Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

Розробка гри на рушієві Unity з використанням власної тайлової системи

Development of the game on the Unity engine using its own tile system

спеціальність 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

освітня програма «Інженерія програмного забезпечення»
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи ІПЗ-41
Воробчук Олександр Васильович
_____
(підпис)
Керівник: д.т.н., професор
Заяць Василь Михайлович
_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«01» 06 2023 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна
_____
(підпис)

Луцьк – 2023 року

Факультет *комп'ютерних та інформаційних технологій*
Кафедра інженерії програмного забезпечення
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *121 Інженерія програмного забезпечення*
Освітня програма: «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«02» 02 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Воробчуку Олександру Васильовичу

1. Тема кваліфікаційної роботи: «Гра на рушієві Unity з використанням власної тайлової системи / Development of the game on the Unity engine using its own tile system
керівник роботи: д.т.н., професор Заяць В. М.
затверджені наказом закладу вищої освіти від «03» лютого 2023 р. № 80-05-35
2. Строк подання здобувачем роботи: 19.06.2023 р.
3. Вихідні дані до роботи *Гра з використанням тайлових систем RimWorld.*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
Аналіз та детальний огляд гри RimWorld . Розробка своєї тайлової системи на основі шумових карт в рушієві Unity. Створення штучного інтелекту для наповнення ігрового світу.
5. Перелік графічного матеріалу: 15 рис; 24 лістингів;

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Заяць В. М.*		
Специфікація вимог до розробленої системи	*Заяць В. М.*		
Розробка об'єкта проектування	*Заяць В. М.*		
Нормоконтроль	*Заяць В. М.*		
Гарант ОП	*Ліщина Н.М.*		
Показник запозичень тексту		21 %	
Академічна доброчесність	*Яцук А.А.*		

7. Дата видачі завдання *«2» лютого 2023 р.*

КАЛЕНДАРНИЙ ПЛАН

№ :№ З/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1 1	*Огляд літературних джерел по темі кваліфікаційної роботи бакалавра*	*05.02.2023 р.*	вик.
2 2	*Аналіз проблеми розробки та впровадження об'єкту проектування*	*27.02.2023 р.*	вик.
3 3	*Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання*	*10.03.2023 р.*	вик.
4 4	*Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних*	*17.04.2023 р.*	вик.
5 5	*Практична реалізація об'єкта проектування та розробка бази даних*	*18.05.2023 р.*	вик.
6 6	*Тестування та налагодження об'єкта проектування*	*01.06.2023 р.*	вик.
7 7	*Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру*	*08.06.2023 р.*	вик.

Здобувач вищої освіти _____ Воробчук О.В.
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ Заяць В.М.
(підпис) (прізвище, ініціали)

Міністерство освіти і науки України
Луцький національний технічний університет

Рецензія
на кваліфікаційну роботу

Здобувач вищої освіти:
Воробчук Олександр Васильович

Тема: Розробка гри на рушієві Unity з використанням власної тайлової системи / Development of the game on the Unity engine using its own tile system

Коротка характеристика кваліфікаційної роботи:
Робота складається з трьох розділів, протягом яких дипломантом здійснюється огляд поставленої перед ним проблеми, пошук шляхів її вирішення та опис їх реалізації. Для реалізації тайлової системи було використано наступні програмні засоби: ігровий рушій Unity, програмна платформа для компіляції C# коду, FMOD для додаваня звуку в ігрі Самостійні розробки і пропозиції автора:
Кандидат пропонує використати згенеровані шумові карти для створення тайлової системи для повального наповненя тим персонажів зі штучним інтелектом.
Практичне значення роботи:
Кандидат створив простий проект, що може легко використовуватисьбудь-ким, як фундамент, для створення власної тайлової системи.

Недоліки:
У роботі не виявлено новації – вона більш походить на альтернативу існуючим варіантам вирішення проблеми, проте слід помітити що складна структура роботи добре висвітлює вміння дипломанта.
Загальний висновок:
Рівень якості виконаної роботи є задовільним. Усі поставлені завдання успішно й повноцінно виконані. Кваліфікаційна робота виконана на якісному та рекомендується до захисту Державній екзаменаційній комісії та заслуговує на високу оцінку.

Рецензент: доцент, к.т.н Кабак В.В
(прізвище, ім'я, по-батькові, посада)

Рецензент кваліфікаційної роботи _____
(підпис)

«8» червня 2023 р.

АНОТАЦІЯ

Воробчук О. В. Реалізація тайлової системи на іговому рушієві Unity. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2023. Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків до кожного розділу, загальних висновків, списку використаних джерел та додатків.У першому розділі досліджено предметну область де висвітлено стан проблеми, переваги, ризики та її перспективи. У другому розділі було здійснено аналіз та визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення. Третій розділ пояснює процес розробки тайлових систем посиланнями на лістинги коду у додатках, та здійснення тестування його тестування У висновках узагальнено інформацію, відображену у розділах. Ключові слова: тайлова система, Unity, шумові карти, матриця тайлів.

ANNOTATION

Vorobchuk O. V. Implementation of a tail system on the Unity web engine. Manuscript. Bachelor's qualifying thesis of the OP «Software Engineering». Lutsk National Technica l University. Lutsk, 2023. The bachelor's qualification work consists of an introduction, three sections, conclusions to each section, general conclusions, a list of used sources and appendices. In the first section, the subject area is investigated, where the state of the problem, advantages, risks and its prospects are highlighted. In the second section, the analysis and determination of the requirements for the developed software and software design were carried out. The third section explains the process of developing tile systems with references to code listings in the appendices, and testing its testing. Keywords: tile system, Unity, noise maps, tile mat

ВСТУП

Актуальність теми проявляється в тому, що завдяки швидкому розвитку технологій та ігрових рушіїв, спосіб створення великих ігор з відкритим і процедурно згенерованим світом зазнав значних змін. Сьогоднішні ігрові рушії дають можливість максимально гнучко налагодити внутрішні процеси розробки. Тайлові системи дозволяють швидко створювати різноманітні рівні, використовуючи набір готових тайлів. Це робить процес розробки більш ефективним і зменшує час, необхідний для створення нових рівнів. Вони дозволяють легко додавати нові тайли і розширювати набір графічних елементів для створення різноманітних рівнів.

Це дозволяє розробникам зосередитися на творчому процесі, а не на вирішенні технічних питань.

Метою дослідження є аналіз та розробка основних функціональних можливостей тайлових системи генерації рівнів на ігровому рушієві Unity та оптимізувати їх з точки зору продуктивності гри та операційної системи.

Завданням є розгляд і аналіз різних існуючих тайлових систем, які використовуються в ігрових рушіях Unity, для з’ясування їх переваг, недоліків та особливостей. Огляд різних підходів до тайлового мапування, системи колізій та інших функцій. Вивчення та впровадження оптимізаційних методів для покращення продуктивності тайлової системи. Оптимізацію алгоритмів розміщення та відображення тайлів, використання кешування даних та інші підходи до покращення продуктивності гри.

Об’єктом дослідження даної кваліфікаційної роботи бакалавра є дослідження структури тайлів та тайлсетів, які використовуються для створення рівнів. Вивчення способів організації та зберігання графічних ресурсів, включаючи текстури тайлів, властивості тайлів та їх анімацію.

РОЗДІЛ 1
АНАЛІЗ СТВОРЕНЯ ЛАНШАФТУ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Для створення свого прототипу Rimworld я вибрав саме C# який дуже близьким родичем мови програмування Java, і який є сумісним з ігровим рушієм Unity.

Unity – більше, ніж рушій, це середовище для розробки комп'ютерних ігор, в якій об'єднані різні програмні засоби, що використовуються при створенні програмного забезпечення – текстовий редактор, компілятор, відладчик тощо. При цьому, завдяки зручності використання, Unity робить створення ігор максимально простим і комфортним, а мультиплатформеність рушія дозволяє розробникам охопити якомога більшу кількість ігрових платформ і операційних систем.

В першу чергу, Unity дає можливість розробляти гри, не вимагаючи для цього якихось особливих знань. Тут використовується компонентно-орієнтований підхід, в рамках якого розробник створює об'єкти (наприклад, головного героя) і до них додає різні компоненти (наприклад, візуальне відображення персонажа і способи управління ним). Завдяки зручному Drag & Drop інтерфейсу (рис. 1.1) і функціонального графічного редактора рушій дозволяє малювати карти, і розставляти об'єкти в реальному часі, а також відразу ж тестувати результат виконання.

При всіх своїх перевагах, рушій має і свої недоліки. При створенні більш масштабного проекту, випливає інша проблема рушія Unity – повільність. Створення масштабних, складних сцен з великою кількістю компонентів може негативно вплинути на продуктивність гри, в результаті чого розробникам доведеться витрачати додатковий час і ресурси на оптимізацію, а можливо – і видалення деяких елементів з проекту.

Крім того, додатки, створені на Unity, досить «великовагові». Навіть

найпростіша піксельна гра може займати кілька сотень мегабайт на комп’ютері.

Так, для жорсткого диска комп’ютерів це невеликий обсяг, але, якщо проект розробляється і для мобільних платформ, слід задуматися про оптимізацію його розміру (рис. 1.2). Також для імерссивності ігравця я хотів би додати аудіорушії (audio middleware) – це програмні комплекси для впровадження та управління звуком у відеоіграх і інших інтерактивних додатках.

FMOD – комерційна аудіобібліотека від компанії Firelight Technologies, що дозволяє програвати музичні файли різних форматів в різних операційних системах і платформах. Використовується в іграх та застосунках для забезпечення різноманітних аудіоможливостей (рис. 1.3).

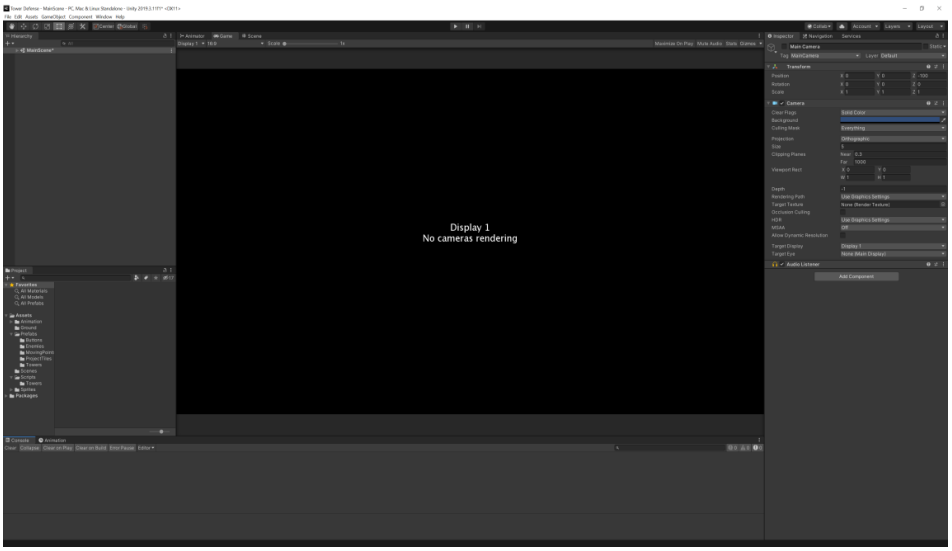

Рисунок 1.1 – Головний екран Unity

Рисунок 1.2 – Інтерфейс взаємодії рушія Unity [1]

Завдяки використанню FMOD можна буде покращити звучання гри, зробити її більш живою. За допомогою FMOD Engine можна інтегрувати звуки в ігровий рушій Unity та налаштувати відтворення звукових ефектів по бажанню. Це зможе суттєво зекономити час, замість створення власної звукової системи.

