0,5
Add pin
Get Actor Right Vector
Target is Actor
Target
Return Value


Return Node
Yes
In Cicle Connection
ForLoop
First Index
Last Index
Loop Body
Index
Completed
Loc Vectors
LAST INDEX
Crossroads
GET
Target
Crossroad
Left to Right
Connect With
Target is BP_Modular Crossroad
Road Connector
Crossroads Ref
SET
Curr Ind
Make DS_ActorsArr
Array
DS Actors Arr
self
Return
Target




Inputs
In Int
Orient
Block Size
Current Height Index
Current Sec Index
Size
Get Actor Up Vector
Target is Actor
Target
self
Return Value
RotateVectorAroundAxis
In Vect
Angle Deg
90.0
Axis
GetActorLocation
Add pin
Outputs
Return Value
Generate Build Side
Orientation
Side Size
Perp Size
Instance
Place Deco
FF Sequence
Then 0
Then 1
SET
Height
Block
Deco
Inst Loc
For Loop
First Index
Last Index
Loop Body
Index
Completed
Current Height Index
Current Sec Index
GetInstanceLocation
Collapsed Graph
In Int
AddWallInstanceAndDeco
World Transform Location



НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”
Кафедра автоматизованих систем обробки інформації та управління

УЗГОДЖЕНО	**ЗАТВЕРДЖУЮ**
Керівник проекту	**В.о. завідувача кафедри**
_____ *С.Л.Проскура*	_____ *О.А.Павлов*
(підпис) (ініціали, прізвище)	(підпис) (ініціали, прізвище)
“16” квітня 2019 р.	“17” квітня 2019 р.

Інформаційна підтримка генерації міста на 3D-сцені

ТЕХНІЧНЕ ЗАВДАННЯ

Шифр *ДП ІС-5222.1181-с.ТЗ*

На 10 сторінках

Київ – 2019 року

ЗМІСТ

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Повне найменування системи та її умовне позначення

Повна назва системи: *Інформаційна підтримка генерації міста на 3D-сцені*

1.2 Найменування організації-замовника та організації-учасника робіт

Генеральним замовником проекту являється кафедра Автоматизованих систем обробки інформації та управління НТУУ "КПІ". Представником замовника є Проскура Світлана Леонідівна.

Розробником системи є студент групи ІС-52 факультету інформатики та обчислювальної техніки НТУУ «КПІ ім. Ігоря Сікорського» Трухан Григорій Олександрович.

1.3 Перелік документів, на підставі яких створюється система

При розробці системи і створення проектно-експлуатаційної документації Виконавець повинен керуватися вимогами наступних нормативних документів:

- ДСТУ 19.201-78. Технічне завдання. Вимоги до змісту і оформлення;
- ДСТУ 34.601-90. Комплекс стандартів на автоматизовані системи. Автоматизовані системи. Стадії створення;
- ДСТУ 34.201-89. Інформаційні технології. Комплекс стандартів на автоматизовані системи. Види, комплексність і позначення документів при створенні автоматизованих систем.

1.4 Планові терміни початку і закінчення роботи зі створення системи

Плановий термін початку роботи над створенням системи генерації міста на 3D-сцені – 5 лютого 2019 року.

Плановий термін по закінченню роботи над створенням системи генерації міста на 3D-сцені – не пізніше 1 червня 2019 року.

2 ПРИЗНАЧЕННЯ І МЕТА СТВОРЕННЯ КОМПЛЕКСУ ЗАДАЧ

2.1 Призначення комплексу задач

Призначенням розробки є створення системи генерації міста на базі Unreal Engine 4.

2.2 Цілі створення комплексу задач

Цілями розробки є спрощення процесу створення 3D-сцени розробниками відеоігор на Unreal Engine 4 за рахунок:

- процедурної генерації міста та його елементів (створення та розміщення модульних будинків, створення карти доріг, генерація парків та рослинності);
- зменшення часу розробки ігрових рівнів;
- можливості задавати власні налаштування генерації, в тому числі передавати на вхід генератора власні 3D-моделі.

Для досягнення поставлених цілей необхідно вирішити наступні задачі:

- розробити контролер-генерації міста з усіма налаштуваннями;
- розробити підсистему генерації доріг та розміщення декоративних елементів доріг;
- розробити підсистему генерації будинку (актор-генератор будинку);
- розробити підсистему розміщення будинків в місті;
- розробити підсистему генерації парків та їх декоративних елементів;
- розробити підсистему генерації рослинності в парках та міських кварталах.

3 ХАРАКТЕРИСТИКА ОБ'ЄКТА АВТОМАТИЗАЦІЇ

Працювати з застосуванням можуть всі розробники на Unreal Engine 4. З його допомогою користувач може отримати готове згенеровано місто, яке можна використати для тестування ігрових механік або для релізної версії гри. Користувач може задавати власні налаштування генерації та передавати на вхід генератора власні 3D моделі (модульні елементи будинку, декоративні елементи різних типів, тайли дороги і т.д.).

Об'єктом автоматизації створення ігрової локації (міста) на 3D-сцені в Unreal Engine 4.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Дане розширення для Unreal Engine 4 має генерувати місто на 3D-сцені згідно заданих налаштувань та виконувати наступні функції:

1. Система повинна надати користувачу можливість вводити налаштування генерації;
2. Система повинна надати користувачу можливість передавати на вхід генерації власні 3D-моделі ;
3. Система повинна надавати користувачу можливість змінювати окремі результати генерації, без необхідності заново генерувати все місто.
   3.1. Змінювати, додавати та видаляти актори-будинки;
   3.2. Перегенерувати вже розміщені в міських кварталах парки;
   3.3. Видаляти будь які окремі актори генерації.

4.2 Вимоги до надійності

Система повинна зберігати працездатність і забезпечувати відновлення своїх функцій при виникненні наступних позаштатних ситуацій:

- задання надто великої генерації, яку не може обробити машина;
- видалення акторів генерації зі сцени, які посилаються або на які посилаються інші актори.

Система повинна бути працездатною після перезапуску Unreal Engine 4 внаслідок його зависання через генерацію занадто великої локації. Система повинна без проблем очищувати сцену та перезапускати генерацію після видалення будь-яких акторів зі сцени.

4.3 Вимоги до складу і параметрів технічних засобів

Склад, структура і способи організації даних в системі повинні бути визначені на етапі технічного проектування.

Структура технічних засобів визначається виходячи із можливості їх забезпечити процедуру генерації міста вказаного розміру та складності.

Для правильної роботи розробленої системи до складу технічних засобів повинен входити комп’ютер, що має конфігурацію наведену нижче:

- комп’ютер з такою конфігурацією:

  1) двох ядерний процесор з частотою не нижче 2.5 ГГц;

  2) достатній об’єм оперативної пам’яті (не менше 4 ГБ);

  3) відеокарта NVIDIA GeForce 470 GTX або AMD Radeon 6870 HD series.

- додатково має бути встановлене таке програмне забезпечення:

  1) операційна система Windows 7 64-bit або Mac OS X 10.9.2;

  2) Visual Studio 2017 або новіша, відповідно до встановленої версії Unreal Engine 4;

  3) DirectX 11 або вище;

  4) Unreal Engine 4.21 або вище.

- комп’ютерна периферія, до складу якої входить:

  1) монітор;

  2) мишка або тачпад;

  3) клавіатура.

5 СТАДІЇ І ЕТАПИ РОЗРОБКИ

Основні етапи виконання робіт з інформаційної підтримки генерації міста на 3D-сцені.

№	Назва етапу роботи	Термін виконання етапу	Результат виконання
1	Підготовка технічного завдання на розробку програмного продукту	07.02.2019	
2	Розробка сценарію роботи	12.02.2019	
3	Технічне проектування – функціональність, модулі, задачі, цілі тощо	20.02.2019	
4	Узгодження з керівником інтерфейсу користувача	02.03.2019	
5	Розробка інформаційного забезпечення	17.03.2019	
6	Розробка програмного забезпечення	29.03.2019	
7	Налагодження програми	13.04.2019	
8	Тестування програми	27.04.2019	
9	Здача готового програмного продукту замовнику	12.05.2019	

6 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ СИСТЕМИ

6.1 Види випробувань

Для контролю правильності роботи програмного забезпечення буде проведено модульне тестування (Unit Test). В ході тестування буде проведено випробування основних підсистем генерації при запуску генерації з 0 та повторній генерації окремих елементів міста. Буде проведено випробування коректності роботи акторів генерації при розміщенні їх на сцені вручну.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”
Кафедра автоматизованих систем обробки інформації та управління

УЗГОДЖЕНО	**ЗАТВЕРДЖУЮ**
Керівник проекту	**В.о. завідувача кафедри**
_____ *С.Л.Проскура*	_____ *О.А. Павлов*
(підпис) (ініціали, прізвище)	(підпис) (ініціали, прізвище)
“13” травня 2019 р.	“14” травня 2019 р.

Інформаційна підтримка генерації міста на 3D-сцені

ПРОГРАМА ТА МЕТОДИКА ВИПРОБУВАНЬ

Шифр *ДП ІС-5222.1181-с.ПМВ*

на 13 сторінках

Київ – 2019 року

ЗМІСТ

1 ОБ'ЄКТ ВИПРОБУВАННЯ

1.1 Найменування програми

Темою дипломного проекту є «Інформаційна підтримка генерації міста на 3D-сцені»

1.2 Область застосування

Область застосування програми – розробка великих 3D-сцен міського типу для ігрових проектів на базі Unreal Engine 4.

Застосування спрощує процес створення сцени завдяки процедурному розміщенню всіх елементів міста. Користувач отримує доступ до великої кількості налаштувань генерації, може передавати на вхід роботи генератора власні 3D-моделі.

1.3 Умовне позначення програми

Умовне позначення програми – UE4 проект CityGenerator.

.

2 МЕТА ВИПРОБУВАНЬ

Мета проведення випробувань – перевірка відповідних характеристик розробленої програми (програмного виробу) функціональним і окремим іншим видам вимог, викладених в документі технічного завдання.

3 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

3.1 Вимоги до функціональних характеристик

Програмне застосування повинно:

- бути функціонально достатнім (повним);
- бути надійним (застосування повинно коректно завершити роботу без втрати даних);
- бути придатним до модернізації та масштабування;
- мати інтуїтивно зрозумілий для користувача інтерфейс;
- бути стійким до хибних дій користувача.

3.1.1 Вимоги до складу виконуваних функцій

Застосування має виконувати наступні функції:

- генерувати місто «з одної кнопки» на обраній сцені;
- коректно генерувати карти доріг за одним з двох алгоритмів (ділення прямокутників та побудова павутини);
- коректно генерувати будинок та його декоративні елементи
- коректно розміщувати будинки в міських кварталах;
- коректно розміщувати парки, його декоративні елементи (паркові дороги, альтанки, рослинність);
- надавати користувачу можливість вводити власні налаштування генерації;
- надатавати користувачу можливість передавати на вхід генерації власні 3D-моделі.