Рисунок 1.3 – Інтерфейс FMOD Studio [2]

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Ціль гри в тому, щоб вижити та відремонтувати пошкоджену пошукову систему для порятунку колонії. Щоб лишитися в «живих», персонажі повинні видобувати різні матеріали, полювати, досліджувати технології для того, щоб відремонтувати пошукову систему і послати сигнал порятунку на базу. Світ гри є процедурно згенерованим та складається з так званих плиток (тайлів). Тайли – невеликі зображення однакових розмірів, які служать фрагментами великої картини. Зазвичай тайлів на один «світ» роблять порядку декількох сотень.

Таким чином можна будувати величезні двомірні простори, які витрачають зовсім небагато пам’яті.

Проблема тайлової графіки в її монотонності – очі легко помічають, елементи, які повторюються.

З монотонністю борються так:

1) «розумно» ріжуть малюнок на тайли, передбачаючи перехідні тайли між різними поверхнями і не допускаючи в межах одного тайла областей з різною оптичною щільністю;

2) роблять кілька тайлів однієї й тієї ж текстури, які трішки відрізняються деталями. Тайли розташовують хаотично, щоб очі не змогли помітити порядок текстури;

3) роблять різні прикраси, що приваблюють погляд – фізично вони можуть бути реалізовані як тайли, поверх яких накладають ще один тайл з текстурою.

Процедурна генерація – це метод алгоритмічного створення даних за допомогою комбінацій алгоритмів, поєднаних із випадковістю. У комп’ютерній графіці він зазвичай використовується для створення текстур та 3D-моделей. У відеоіграх він використовується для автоматичного створення великої кількості контенту. Залежно від реалізації, переваги процедурної генерації можуть включати менший розмір файлу, більший обсяг контенту та випадковість для певного ігрового процесу.

Головне завдання при створенні генерації рівня – це підготувати необхідні матеріали та налаштувати пареметри для генерації, так як неправильно підібрані параметри можуть дуже нереалістично згенерувати рівень. Вона може сама розставити дерева або кущі, підставити ландшафт у потрібному місці і тому подібні речі.

Суть цього проекту створити копію гри RimWorld, яка являє собою науково-фантастичний симулятор колонії на базі тайлової системи (рис. 1.4).

Цілі кваліфікаційної роботи які повинні бути досягнуті в результаті виконання кваліфікаційної роботи можна поділити на такі пункти:

– аналіз предметної області, можливостей та перспективи, ризиків та переваг подібного проекту;
– реалізація обраної комп’ютерної гри;
– оволодіти методами побудови створення тайлової системи з використанням;
– шумових карт для створення ланшафту;
– розробити інтерфейс та звукову доріжку для іммерсивності гравця;
– оптимізувати програмне забезпечення;
– розробити економічну частину та доцільність створення ігор на ігровому рушієві Unity.

Рисунок 1.4 – Комп’ютерна гра RimWorld [3]

З точки зору програміста, для створення будь-якого предмета є 2 шляхи: зробити його руками або написати код, для його генерації.. Для початку визначимося з базовими поняттями:

– процедурна генерація – загальна назва для механізмів автоматичного створення контенту з будь-якого алгоритму. Це поняття включає в себе створення всього від музики до правил гри;

– створення контенту з будь-якого алгоритму. Це поняття включає в себе створення всього від музики до правил гри;активна спільнота. Codecademy

має активну спільноту користувачів, де можна обмінюватися досвідом, отримувати підтримку та співпрацювати з іншими програмістами;

– процедурна генерація – загальна назва для механізмів автоматичного створення контенту з будь-якого алгоритму.

Це поняття включає в себе створення всього від музики до правил гри;

– процедурна генерація – загальна назва для механізмів автоматичного створення контенту з будь-якого алгоритму. Це поняття включає в себе створення всього від музики до правил гри;

– генератор псевдовипадкових чисел (ГПВЧ) – алгоритм, який породжує послідовність чисел, елементи якої майже незалежні один від одного і підкоряються заданому розподілу;

– породжуючий елемент (англ. Seed) – визначає послідовність чисел ГПВЧ, які видаються.

Якщо необхідна невелика кількість контенту, то, звичайно ж, найпростішим способом буде створити його руками (розставити по карті дерева або намалювати текстуру). Однак якщо говорити про дійсно великі обсяги матеріалу, то процедурна генерація може стати більш вигідним способом. Принцип досить простий, використовуючи генератор псевдовипадкових чисел, який породжує елемент та додаткові вхідні дані ми отримуємо унікальну одиницю контенту. Згодом, якщо нам необхідно повторити згенерований об’єкт, то, знаючи його породжуючий елемент, вхідні дані, і внесені зміни можна відтворити його генерацію, і отримати даний об’єкт. Це загальний опис схеми, яка при використанні різних алгоритмів підходить практично для будь-якого завдання починаючи від генерації музики, продовжуючи текстурами вогню, закінчуючи генерацією тварин для якої-небудь планети або світу. Для генерації пов’язаних даних використовується генератор псевдовипадкових чисел під назвою Шум Перліна. Картинка нижче (рис. 1.5) показує як відбувається конфігурація нового світу в головному меню.

Рисунок 1.5 – Конфігурація нового світу в головному меню

Висновки до розділу 1

У першому розділі було розглянуто загальні відомості про рушій Unity, FMOD Studio, структуру ігрового рівня і план виконання. Я проаналізував якими технологіями створюються процедурно згенеровані світи і обрав оптимальний шлях для створення свого, а завдяки аудіо рушію я зможу краще передати атмосферу гри для гравці.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Тайл виглядає як картинка фіксованого розміру. Причому намальована таким чином, що при зіставленні з іншими тайлами виходить єдине зображення без помітних «швів». Найлегше уявити тайли як кахель, яким облицьовують стіни або підлоги. Найпростіший тайл – квадратне зображення, симетричне по вертикалі і горизонталі. Якщо скласти ці картинки одну з одною, то вийде велике полотно (рис. 2.1). А якщо внести в цю мальовничу картину для різноманітності ще один вид тайла, скажімо, що зображує дорогу, то можна вже скласти примітивну карту [4].

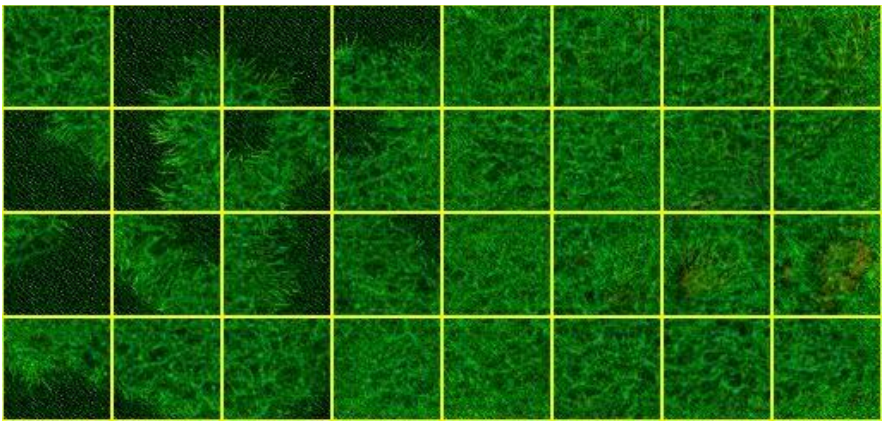

Рисунок 2.1 – Приклад тайлової карти [4]

Крім цього, тайли ідеально підходять для створення величезних світів. З використанням сегментування, коли карта ділиться на кілька шматків, кожен з яких динамічно завантажується, можна робити карти просто фантастичних розмірів. І це активно використовується в іграх жанру RPG.

Порахуємо витрати, які потрібні для створення карти розміром 512x512 клітинок. Припустимо, що сам тайл буде розміром 32x32 пікселя, тоді він буде займати в пам’яті 1024 байта – 1 кілобайт. Після підрахунків виявляється, що найпростіша карта займає в пам’яті близько 262 кілобайт плюс пам’ять для

зберігання зображень самих тайлів. А якщо малювати таку ж за площею область вручну, то буде використано 268 мегабайт пам’яті. І це при тому, що для простоти брався режим в 256 кольорів. Тайлова будова карт використовується в переважній більшості двомірних і в достатній кількості тривимірних ігор.

У реальних іграх вся карта є багатошаровою: перший шар – це земля, каміння, бруд, вода тощо; другий шар – будь-яка периферія типу дерев, скель, столів і стільців; третій шар – об’єкти: люди, монстри, ключі та магічні зілля.

Можна додавати ще шари для вищезазначених ефектів, але зазвичай для забезпечення кращої продуктивності використовують чотири-п’ять. Скажімо, четвертий шар в нашій схемі можна додати для створення об’єктів, які будуть перебувати вище рівня голови персонажів, наприклад дахів будинків. Всі зустрічалися з таким ефектом: коли герой підходить до будівлі впритул, у тієї раптом зникає дах, і можна побачити нутрощі будівлі. Так ось, дуже легко змусити тайлову систему перестати малювати четвертий шар, щоб користувач побачив все, що ховається під ним.

Тайлові системи, що засновують свою графіку на принципі палітр, активно використовують ефект ротації кольорів. Нехай якась синя гамма використовується для відтворення води. Тоді, змінюючи цю гаму або, навпаки, переставляючи індекси в тайлі, можна отримати ефект перетікання води. Той же самий фокус проходить і для інших поверхонь. Скажімо, в печері тайли, що зображують скелю, можна зробити темніше. А якщо вручити персонажу в руки факел і в залежності від його позиції освітлювати навколишню місцевість – ефект вийде дуже гарний. Не кажучи навіть про динамічну зміну дня і ночі.

Напрямок відліку розташування плитки по горизонталі, як правило, йде зліва на право (із заходу на схід).

Напрямок відліку розташування плитки по вертикалі може йти як зверху вниз (на південь), так і від низу до верху (на північ).

Початок відліку системи (положення плитки з параметрами X = 0, Y = 0) теж може здаватися по-різному.

На малюнку (рис. 2.2) наведено кілька прикладів систем координат з різними початками відліку і орієнтацією осей для рівня Z = 2.

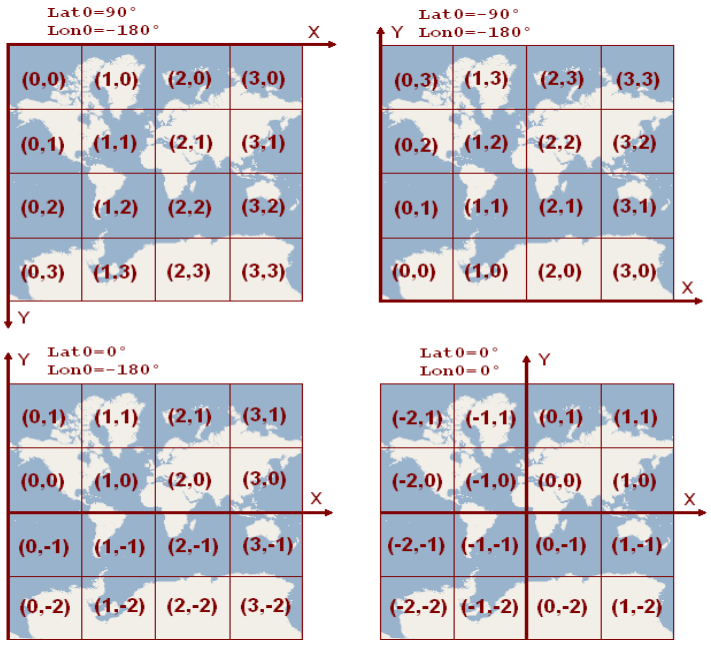

Рисунок 2.2 – Система координат тайлової системи [5]

Будь-яка карта – це масив, що задає розташування тайлів на своїх шарах, а також їх параметри. Найпростіша карта являє собою матрицю MxN (рис. 2.3), де кожен її елемент містить номер тайла з умовно прийнятої послідовності. Додаються нові шари, матриця ускладнюється, виходить вже набір (структура) матриць, а з використанням будь-яких ефектів виходить власний формат карти, який займає на диску не один мегабайт.