4 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

Програмний продукти розробляється на основі Технічного Завдання.

Випробування проводяться на основі наступних документів:

– ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;
– ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

5 СКЛАД І ПОРЯДОК ВИПРОБУВАНЬ

Умови проведення випробувань: програма з готовими модулями та зв’язками між ними.

Умови початку та завершення окремих етапів тестування: тестування кожних елементів програми має відбуватися з урахуванням всіх можливих виключних ситуацій в залежності від функціональних можливостей програмного продукту.

Обмеження щодо умов проведення тестування: тестування має проводитися в рамках функціонального апарату програмного забезпечення.

Вимоги до технічного обслуговування системи: система має бути розміщена на комютері з ОС Windows, розрядності 64-bit. Має бути встановлений Unreal Engine 4.21+ та Visual Studio 2017+.

Міри, забезпечуючі безпеку та безаварійність проведення тестування: тестування системи не може визвати аварійних ситуацій.

Порядок взаємодій організацій, які беруть участь у тестуванні: тестування проводить один студент КПІ групи ІС-52 Трухан Григорій Олександрович.

6 МЕТОДИ ВИПРОБУВАНЬ

У наступних таблицях наведений перелік випробувань основних функціональних можливостей.

Таблиця 6.1 – Генерація міста з прямокутною картою доріг

Мета тесту	Перевірка генерації міста з прямокутною картою доріг
Початковий стан моделі	Відкрита 3D-сцена, актор CityGenerationController розміщений на сцені
Вхідні дані:	Метод генерації доріг = squares, 3D-моделі доріг, налаштування для прямокутної карти доріг
Схема проведення тесту:	Ввести налаштування, натиснути «Start generation» у вкладці details актора CityGenerationController
Очікуваний результат:	Згенероване місто з прямокутною картою доріг, заданою кількістю парків та будинками, розміщеними в кварталах
Стан моделі після проведення випробувань:	Згенероване місто з прямокутною картою доріг, заданою кількістю парків та будинками, розміщеними в кварталах

Таблиця 6.2 – Генерація міста з павутиноподібною картою доріг

Мета тесту	Перевірка генерації міста з павутиноподібною картою доріг
Початковий стан моделі	Відкрита 3D-сцена, актор CityGenerationController розміщений на сцені
Вхідні дані:	Метод генерації доріг = net, 3D-моделі доріг, налаштування для павутиноподібної карти доріг

Продовження таблиці 6.2

Мета тесту	Перевірка генерації міста з павутиноподібною картою доріг
Початковий стан моделі	Відкрита 3D-сцена, актор CityGenerationController розміщений на сцені
Вхідні дані:	Метод генерації доріг = net, 3D-моделі доріг, налаштування для павутиноподібної карти доріг
Схема проведення тесту:	Ввести налаштування, натиснути «Start generation» у вкладці details актора CityGenerationController
Очікуваний результат:	Згенероване місто з павутиноподібною картою доріг, дороги плавно входять в перехрестя (відбувається деформація), розміщені будинки вздовж доріг, місто має круглу форму
Стан моделі після проведення випробувань:	Згенероване місто з павутиноподібною картою доріг, дороги плавно входять в перехрестя (відбувається деформація), розміщені будинки вздовж доріг, місто має круглу форму

Таблиця 6.3 – Генерація парку

Мета тесту	Перевірка генераціх парків
Початковий стан моделі	Відкрита 3D-сцена, актор CityGenerationController розміщений на сцені
Вхідні дані:	Метод генерації доріг = squares, 3D-моделі рослинності для парку, 3D-моделі декоративних елементів для парку, кількість парків у місті, налаштування генерації парку, налаштування генерації прямокутних доріг

Продовження таблиці 6.3

Мета тесту	Перевірка генерації міста з павутиноподібною картою доріг
Схема проведення тесту:	Ввести налаштування, натиснути «Start generation» у вкладці details актора CityGenerationController
Очікуваний результат:	Згенеровано місто з прямокутною картою доріг, у кварталах розміщена вказана кількість парків. В парку розміщені альтанки, паркові дороги, декоративні елементи, сам парк заповнений рослинністю
Стан моделі після проведення випробувань:	Згенеровано місто з прямокутною картою доріг, у кварталах розміщена вказана кількість парків. В парку розміщені альтанки, паркові дороги, декоративні елементи, сам парк заповнений рослинністю

Таблиця 6.4 – Генерація будинку

Мета тесту	Перевірка генерації будинку
Початковий стан моделі	Відкрита 3D-сцена
Вхідні дані:	Ширина, висота та довжина будинку, розмір блоку будинку, 3D-моделі архітектурних елементів будинку (стіни, кути, міжповерхові лінії), масив декоративних елементів будинку (включає ймовірність розміщення, посиланн на модель та відступ)
Схема проведення тесту:	Витягнути на сцену актор класу BP_HouseGenerator, ввести всі необхідні налаштування

Продовження таблиці 6.4

Мета тесту	Перевірка генерації міста з павутиноподібною картою доріг
Очікуваний результат:	Згенерований будинок в формі паралелепіпеда вказаних розмірів, на стінах розміщені задані декоративні елементи із відповідних масивів
Стан моделі після проведення випробувань:	Згенерований будинок в формі паралелепіпеда вказаних розмірів, на стінах розміщені задані декоративні елементи із відповідних масивів

Таблиця 6.5 – Розміщення будинків в прямокутному місті

Мета тесту	Перевірка розміщення будинків в місті з прямокутною картою доріг
Початковий стан моделі	Відкрита 3D-сцена, актор CityGenerationController розміщений на сцені
Вхідні дані:	Метод генерації доріг = squares, 3D-моделі доріг, налаштування для прямокутної карти доріг, налаштування та моделі будинків
Схема проведення тесту:	Ввести налаштування, натиснути «Start generation» у вкладці details актора CityGenerationController
Очікуваний результат:	Згенероване місто з прямокутною картою доріг. Будинки розміщені всередині міських кварталів та не виходять за рамки прямокутника міста. Висота будинків відповідає функції висоти. Будинки дивляться лицем на найближчу дорогу

Продовження таблиці 6.5

Мета тесту	Перевірка розміщення будинків в місті з прямокутною картою доріг
Стан моделі після проведення випробувань:	Згенероване місто з прямокутною картою доріг. Будинки розміщені всередині міських кварталів та не виходять за рамки прямокутника міста. Висота будинків відповідає функції висоти. Будинки дивляться лицем на найближчу дорогу

Таблиця 6.6 – Розміщення будинків в павутиноподібному місті

Мета тесту	Перевірка розміщення будинків в місті з павутиноподібною картою доріг
Початковий стан моделі	Відкрита 3D-сцена, актор CityGenerationController розміщений на сцені
Вхідні дані:	Метод генерації доріг = net, 3D-моделі доріг, налаштування для павутиноподібної карти доріг, налаштування та моделі будинків
Схема проведення тесту:	Ввести налаштування, натиснути «Start generation» у вкладці details актора CityGenerationController
Очікуваний результат:	Згенероване місто з павутиноподібною картою доріг. Будинки розміщені вздовж доріг та не виходять за останній радіус міських доріг + радіус околиць. Висота будинків відповідає функції висоти. Будинки дивляться лицем на дорогу, вздовж якої розміщені

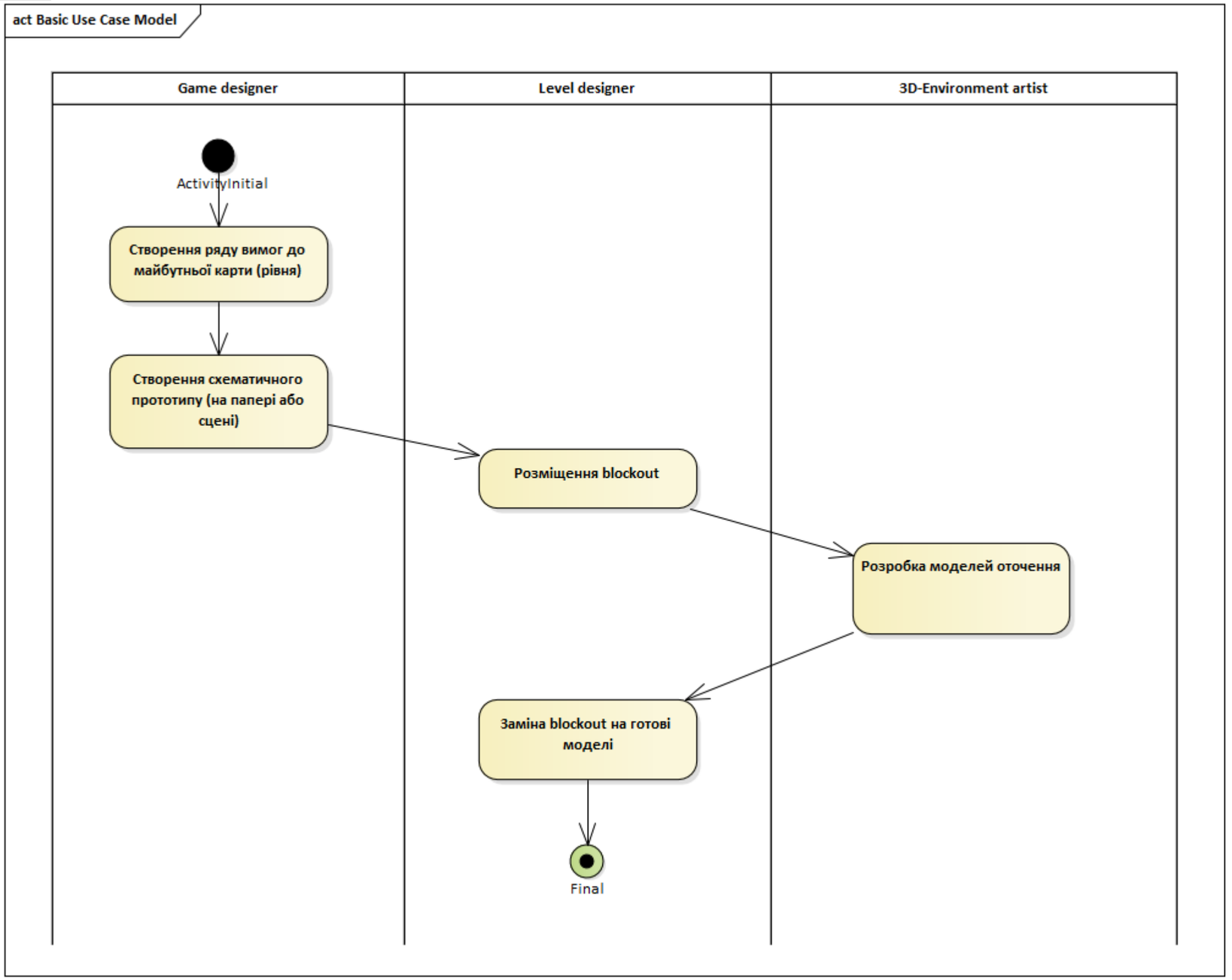
Продовження таблиці 6.6

Мета тесту	Перевірка розміщення будинків в місті з прямокутною картою доріг
Стан моделі після проведення випробувань:	Згенероване місто з павутиноподібною картою доріг. Будинки розміщені вздовж доріг та не виходять за останній радіус міських доріг + радіус околиц. Висота будинків відповідає функції висоти. Будинки дивляться лицем на дорогу, вздовж якої розміщені