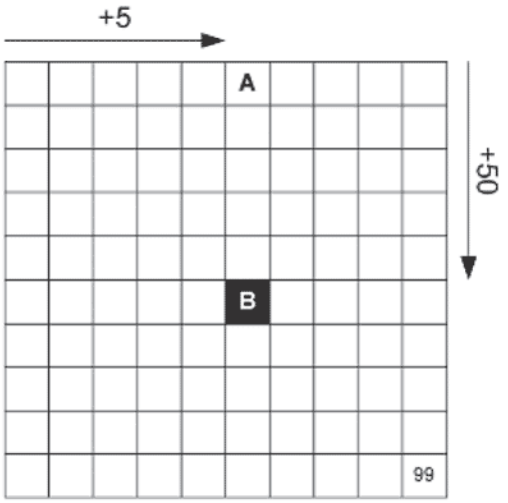

Рисунок 2.3 – Матриця тайлів [6]

Далі потрібно створити сітку (mesh). Для кожної плитки потрібно 4 вершини та 2 трикутники. Це дуже просто, можна побачити на зображенні (рис. 2.4) вершини плитки 0,0, тепер потрібно це зробити для кожної окремої плитки (і додати положення плитки).

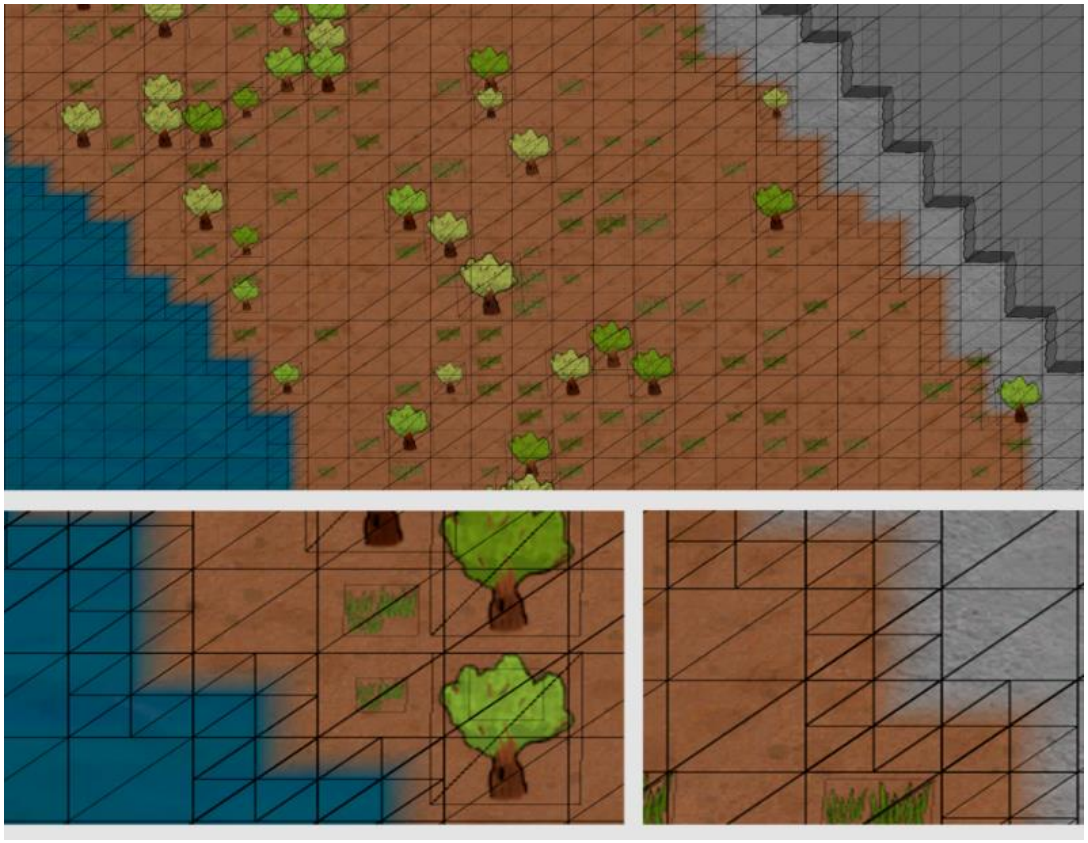

Рисунок 2.4 – Реалізація згладжування країв між різними тайлами

Але перед цим клас MeshData обробляє всі дані, які пов’язані із сітками. Наприклад, у грі використовуються деякі методи, такі як meshData.Update з вхідним параметром прапора (flags). Прапори можуть вказувати на оновлення UV/Colors тощо, або просто деякі допоміжні методи, такі як AddTriangle (vIndex, a, b, c).

Для згладжування країв між тайлами потрібно буде знати «сусідів» для кожної з наших плиток (максимум 8) та те, яка сітка використовувалася у наших «сусідів». Також потрібно буде знати тип тайлу кожного «сусіда».

Якщо сусіди різні, то додається новий прямокутник до сітки та ділиться на чотири невеликих прямокутника (рис. 2.5).

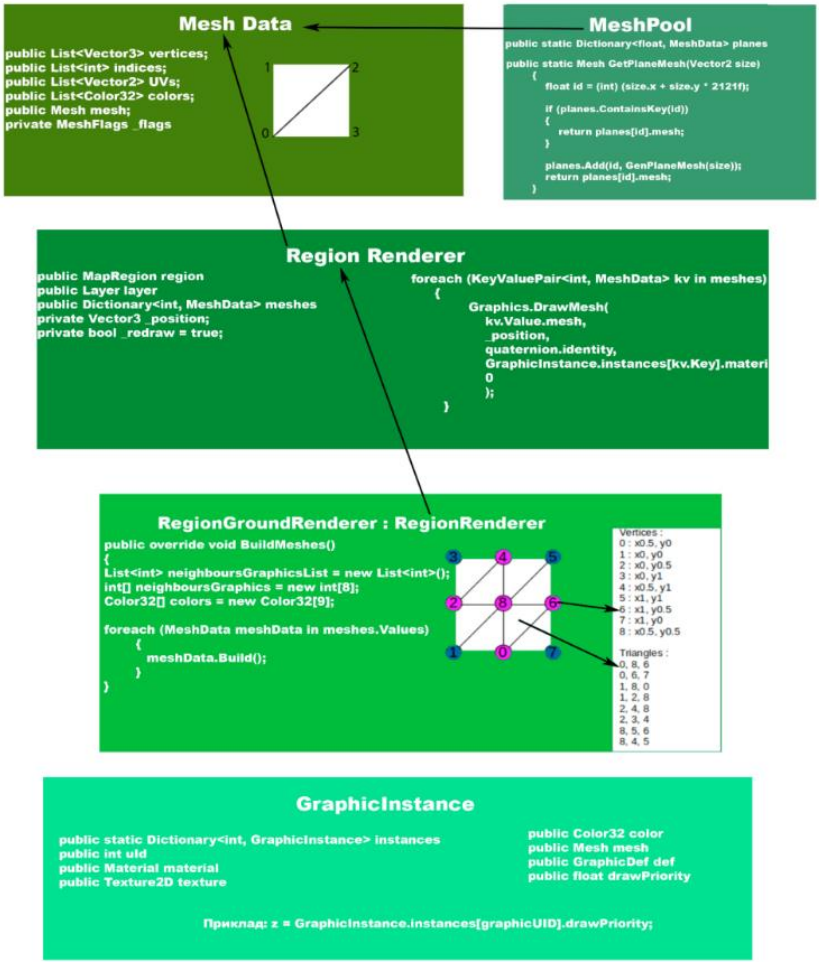

Рисунок 2.5 – Система відтворення тайлів

Створення візуально привабливого та різноманітного набору плиток – це трудомісткий процес, але результати часто того варті. Однак навіть після створення тайлів, всеодно потрібно розташувати тайли на свої місця. Автоматизувати цей процес допомагає бітмаскинг.

Бітмаскинг плитки – це метод автоматичного вибору відповідного спрайта з певного набору плиток. Це дозволяє розміщувати загальну плитку скрізь, де потрібно, щоб з’явився певний тип місцевості, замість того, щоб вручну розміщувати величезний вибір різноманітних плиток.

При роботі з різними типами місцевості кількість різних варіацій може перевищувати 300 і більше плиток. Відтворення цієї безлічі різних спрайтів – це, безумовно, трудомісткий процес, але бітмаскінг плитки гарантує швидкий та ефективний акт розміщення цих плиток. За допомогою статичної реалізації бітового маскування карти все генерує під час виконання.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Стандартний спосіб генерації 2D–карт полягає у використанні в якості будівельного блоку функції шуму з обмеженою смугою частот, наприклад шуму Перліна. Ось, як виглядає функція шуму (рис. 2.6). Кожній точці карти потрібно присвоїти число від 0.0 до 1.0. У цьому зображенні 0.0 – це чорний колір, а 1.0 – білий. Наступний скрипт – NoiseMap він розроблявся для генерації карти шуму, у ньому ми генеруємо масив висот від 0 до 1 по заданим параметрам: розмір, породжуючий елемент, кількість октав, постійність, лакунарність та зміщення.

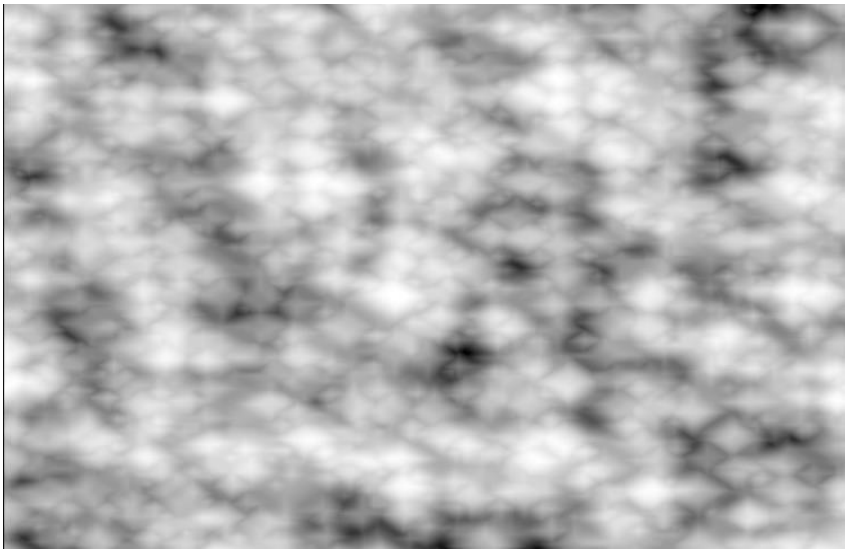

Рисунок 2.6 – Приклад шуму Перліна

Тайли не єдиний метод процедурної генерації. Автори Minecraft і Terraria, наприклад, використовують шум Перліна – алгоритм градієнтного шуму. Спочатку його придумали для створення достовірних текстур поверхні. Згодом, однак, дизайнери знайшли йому ще одне застосування – в процедурній генерації ландшафту.