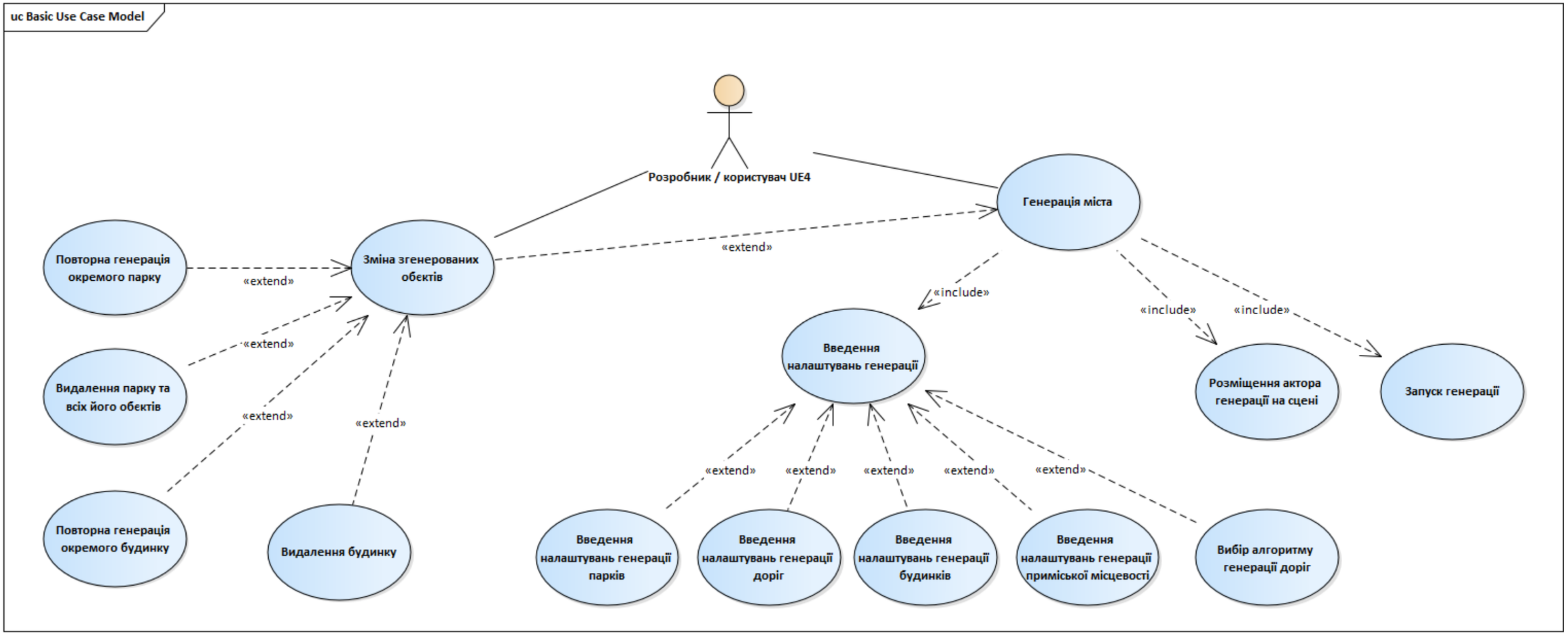
Графічний матеріал
до дипломного проекту

на тему: «Інформаційна підтримка генерації міста на 3D-сцені»

Київ – 2019 року

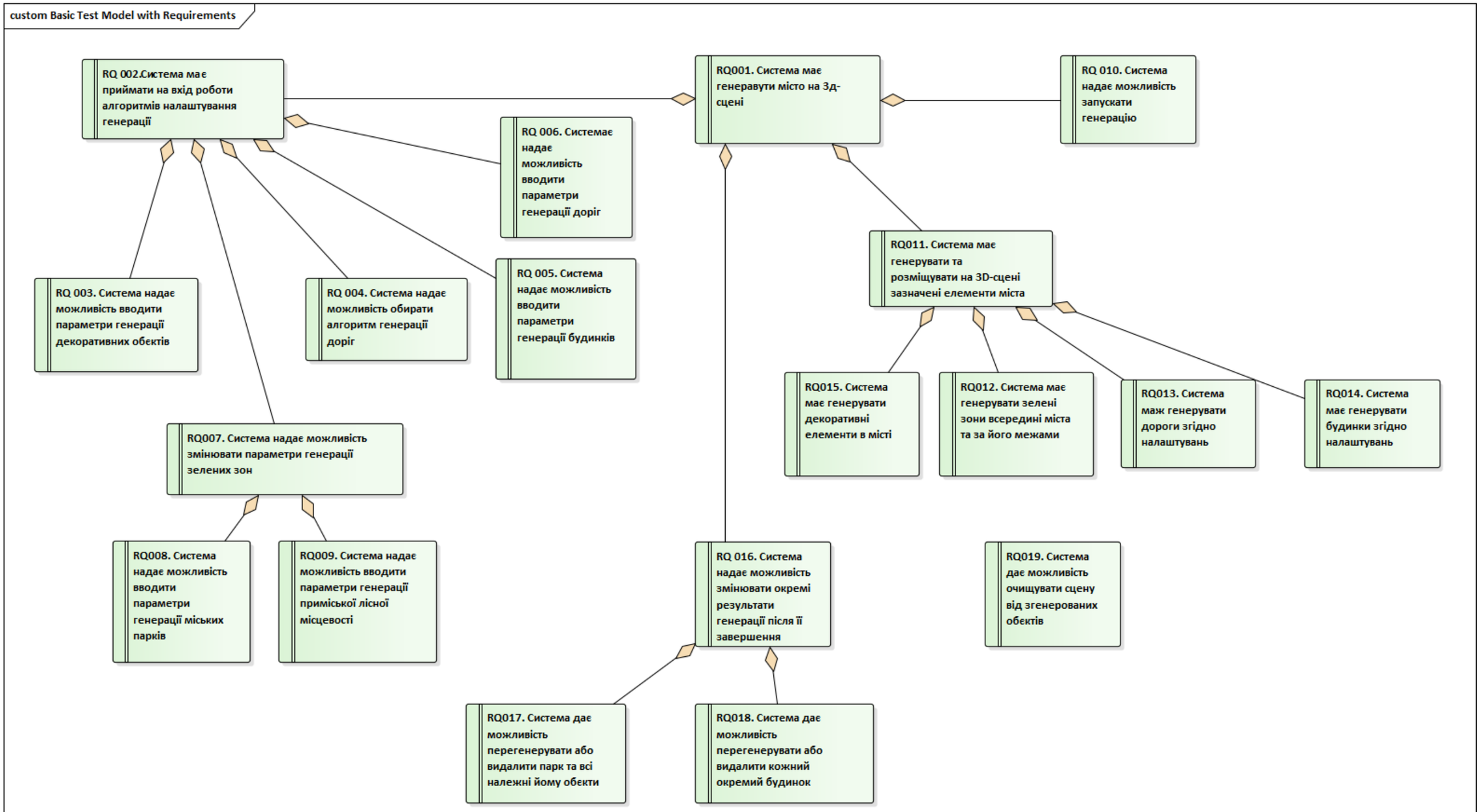


					ДП ІС-5222.1181-с.ССД		
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна діяльності	Літера	Маса / Масштаб
Розробив		Трухан Г.О.					
Перевірив		ПроскураС.Л.				Аркуш 1	Аркушів 6
Т. кон.							
Н. кон.		Халус О.А.			Інформаційна підтримка генерації міста на 3D-сцені	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-52	
Затвердив		ПроскураС.Л.					