Ігровий штучний інтелект зосереджений на тому, які дії повинен виконувати об’єкт, виходячи з умов, в яких знаходиться. Зазвичай це називають управлінням «інтелектуальними агентами», де агент є ігровим персонажем,

транспортним засобом, ботом, а іноді і чимось більш абстрактним: цілою групою сутностей або навіть цивілізацією. У кожному разі це річ, яка повинна бачити своє оточення, приймати на його основі рішення і діяти відповідно до них.

Це називається циклом Sense / Think / Act (Відчувати / Мислити / Діяти):

– Sense: агент знаходить або отримує інформацію про речі в своєму середовищі, які можуть вплинути на його поведінку (загрози поблизу, предмети для збору, цікаві місця для дослідження);

– Think: агент вирішує, як реагувати (розглядає, чи достатньо безпечно збирати предмети або спочатку він повинен битися / ховатися);

– Act: агент виконує дії для реалізації попереднього рішення (починає рух до супротивника або предмету).

У штучного інтелекту є ряд обмежень, які необхідно дотримуватися:

– штучний інтелект не потрібно заздалегідь тренувати, ніби це алгоритм машинного навчання. Безглуздо писати нейромережу під час розробки, щоб спостерігати за десятками тисяч гравців і вивчати кращий спосіб гри проти них. Тому що гра не випущена, а гравців немає;

– гра повинна розважати і кидати виклик, тому агенти не повинні знаходити найкращий підхід проти людей.

Агентам потрібно виглядати реалістичними, щоб гравці відчували ніби грають проти справжніх людей. Програма AlphaGo перевершила людей, але вибрані кроки були сильно далекі від традиційного розуміння гри. Алгоритм потрібно змінити, щоб він приймав правдоподібні рішення, а не ідеальні.

Штучний інтелект повинен працювати в реальному часі. Це означає, що алгоритм не може монополізувати використання процесора протягом тривалого часу для прийняття рішень. Навіть 10 мілісекунд на це – занадто довго, тому що більшості ігор досить від 16 до 33 мілісекунд, щоб виконати всю обробку і перейти до наступного кадру графіки. Кожна частина дерева називається node (вузол) – штучний інтелект використовує теорію графів для опису подібних структур(рис. 2.7).

Є два типи вузлів:

– вузли прийняття рішень: вибір між двома альтернативами на основі перевірки деякої умови, де кожна альтернатива представлена у вигляді окремого вузла;

– кінцеві вузли: дія для виконання, що представляє остаточне рішення;

– Алгоритм починається з першого вузла («кореня» дерева). Він або приймає рішення про те, до якого дочірнього вузлу перейти, або виконує дію, що зберігається в вузлі, і завершується.

Існує кілька способів їх реалізації, але суть для всіх приблизно однакова і схожа на дерево рішень – алгоритм починається з «кореневого» вузла, а в дереві знаходяться вузли, що представляють рішення або дію.

Правда є кілька ключових відмінностей:

– тепер вузли повертають одне з трьох значень: Succeeded (якщо робота виконана), Failed (якщо не можна запустити) або Running (якщо вона все ще запущена і немає кінцевого результату);

– більше немає вузлів рішень для вибору між двома альтернативами. Замість них є вузли Decorator, у яких є один дочірній вузол. Якщо вони виконані(Succeed), то виконують свій дочірній вузол;

– вузли, які виконують дії, повертають значення Running для подання виконуваних дій;

Цей невеликий набір вузлів можна об’єднати для створення великої кількості моделей.

З цією структурою (рис. 2.7) не повинно бути явного переходу від станів Idling / Patrolling до стану Attacking або будь-яких інших. Якщо ворога видно, а здоров’я персонажа низьке, виконання зупиниться на вузлі Fleeing, незалежно від того, який вузол він раніше виконував – Patrolling, Idling, Attacking або будь-який інший.

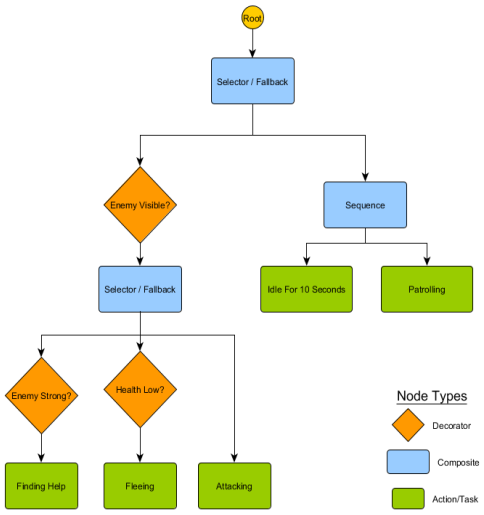

Рисунок 2.7 – Зображення дерева поведінки і його вузлів [7]

Для інтеграції звукового двигуна FMOD, його потрібно імпортувати в наш проект. Це можна зробити за допомогою вбудованого в Unity магазину з асетами – Asset Store. Щоб відкрити магазин асетів його потрібно викликати комбінацією Ctrl + 9, після цього в пошуку пишемо FMOD, знаходимо та імпортуємо в проект (рис. 2.8).

Після встановлення звуковий рушій потрібно налаштувати. Це можна виконати за допомогою меню налаштувань FMOD –> Edit Settings (рис. 2.9). Після цього потрібно вказати де знаходиться папка з проектом, щоб FMOD міг з ним взаємодіяти та зчитувати необхідні йому файли з розширенням .bank, в яких знаходиться всі музичні файли та їхні події.

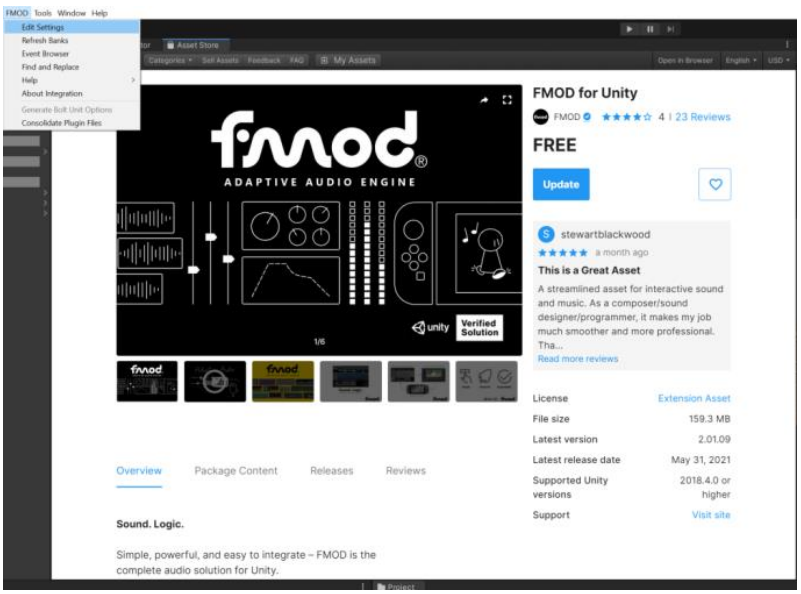

Рисунок 2.8 – Звуковий рушій FMOD в Asset Store [8]

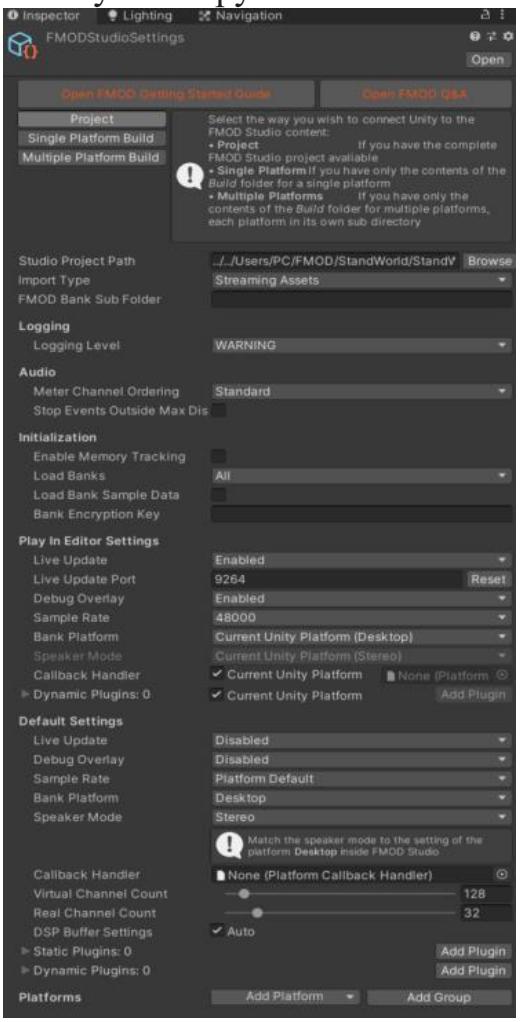

Рисунок 2.9 – Налаштування рушія FMOD

Після налаштування потрібно додати музику за допомогою FMOD Studio (рис. 2.10) та зберегти її в .bank файл за допомогою клавіши F7. Схему взагаємодії звукового рушія з рушієм Unity можна побачити на рисунку 2.11.

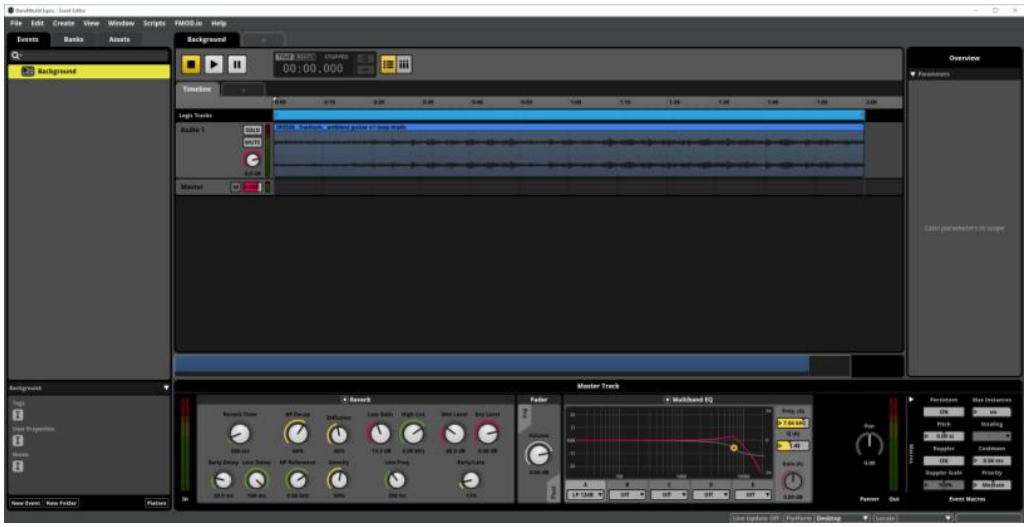

Рисунок 2.10 – Інтерфейс FMOD Studio [2]

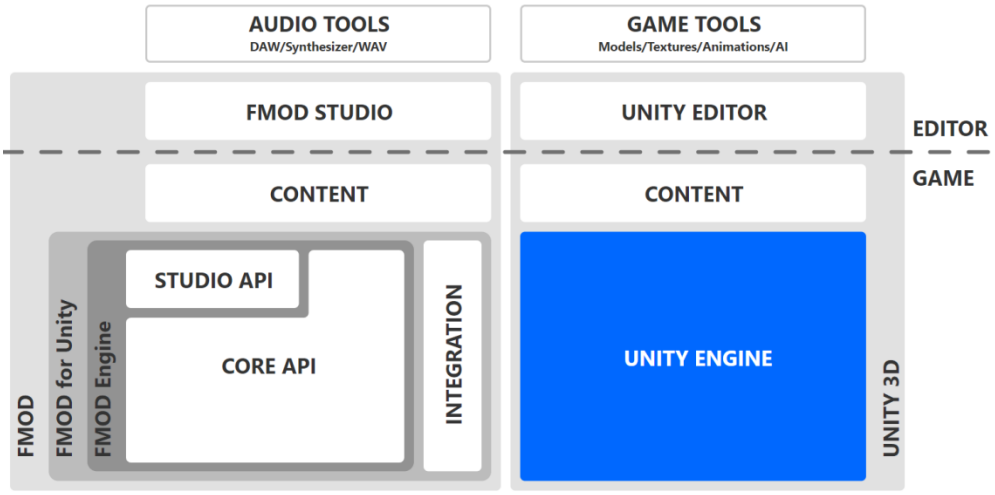

Рисунок 2.11 – Структура взаємодії FMOD з Unity

Висновки до розділу 2

У другому розділі було проаналізовано вимоги до розробки задачі. Було описано процес створеня шумових карт і їхня реалізація у грі. Проаналізував яким чином буде створюватися штучний інтелект і які інтрумети потрібні для його реалізації і розповів які функції буде мати у грі. Також було сказано як інтегрувати аудірушій FMOD для Unity та налаштувати.

РОЗДІЛ 3
РОЗРОБКА ШУМОВИХ КАРТ, ЛАНШАФТУ ТА ШТУЧНОГО ІНТЕЛЕКТУ

3.1 Практична реалізація об’єкта проектування

Першою задачою для реалізації проекту було створення шумових карт(карт Перлінга), що представлено у лістингу 3.1.

Лістинг 3.1 – Частина коду створення шумових карт

```
      using UnityEngine;

namespace StandWorld.MapGenerator
{
   public static class NoiseMap
    {
       public static float[] GenerateNoiseMap(Vector2Int size, int seed, float scale,
int octaves, float persistance, float lacunarity, Vector2 offset)
        {
         float[] noiseMap = new float[size.x * size.y];
         System.Random prng = new System.Random (seed);
         Vector2[] octaveOffsets = new Vector2[octaves];
         for (int i = 0; i < octaves; i++)
         {
            float offsetX = prng.Next (-100000, 100000) + offset.y;
            float offsetY = prng.Next (-100000, 100000) + offset.x;
            octaveOffsets [i] = new Vector2 (offsetX, offsetY);
         }

         if (scale <= 0)
         {
            scale = 0.0001f;
         }
         float maxNoiseHeight = float.MinValue;
  float minNoiseHeight = float.MaxValue;
         float halfWidth = size.x / 2f;
         float halfHeight = size.y / 2f;
```

кінець лістингу 3

Тепер в нас є шумова карта, база, на якій далі все буде будуватися. Ця карта надає нам основу для подальшого розвитку і планування. Завдяки цій карті ми можемо визначити області, присутній надмірний шум, і вжити його зменшення або усунення. Зокрема ланшафт, що представлено у лістингу 3.2.

Лістинг 3.2 – частина коду створення ланшафту

```
namespace StandWorld.MapGenerator
      {
        public class MapGenerator : MonoBehaviour
        {
          public enum DrawMode {NoiseMap, ColourMap};
          public DrawMode drawMode;
          public int mapWidth;
          public int mapHeight;
          public float noiseScale;
          public int octaves;
          [Range(0,1)]
          public float persistance;
          public float lacunarity;
          public int seed;
          public Vector2 offset;
          public bool autoUpdate;
          public TerrainType[] regions;
          private void Start()
          {
            if (autoUpdate)
            {
              Settings.lacunarity = lacunarity;
              Settings.octaves = octaves;
              Settings.persistance = persistance;
              Settings.noiseScale = noiseScale;
              Settings.offset = offset;
            }
```

кінець лістингу 3.2

Тепер у нас є ландшафт. Можна приступати до наповнення цього згенерованого світу людьми, тваринами та рослинністю.

Наповнюючи цей згенерований світ різноманітністю людей, тварин і рослин, ми створимо цікавий та живописний віртуальний ландшафт, де буде багато можливостей для дослідження, взаємодії та вигадки захоплюючих історій. Вся рослинність буде генеруватися на основі шумових карт, як і ландшафт.

Люди і тварини будуть створені на основі штучного інтелекту.

3.2 Тестування та налагодження інформаційно-комп’ютерної системи

Після запуску гри ми одразу попадаємо в головне меню (рис. 3.1), в ньому є велика кількість різних кнопок. Також в головному меню можна зайти в меню налаштувань (рис 3.2) та налаштувати розширення екрану чи гучність звуку.

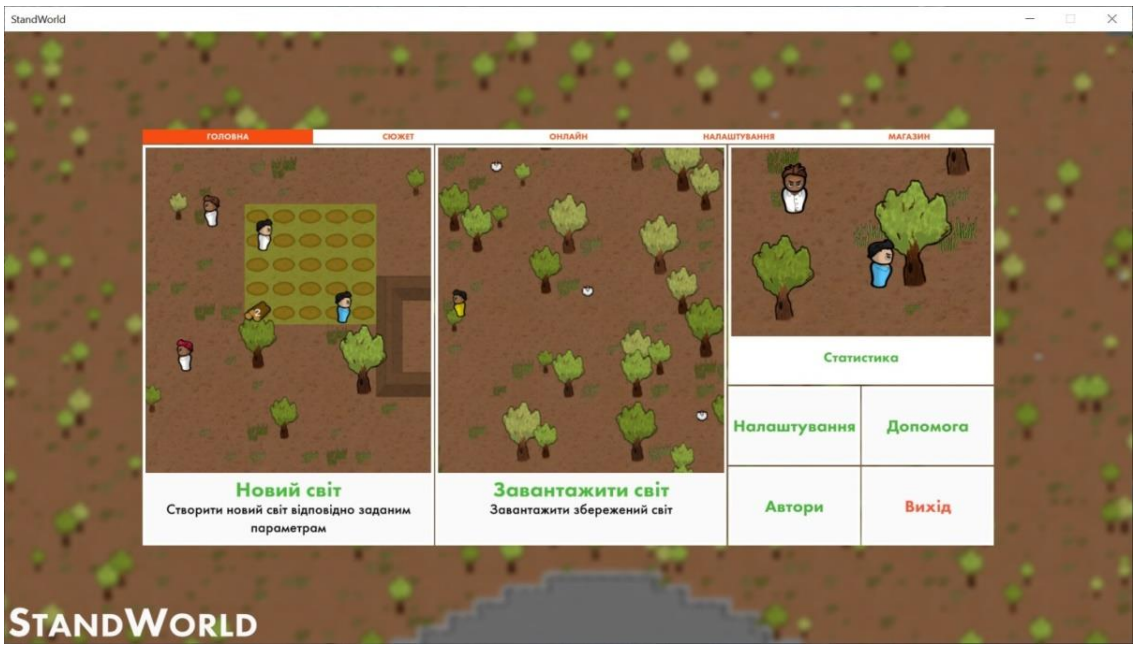

Рисунок 3.1 – Головне меню гри

Рисунок 3.2 – Меню налаштувань

Налаштувавши розширення екрану та звук, можна повернутися на головну та спробувати створити новий світ (рис. 3.3), після натискання на кнопку для створення нового світу потрібно вибрати розмір карти(за замовчуванням 300x300) та зерно світу (Seed).

Після конфігурації нового світу, нажавши на кнопку старт почнеться генерація світу та завантаження гри (рис. 3.4)

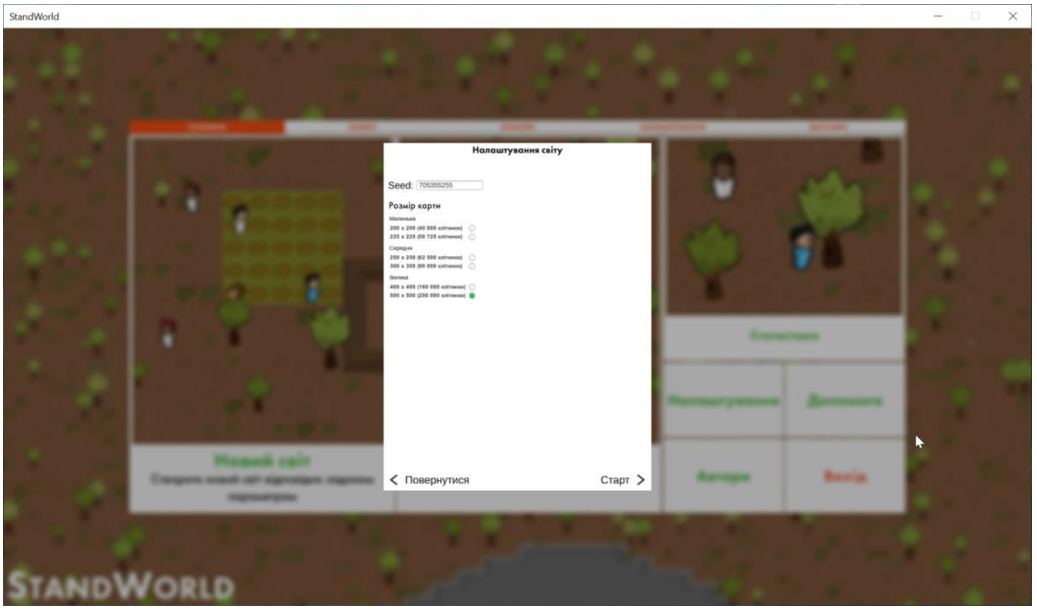

Рисунок 3.3 – Меню конфігурації нового світу

Після завантаження гри можна побачити таку картину (рис 3.5). Світ згенерувався, люди та тварини теж породилися. Після цього етапу можна сміливо приступати до гри. Наприклад при натисканні кнопки S на англійській розкладці клавіатури, з’явиться меню з зонами, вибравши та встановивши зону вирощування рослин, можна побачити таку картину (рис. 3.6). Люди скошують всю рослинність та на її місці саджають моркву, яку в подальшому можна зібрати.