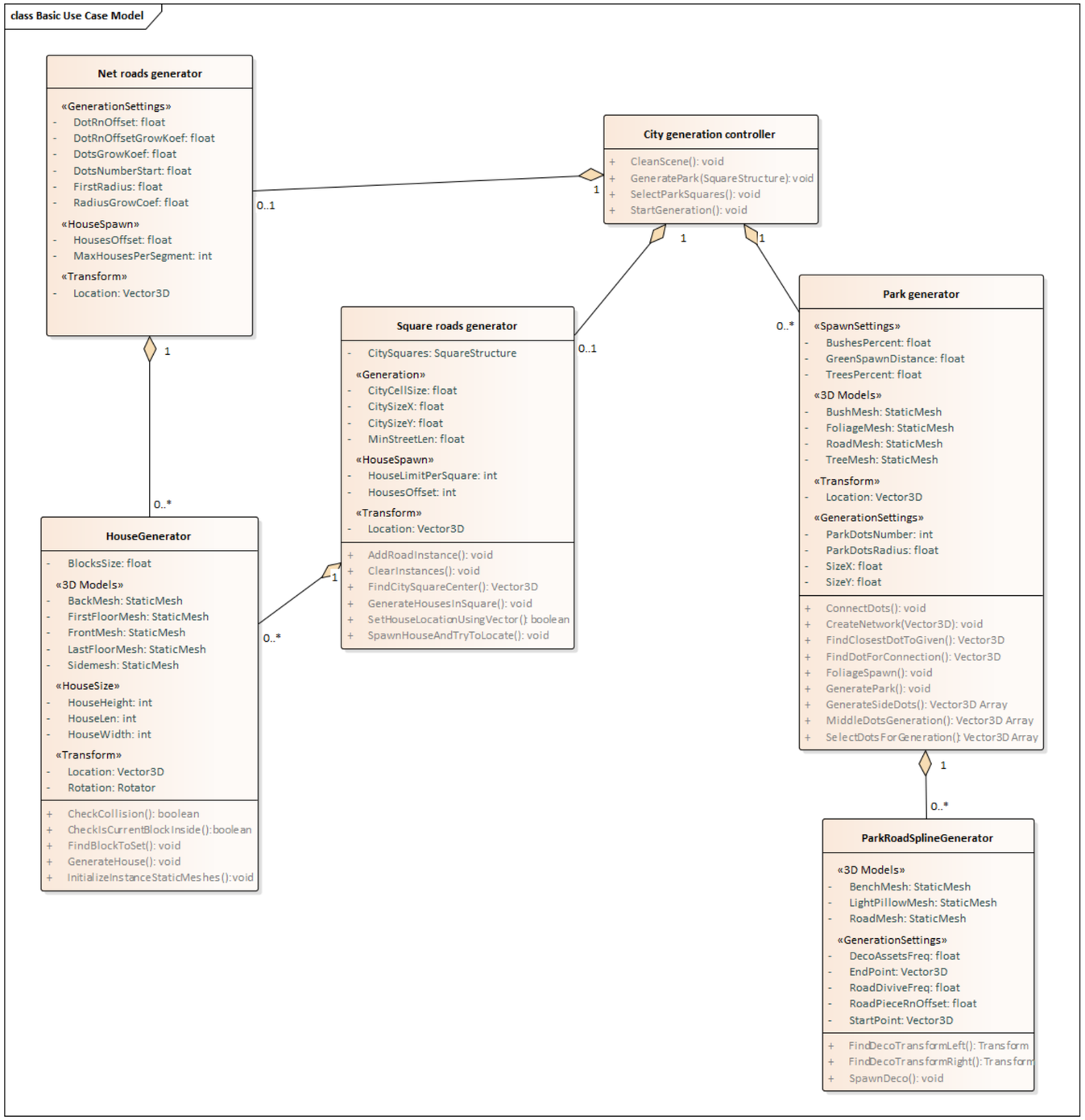


					ДП ІС-5222.1181-с.ССВ		
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна варіантів використання	Літера	Маса / Масштаб
Розробив		Трухан Г.О.					
Перевірив		ПроскураС.Л.				Аркуш 2	Аркушів 6
Т. кон.							
Н. кон.		Халус О.А.			Інформаційна підтримка генерації міста на 3D-сцені	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-52	
Затвердив		ПроскураС.Л.					

custom Basic Test Model with Requirements

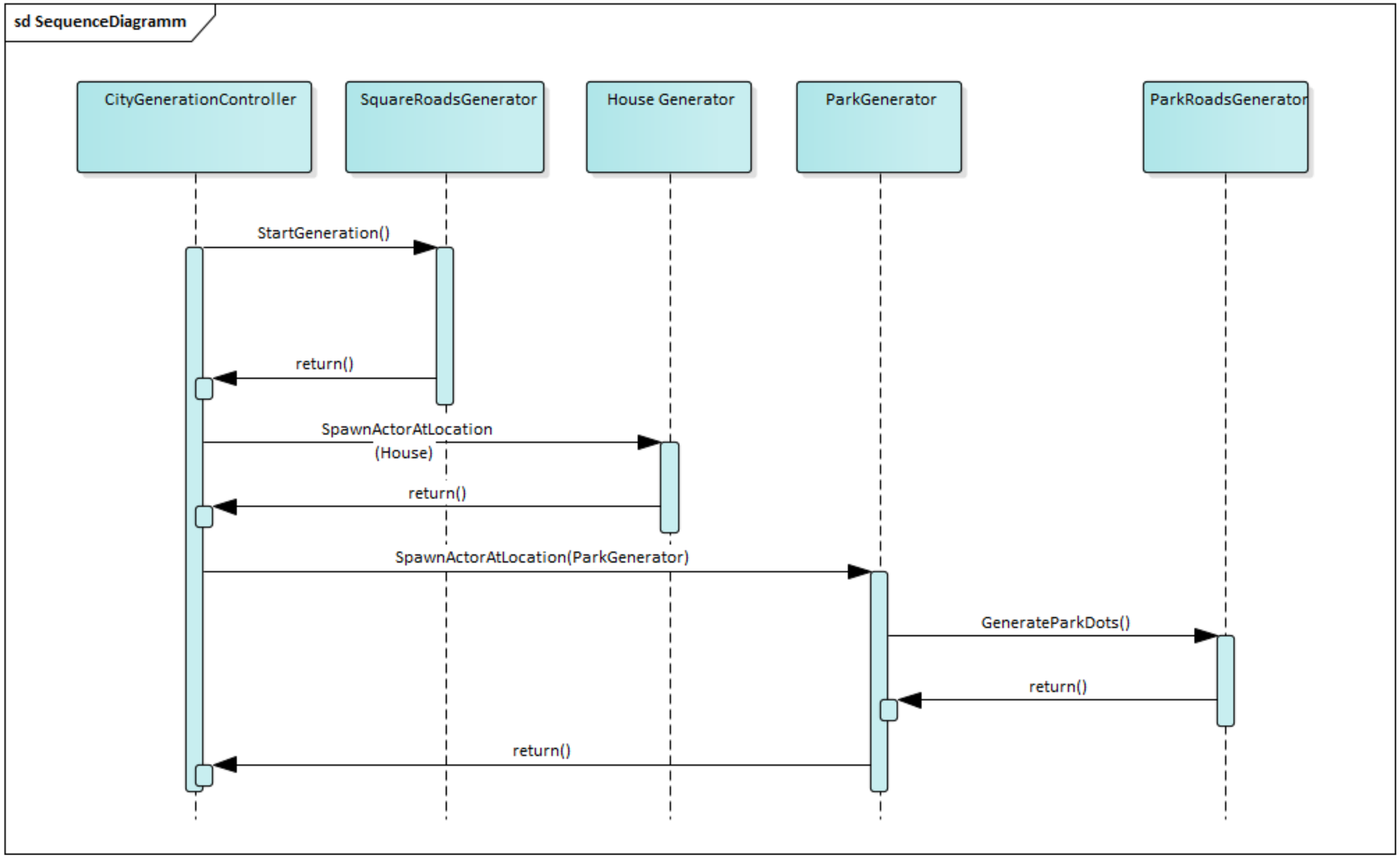


					ДП ІС-5222.1181-с.ССВ	
						Літера / Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата	*Схема структурна функціональних вимог*	
Розробив		Трухан Г.О.				Аркуш 3 / Аркушів 6
Перевірив		ПроскураС.Л.				
Т. кон.						
Н. кон.		Халус О.А.			*Інформаційна підтримка генерації міста на 3D-сцені*	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-52*
Затвердив		ПроскураС.Л.				