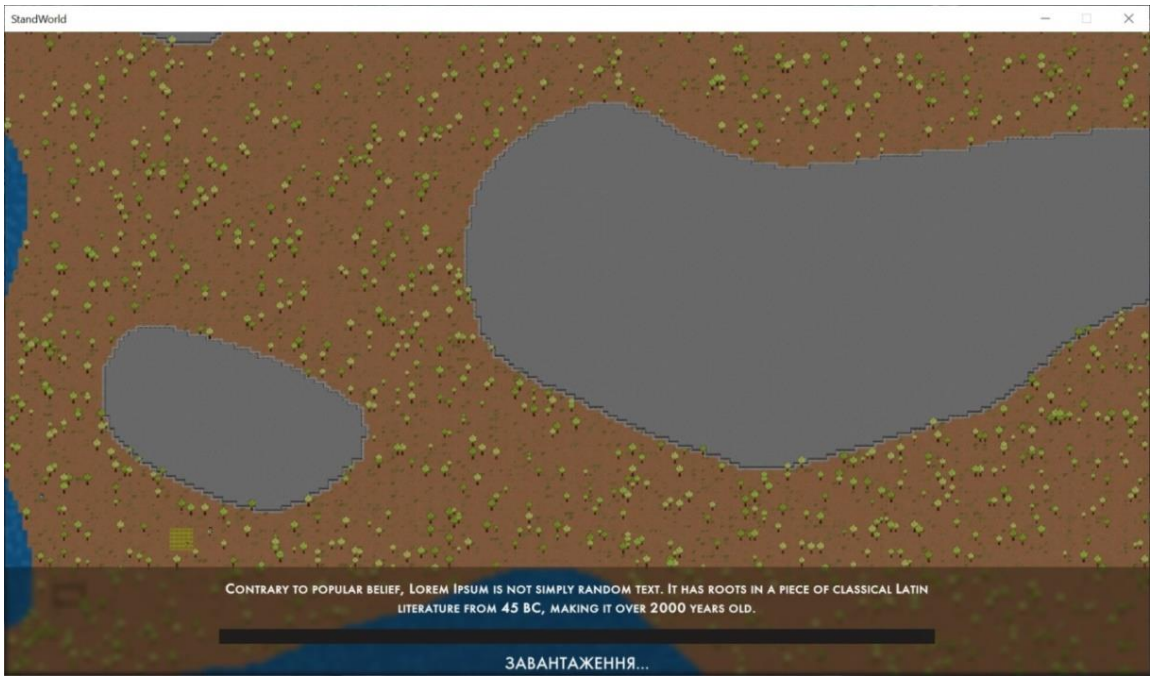

Рисунок 3.4 – Екран завантаження гри

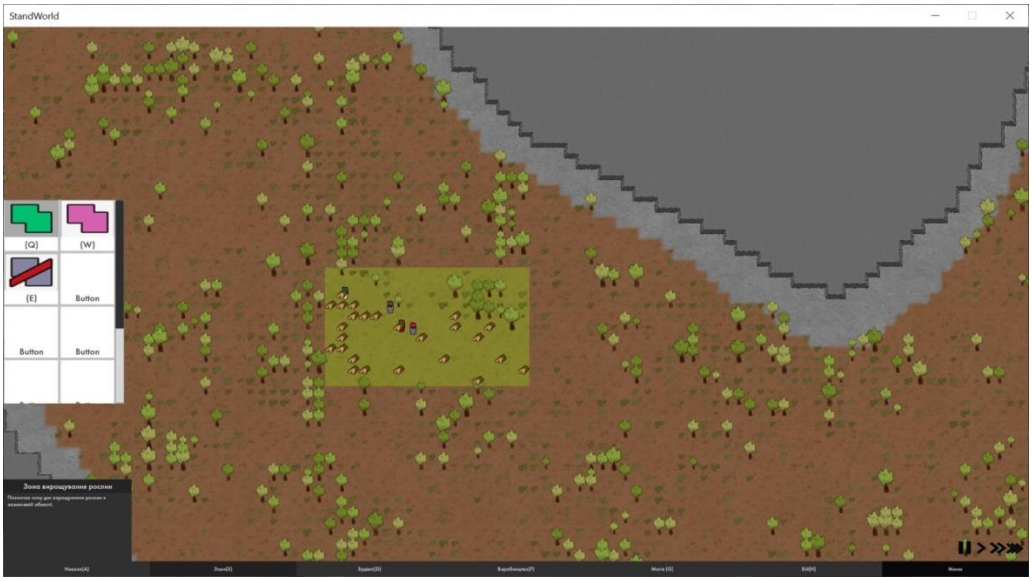

Рисунок 3.5 – Зона вирощування рослин

Люди можуть виконувати низку різних задач таких як: зрізати дерева, скосити траву (рис 3.6) зібрати врожай, будувати стіни або налаштовувати різні зони. Також в них є різні характеристики, вони можуть втомитися і лягти спати або просто зголодніти.

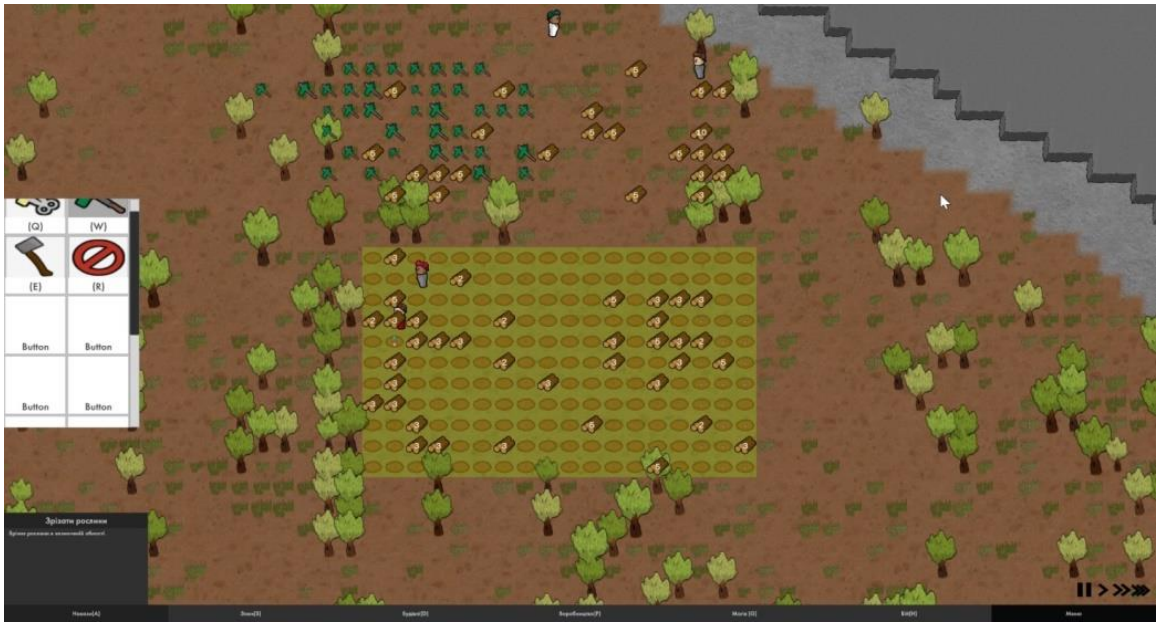

Рисунок 3.6 – Задача скосити траву

Всі рослини ростуть та вимирають з часом, наприклад морква в зоні вирощування, що зображено на рисунку 3.7. Також після вирубки дерев та збору врожаю отримуються ресурси. Результат процедурно згенерованого світу з горами, озерами та рослинністю можна побачити на рисунку 3.8.

Рисунок 3.7 – Демонстрація можливостей зони вирощування

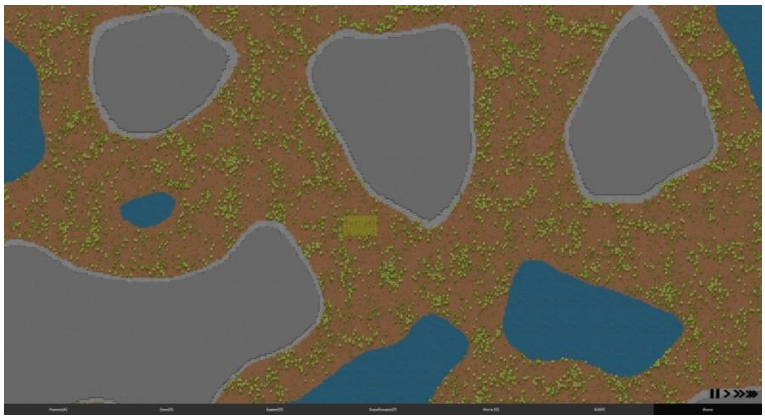

Рисунок 3.8 – Результат процедурної генерації світу

3.3 Економічне обґрунтування розробки

Метою даного розділу є обґрунтування доцільності надання послуги створення гри на рушії Unity.

Рушій надає можливість зменшити час та витрати на розробку завдяки своїй потужній інтегрованій середовищі розробки, готовим шаблонам та ресурсам, а також спрощеному процесу портінгу гри на різні платформи.

1) обчислюємо баланс робочого часу працівника. Фективний фонд часу розраховується за формулою;

$$Ф_{еф} = Ф_{ном} \times h, \quad (3.1)$$

де $F_{ном}$ – номінальний фонд робочого часу працівників, год.

h – плановий коефіцієнт витрат часу, приймається в межах 2 – 5%.

Припускаємо втрати часу 3%.

Номінальний фонд часу визначається за формулою:

$$Ф_{ном} = (Д_к – Д_{св} – Д_в – Д_{від} – Д_{п.св}) \times t_{зм\ 1} + Д_{п.св} \times t_{зм2}\ , \quad (3.2)$$

де $Д_к$ – кількість календарних днів в році, дн.;

$Д_{вс}$ – кількість вихідних і святкових днів в році, дн.;

$Д_{від}$ – середня кількість днів у році наданих робітнику під відпустку, дн;

$Д_{п.св}$ – передсвяткові, дн.;

$t_{зм1}, t_{зм2}$ – тривалість зміни у звичайні та передсвяткові дні, год.

Для розрахунку наведеної формули її дані визначаємо за 2023 р. і вони становлять:

– дні календарні – 365;

– вихідні дні – 104;

– святкові дні – 11;

– дні відпустки – 24;

– тривалість зміни – 8 годин, в передсвятковий день – 7 годин.

Звідси фонд номінальний дорівнює:

$$Ф_{ном} = (365 – 11 – 104 – 24) * 8 + 11 * 7 = 1797 \text{ год.}$$

Фонд ефективний становить:

$$Ф_{еф} = 1797 * 0.97 = 1743{,}09 \text{ год.}$$

2) Розраховуємо трудомісткість робіт з надання послуги створення гри на

рушії Unity. Спочатку визначають трудомісткість робіт щодо 1 послуги, тобто час за формулою;

$$T_{од} = \Sigma\ t\ н/год. \quad (3.3)$$

де $T_{од}$ – трудомісткість робіт щодо 1 послуги, н/год.;

$t_{н/год}$ – загальна сума норми часу згідно технологічного процесу, год.

Трудомісткість обчислюємо у таблиці 3.1.

Таблиця 3.1 – Трудомісткість робіт

Найменування операції	Час на 1 операцію, год.	Кількість операцій	Загальний час, год.
Написання дизайн документу	50	1	50
Написання ігрового коду	80	1	80
Створення 3D моделі	55	1	55
Інтегрування сторонніх SDK	15	1	15
РАЗОМ			200

Далі визначаємо загальну трудомісткість робіт, тобто всю кількість робочого часу, яку необхідно витратити на планову річну кількість послуг за формулою:

$$T_{посл} = T_{од} \times N, \quad (3.4)$$

де $T_{посл}$ – загальна трудомісткість робіт, н/год;

$T_{од}$ – трудомісткість 1 послуги, н/год.;

N – річна кількість планових послуг, од.

Приймаємо кількість послуг створення гри на рушії Unity – 5 од. в рік.

Звідси:

$$T_{посл.} = 200 * 5 = 1000 \text{ н/год.}$$

Припускаємо, що даний вид роботи не є єдиним на підприємстві і становить 15% від решти наданих послуг.

Отже на весь обсяг:

$$Т_{заг} = (1000 / 15) * 100 = 6666,66 \text{ н/год.}$$

3) Розрахунок необхідної кількості працюючих. Основних працівників підприємства визначаємо з розрахунку планової трудомісткості всіх наданих послуг фірми протягом планового року за формулою;

$$Чосн = \frac{Тзаг}{Феф \cdot Квн}, \quad (3.5)$$

де $Ч_{осн}$ – чисельність основних робітників, чол;

$Т_{заг}$ – загальна трудомісткість робіт на підприємстві, н/год;

$Ф_{еф}$ – ефективний фонд, год.;

$k_{вн}$ – коефіцієнт виконання норм, що приймаються рівним від 1 до 1,2. В нашому випадку припускаємо перевиконання на рівні 5 %, звідси $k_{вн}$ =1,05 .

Обчислюємо чисельність основних робітників для надання даної послуги:

$$Ч_{осн} = 6666,66 / (1743,09 * 1.1) = 4 \text{ чол.}$$

Крім того, приймаємо інші категорії персоналу згідно штатного розкладу у розмірі:

1) директор – 1 чол;

2) бухгалтер – 1 чол;

3) допоміжний робітник – 1 чол;

4) прибиральниця – 1 чол;

4) обчислення витрат на оплату праці.

Заробітна плата основних робітників визначається за відрядною

заробітною платою за формулою:

$$ЗП_{ор} = \Sigma P_{зв} \times N, \quad (3.6)$$

де $ЗП_{ор}$ – заробітна плата основних робітників, грн;

$P_{зв}$ – це звичайний розцінок за 1 послугу створення гри на рушії Unity., грн.

N – річна кількість послуг, од.;

Звичайний розцінок за 1 послугу створення гри на рушії Unity, що визначається за формулою:

$$P_{зв} = T_{од} \times T_{стс}, \quad (3.7)$$

де $T_{од}$ – трудомісткість робіт, н/год.;

$T_{стс}$ – тарифна ставка робітників, що зайняті у наданні цієї послуги, грн.

Згідно з законодавством на 1.01.20 мінімальна заробітна плата встановлена на рівні 6000 грн. Отже тарифну ставку 1 розряду приймаємо у сумі 36,11 грн. за 1 годину.

Середній коефіцієнт складності робіт з надання послуги приймаємо 1,54.

Визначаємо середню тарифну ставку працівника за формулою:

$$T_{стс} = T_{ст1} \times К_{с}, \quad (3.8)$$

де $T_{стс}$ – середня тарифна ставка працівника, грн.;

$T_{ст1}$ – тарифна ставка 1 розряду, грн..

$к_{с}$ – середній коефіцієнт складності робіт з послуги.

$$T_{стс} = 36,11 * 1,54 = 55,61 \text{ грн.}$$

Обчислюємо розцінок звичайний:

$$P_{зв} = 200 * 55,61 = 11\,122 \text{ грн.}$$

Обчислюємо заробітну плату основних робітників:

$$ЗП_{ор} = \ 11\ 122 * 5 = 55\ 610 \text{ грн.}$$

Отже прямий фонд оплати праці основних робітників, що надаватимуть послугу створення гри на рушії Unity за рік становить 55 610 грн. А на весь обсяг робіт фірми:

$$ЗП_{ор\ заг} = \ 55\ 610 / 15 * 100 = 370\ 733,33 \text{ грн.}$$

Заробітну плату іншим категоріям персоналу розраховуємо за формулою:

$$ЗП_i = О_i \times Ч_i \times М, \quad (3.9)$$

де $ЗП_i$ – заробітна плата інших категорій персоналу, грн.;

$О_i$ – оклад одного працівника певної категорії, грн.;

$Ч_i$ – чисельність працівників, чол.;

М – кількість місяців у році, од.

Премії визначаю за формулою:

$$,П = ЗП \times К_д, \quad (3.10)$$

де П – пермії, грн;

ЗП – пряма заробітна плата працівників, грн.;

Кд – коефіцієнт премії.

Річний фонд оплати праці за кожною категорією визначаємо за формулою:

$$ФЗП = ЗП + П \quad (3.11)$$

де ФЗП – фонд оплати праці, грн.;

ЗП – пряма заробітна плата, грн.;

П – премія, грн..

Нарахування на соціальні потреби містять: відрахування на соціальне страхування, у пенсійний фонд, фонд зайнятості, та інші фонди. На сьогодні єдиний соціальний внесок становить 22%. Цей відсоток, який є єдиним соціальним внеском, обов'язково сплачується підприємством у встановленому порядку.

Відрахування на соціальні потреби є важливою складовою фонду оплати праці, оскільки вони забезпечують соціальний захист працівників у разі хвороби, безробіття, а також забезпечують пенсійне забезпечення та інші соціальні виплати.

Розрахунки здійснюємо у зведеній відомості фонду оплати праці підприємства (табл. 3.2).

Таблиця 3.2 – Зведена відомість фонду оплати праці

Категорія персоналу	Оклад, грн.	Прямий ФЗП, грн.	Премії		ФЗП, грн.	Сума грн
			%	грн.		
Основні робітники	-	370 733,33	15	55 610	426 343,33	93 795,53
Допоміжний працівник	9000	108 000	5	5400	113 400	24 948
Директор	14 000	168 000	15	25 200	193 200	45 504
Бухгалтер*	8 000	64 000	10	6 400	70 400	15 488

Продовження таблиці 3.2

Прибиральниця*	600	3600	5	1800	37800	-
РАЗОМ	-	746 733,33	-	94 410	841 144,33	188 051,53

4) Розрахунок матеріальних витрат на послугу. Розрахунок по елементам кошторису матеріальних витрат здійснюємо за формулами:

$$МВ = \Sigma Ц_i \times n_i, \quad (3.12)$$

де МВ – матеріальні витрати, грн.;

$Ц_i$ – ціна купованого виробу або матеріалу, грн.,

n_i – кількість купованих виробів та матеріалів, од.

В нашому випадку матеріальні витрати відображені в табл. 3.3.

Таблиця 3.3 – Розрахунок матеріальних витрат

Найменування елементів	Джерело доступу	Ціна за одиницю, грн.	Сума, грн.
Інтегроване середовище розробки Rider *	https://www.jetbrains.com/ua-ua/rider/buy/	3 000	1 200
Набір для 3D моделювання Blender	https://www.blender.org/	Умовно безкоштовно	0
Разом:			200

Підприємство планує орендувати приміщення з виробничою площею

48м². Вартість оренди в місяць приймаємо: 21грн/м². Загальну вартість оренди в рік визначаємо за формулою:

$$В_{ор.} = S_{заг.} \times В_{ор.1м}^{2} \times М, \quad (3.13)$$

де $В_{ор}$ – загальгна вартість оренди, грн.;

$S_{заг.}$ – загальна площа орендованого приміщення підприємства, м^2;

$В_{ор.1м}^{2}$ – вартість оренди 1 м^2 в місяць, грн.;

М – кількість місяців, од.

$$В_{ор.} = 48 * 21 * 12 = 12\,096 \text{ грн.}$$

Визначаємо суму комунальних платежів. Щоб визначити кількість енергії та освітлення використовуємо формулу:

$$E_{осв} = \frac{S_{заг} \cdot Ф_{еф} \cdot N_{осв}}{1000 \cdot кВт} \quad (3.14)$$

де $Е_{осв}$ – кількість енергії освітлення, кВт/год;

$S_{заг.}$ – загальна площа орендованого приміщення підприємства, м^2;

$Ф_{еф}$ – ефективний фонд, год.;

$N_{осв}$ – це норма інтенсивності освітлення одного м² площі. Приймаємо – 40 Вт/м²

$к_{Вт}$ – коефіцієнт втрат електроенергії в мережі 0,9.

$$Е_{осв} = (48 * 1743{,}09 * 40) / (1000 * 0{,}9) = 3718{,}59 \text{ кВт/год.}$$

Вартість освітлення визначаємо за формулою:

$$В_{осв} = Е_{осв} \times Ц1, \quad (3.15)$$

де $В_{осв}$ – вартість освітлення, грн.;

Ц1 – ціна однієї кВт години, яка з 1.01.2021 р. встановлена 1,68 грн. за кВт•год.;

$$В_{осв}= 3718,53 * 1,68 = 6247,23 \text{ грн.}$$

Витрати на опалення визначаються за формулою:

$$В_{оп} = S_{заг.} \times Ц2 \times М, \quad (3.16)$$

де $В_{оп}$ – витрати на опалення, грн.;

$S_{заг.}$ – загальна площа, $м^2$;

Ц2 –вартість опалення одного $м^2$ площі, грн.

М – кількість опалюваних місяців у році, од.

$$В_{оп} = 48 * 42,9 * 6 = 12355,2 \text{ грн.}$$

Витрати на воду визначаються за формулою:

$$В_{в} = Ч \times Ц3 \times 0,5 \times М, \quad (3.17)$$

де $В_{в}$ – витрати на воду, грн.;

Ч – чисельність персоналу, чол..;

Ц3 – вартість за витрачання одного $м^3$ води, грн.;

Централізоване водопостачання та водовідведення – 33,94 грн. (з ПДВ)

0,5 $м^3$ – орієнтовна норма витрачання води на 1 працівника в місяць.

М – кількість місяців у році, од.

$$В_{в}= 8 * 33,94 * 0,5 * 12 = 1629,12 \text{ грн.}$$

Знаходимо амортизацію на кожен елемент основних фондів за формулою:

$$А = (В_{оф} \times Н_а) : 100, \quad (3.18)$$

де А – амортизація на кожен елемент основних фондів, грн.;

$Н_а$ – норма амортизації, %;

$В_{оф}$ – вартість основних фондів, грн.

У таблиці відображаються різні види витрат, які можуть бути враховані при розрахунку податкових відрахувань. Кожна категорія витрат має свої особливості щодо обліку та відрахувань. Наприклад, витрати на заробітну плату можуть бути віднесені на відрахування за рахунок визначення оподатковуваного доходу, а витрати на амортизацію можуть розподілятися.

Визначаємо відрахування за податковим методом у таблиці 3.4

Таблиця 3.4 – Розрахунок амортизаційних відрахувань

Категорія основних фондів	Вартість ОФ, грн.	% відрахувань	Амортизація, грн.
Обладнання	85 000	10	8500
Інструмент	3000	25	750
Інвентар	1000	27	270
РАЗОМ	89000	–	9520

Витрати на ремонт основних фондів приймаємо в розмірі 5% від вартості основних фондів. Розраховуємо витрати на ремонт:

$$В_{р.осн.ф.} = (89000 * 5) / 100 = 4450 \text{ грн.}$$

Обчислюємо витрати на охорону праці, які приймаємо у розмірі 150 грн.:

$$В_{ох.пр.} = 8 * 150 = 1200 \text{ грн.}$$

Інші витрати приймаємо в сумі 7% від всіх загальновиробничих витрат.

Всі елементи витрат зводимо у таблицю 3.5.

Таблиця 3.5 – Загальновиробничі витрати

Елементи витрат	Сума, грн.
1. Оренда	12096
2. Електроенергія	6247,23
3. Опалення	12355,2
4. Вода	1629,12
5. Амортизація	9520
6. Ремонт основних фондів	4450
7. Охорона праці	1200
Всього	47497,55

5) Визначення собівартості та ціни послуги створення гри на рушії Unity. Додаткові складові витрат можуть включати витрати на графічний дизайн, звукове оформлення, маркетинг та рекламу, тестування і поліпшення продукту, а також витрати на управління проектом і адміністративні витрати. Важливим етапом є також урахування потенційних ризиків та збитків, пов'язаних з розробкою гри.

Калькуляцію собівартості здійснюємо у таблиці 3.6.

Обчислюємо прибуток згідно планової норми рентабельності за формулою:

$$П = С \times k_р, \tag{3.19}$$

де П – прибуток, грн.;

С – собівартість однієї послуги, грн.,

$k_р$ – плановий коефіцієнт рентабельності.

Приймаємо плановий коефіцієнт рентабельності в розмірі 15 %.

Здійснюємо розрахунки:

$$П = (136\ 837{,}41 * 15) / 100 = 20\ 525{,}61 \text{ грн.}$$

Обчислюємо ціну гри на рушії Unity (без ПДВ) за формулою:

$$Ц_{опт} = С + П \quad . \tag{3.20}$$

Таблиця 3.6 – Розрахунок собівартості 1 послуги

Статті калькуляції собівартості	Сума, грн.	Примітки
1.Матеріальні витрати	1 200	З таблиці 3.3
2.Транспортні витрати	–	відсутні
3.Пряма ЗП	93 341,66	З таблиці 3.2 в розрахунку на 1 послугу
4. Премії	11 801,25	
5.Нарахування	23 506,44	
6.Загальновиробничі витрати	6 352,79	З таблиці 3.5 в розрахунку на 1 послугу
7.Позавиробничі витрати	635,27	10% від пункту 6
Повна собівартість	136 837,41	Сума пунктів 1–7

Здійснюємо розрахунки:

Ц = 136 837,41+ 20 525,61 = 156 363.02 грн.