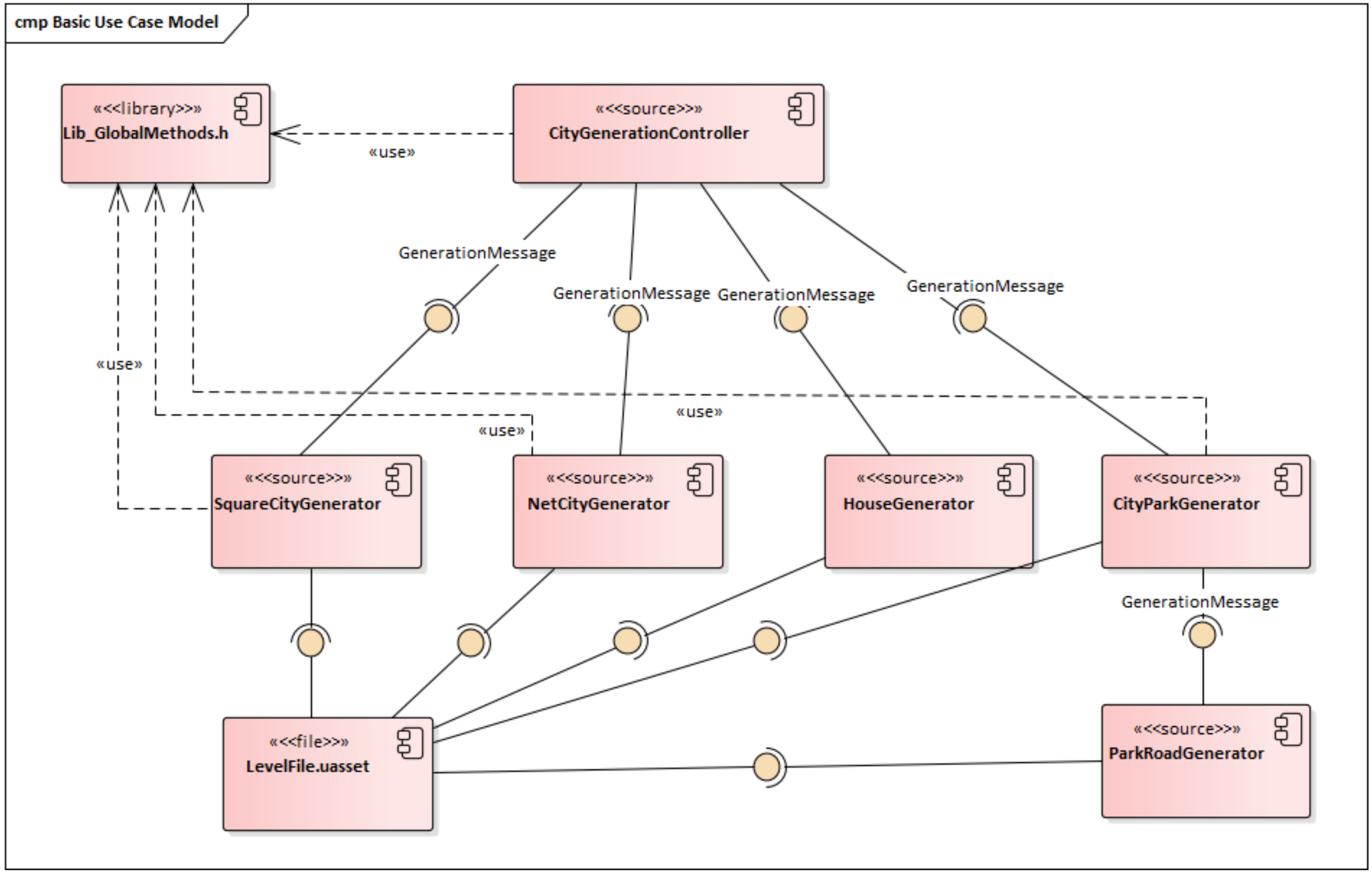
					ДП ІС-5222.1181-с.ССК	
					Схема структурна класів	Літера: — / Маса: — / Масштаб: —
Зм.	Арк.	№ документа	Підпис	Дата		Аркуш 4 / Аркушів 6
Розробив		Трухан Г.О.			Інформаційна підтримка генерації міста на 3D-сцені	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-52
Перевірив		ПроскураС.Л.				
Т. кон.						
Н. кон.		Халус О.А.				
Затвердив		ПроскураС.Л.				

ДП ІС-5222.1181-с.ССП



					ДП ІС-5222.1181-с.ССП	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна послідовності	Літера / Маса / Масштаб
Розробив		Трухан Г.О.				
Перевірив		ПроскураС.Л.				Аркуш 5 / Аркушів 6
Т. кон.						
Н. кон.		Халус О.А.			Інформаційна підтримка генерації міста на 3D-сцені	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-52
Затвердив		ПроскураС.Л.				

ДП ІС-5222.1181-с.ССК



					ДП ІС-5222.1181-с.ССК		
Зм.	Арк.	№ документа	Підпис	Дата	*Схема структурна компонентів*	Літера	Маса / Масштаб
Розробив		Трухан Г.О.					
Перевірив		ПроскураС.Л.				Аркуш 6	Аркушів 6
Т. кон.							
					Інформаційна підтримка генерації міста на 3D-сцені	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-52*	
Н. кон.		Халус О.А.					
Затвердив		ПроскураС.Л.					