Виходячи з розрахунків, проведених в даному розділі, можна зробити висновок, що вартість послуги створення гри на рушії Unity в середньому буде 156 363.02 грн.

На ринку комп’ютерних послуг України ціна на дану послугу може залежати від популярності гри та перегляду реклами вбудованої в гру за

допомогою сторонніх SDK, кожен перегляд реклами збільшує прибуток від гри. Якщо гра буде платна, то все залежить від кількості проданих копій, також площадки для розміщення ігор беруть комісію в 30% від заробітку гри.

Висновки до розділу 3

У даному розділі було проведено тестування та налагодження системи, що включало перевірку різних її елементів, таких як генерація світу, навігація гравця. Після тестування стало зрозуміло, що всі необхідні функції працюють справно і жодних дефектів не було виявлено.

Також було проведено аналіз економічної частини, метою якої було обґрунтувати доцільність надання послуг на рушієві Unity.

ВИСНОВКИ

Під час розгляду та аналізу різних існуючих тайлових систем в ігрових рушіях Unity було проведено огляд і аналіз різних тайлових систем. Були розглянуті різні тайлові системи, доступних в ігрових рушіях, зокрема систем, що базуються на сітці (Tilemap), розбитті на квадрати, шестикутники (Rule Tiles) та додатковий пакет із внутрішньої бібліотеки Unity 2D Extras.

До плюсів Tilemap входять: простий у використанні і швидка установка тайлів на сцені, інтегрований інструмент для створення і редагування карт прямо в Unity, підтримка функціональності колізій і тригерів для тайлів. До недоліків входять: обмежена гнучкість в створенні складних шаблонів та правил розміщення тайлів та функціональність анімації та спеціальних ефектів для тайлів. До плюсів Rule Tiles входять: автоматичне генерування тайлів на основі заданих правил, що спрощує створення складних середовищ, зменшує кількість необхідного ручного розміщення тайлів, підтримка анімації, зміни масштабу та обертання тайлів. Із мінусів: вимагає додаткового часу та зусиль для налаштування правил розміщення тайлів, може бути складним для розуміння і використання для новачків. Плюси 2D Extras: додатковий пакет функціональності, який розширює можливості тайлової системи Unity, підтримка анімації тайлів та функцій перетворення тайлів. До мінусів можна віднести додаткове вивчення та розуміння додаткових функцій та інтерфейсу. Аналіз підходів до тайлового мапування допомагає розуміти переваги та недоліки кожного підходу і визначити найбільш підходящий для конкретного проекту.

Для кожної із них були визначені їх переваги, недоліки та особливості та обрано оптимальний шлях для реалізації тайлових текстур з використанням шумових карт на базі Tilemap та їх генерації у самій грі.

Під час аналізу підходів до тайлового мапування було досліджено різні методики мапування тайлів, такі як звичайне мапування на двовимірній сітці (Grid-Based Collisions), використання тайлсетів на автоматичне мапування

згідно з правилами (Tile-Based Physics Engines) та полігональні колізії (Polygonal Collisions). Плюси Grid-Based Collisions: простий у використанні і реалізації, ефективний для простих 2D ігор з прямокутними тайлами, низьке навантаження на обчислювальні ресурси. Із мінусів: обмежена гнучкість для складних форм та розміщення об’єктів та важко реалізувати колізії з неточними формами. Плюси Polygonal Collisions: дозволяє створювати більш складні та деталізовані колізії, гнучкість у формі та розміщенні тайлів, можливість моделювання неточних форм об'єктів. Із мінусів: вимагає більше обчислювальних ресурсів, складніше у реалізації та обробці колізій, може викликати проблеми з точністю обчислень. Tile-Based Physics Engines плюси: інтеграція з потужними фізичними двигунами, які забезпечують реалістичну фізику об'єктів, можливість розрахунку колізій та фізики між тайлами та іншими об'єктами, гнучкість у встановленні фізичних властивостей для тайлів. До мінусів можна віднести потреби у великих обчислювальних ресурсів для обробки фізики і складність у налаштуванні та управлінні. Кожен підхід було проаналізовано щодо його ефективності, гнучкості та витрат ресурсів.

При вивчені системи колізії було досліджено різні методики обробки колізії у тайлових системах обрано для реалізації саме Grid-Based Collisions.

При оптимізації продуктивності вивчено та впроваджено різні оптимізаційні методи для покращення продуктивності тайлової системи, це включало в себе оптимізацію алгоритмів розміщення та відображення тайлів, використовування кешування даних, видалення непотрібних об’єктів. Якщо виконується складний розрахунок або обробка даних для відображення тайлів, розглянуті можливісті кешування результатів цих обчислень. Це дозволяє уникнути повторних обчислень, що покращує продуктивність. Якщо виконується складний розрахунок або обробка даних для відображення тайлів, можна розглянути можливість кешування результатів цих обчислень. Це дозволяє уникнути повторних обчислень, що покращує продуктивність. В результаті виконаних дій з оптимізації продуктивності, було досягнуто покращення швидкодії та ефективності тайлової системи генерації рівня.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Deals T. Learn game design from experienced professionals for under $40. TechSpot. URL: https://www.techspot.com/news/76276-learn-game-design-experienced-professionals-under-40.html (дата звернення: 17.03.2023).
2. URL: https://www.fmod.com/studio (дата звернення: 17.03.2023).
3. Tynan Sylvester. RimWorld Launch Trailer, 2020. YouTube. URL: https://www.youtube.com/watch?v=3tDrxOASUog (дата звернення: 17.03.2023).
4. Tiles – Vector Maps API | MyPTV. Location Service APIs to solve vehicle routing problems | PTV Developer | MyPTV. URL: https://developer.myptv.com/en/documentation/vector-maps-api/concepts/tiles-vector-maps-api (дата звернення: 17.03.2023).
5. Brackeys. PERLIN NOISE in Unity – Procedural Generation Tutorial, 2022. YouTube. URL: https://www.youtube.com/watch?v=bG0uEXV6aHQ (дата звернення: 17.03.2023).
6. Make Perlin noise map generation a little more detailed?. Unity Forum. URL: https://forum.unity.com/threads/make-perlin-noise-map-generation-a-little-more-detailed.1281584/ (дата звернення: 14.04.2023).
7. Perlin Noise: Part 2. Scratchapixel 3.0, Learn Computer Graphics Programming. URL: https://www.scratchapixel.com/lessons/procedural-generation-virtual-worlds/perlin-noise-part-2/perlin-noise.html (дата звернення: 17.03.2023).
8. Sebastian Lague. Procedural Landmass Generation (E02: Noise Map), 2019. YouTube. URL: https://www.youtube.com/watch?v=WP-Bm65Q-1Y (дата звернення: 18.03.2023).
9. Sebastian Lague. Procedural Landmass Generation (E03: Octaves), 2016. YouTube. URL: https://www.youtube.com/watch?v=MRNFcywkUSA (дата звернення: 11.05.2023).
10. Unity Asset Store – The Best Assets for Game Making. Unity Asset Store – The Best Assets for Game Making. URL: https://assetstore.unity.com/ (дата звернення: 18.04.2023)

11. Unity – Scripting API: Mathf.PerlinNoise. Unity – Manual: Unity User Manual 2021.3 (LTS). URL: https://docs.unity3d.com/ScriptReference/Mathf.PerlinNoise.html (дата звернення: 17.03.2023).

12. Unity. Unity – Manual: Unity User Manual 2021.3 (LTS). URL: https://docs.unity3d.com/ru/530/Manual/terrain-Textures.html (дата звернення: 12.04.2023).

13. Unity. Unity – Manual: Unity User Manual 2021.3 (LTS). URL: https://docs.unity3d.com/ru/530/Manual/terrain-Textures.html (дата звернення: 12.04.202

14. Dave / GameDevelopment. FULL 3D ENEMY AI in 6 MINUTES! || Unity Tutorial, 2020. YouTube. URL: https://www.youtube.com/watch?v=UjkSFo (дата звернення: 17.03.2023).

15. Code Monkey. Simple Enemy AI in Unity (State Machine, Find Target, Chase, Attack), 2020. YouTube. URL: https://www.youtube.com/watch?v=db0KWYaWfeM (дата звернення: 10.03.2023).

16. Brackeys. ENEMY AI – Making an RPG in Unity (E10), 2017. YouTube. URL: https://www.youtube.com/watch?v=xppompv1DBg (дата звернення: 11.03.2023).

17. Learn To Create Enemy AI Systems With A Few Lines Of Code In Unity Game Engine. Anyone Can Learn To Make Games. URL: https://awesometuts.com/blog/unity-enemy-ai/ (дата звернення: 17.03.2023).

18. Vionix. How to make enemy AI in Unity – VionixStudio. VionixStudio. URL: https://vionixstudio.com/2021/08/17/make-enemy-ai-in-unity/ (дата звернення: 17.03.2023).

ДОДАТКИ

Додаток А

Лістінг 1 – Приклад програмного коду для генерації шуму карт

```
using UnityEngine;

namespace StandWorld.MapGenerator
{
  public static class NoiseMap
   {
     public static float[] GenerateNoiseMap(Vector2Int size, int seed, float scale, int
octaves, float persistance, float lacunarity, Vector2 offset)
      {
       float[] noiseMap = new float[size.x * size.y];

       System.Random prng = new System.Random (seed);
       Vector2[] octaveOffsets = new Vector2[octaves];
       for (int i = 0; i < octaves; i++)
       {
         float offsetX = prng.Next (-100000, 100000) + offset.y;
         float offsetY = prng.Next (-100000, 100000) + offset.x;
         octaveOffsets [i] = new Vector2 (offsetX, offsetY);
       }

       if (scale <= 0)
       {
         scale = 0.0001f;
       }

       float maxNoiseHeight = float.MinValue;
       float minNoiseHeight = float.MaxValue;

       float halfWidth = size.x / 2f;
       float halfHeight = size.y / 2f;


       for (int y = 0; y < size.y; y++)
       {
         for (int x = 0; x < size.x; x++)
         {

           float amplitude = 1;
           float frequency = 1;
           float noiseHeight = 0;

           for (int i = 0; i < octaves; i++)
           {
             float sampleX = (x-halfWidth) / scale * frequency + octaveOffsets[i].x;
             float sampleY = (y-halfHeight) / scale * frequency + octaveOffsets[i].y;
```

```
                float perlinValue = Mathf.PerlinNoise (sampleX, sampleY) * 2 – 1;
                noiseHeight += perlinValue * amplitude;

                amplitude *= persistance;
                frequency *= lacunarity;
            }

            if (noiseHeight > maxNoiseHeight)
            {
                maxNoiseHeight = noiseHeight;
            }
            else if (noiseHeight < minNoiseHeight)
            {
                minNoiseHeight = noiseHeight;
            }

            noiseMap [y + x * size.x] = noiseHeight;
        }
    }

    for (int y = 0; y < size.y; y++)
    {
        for (int x = 0; x < size.x; x++)
        {
            noiseMap [y + x * size.x] = Mathf.InverseLerp (minNoiseHeight, maxNoiseHeight,
noiseMap [y + x * size.x]);
        }
    }

    return noiseMap;
  }
 }
}
```

Кінець лістингу 1

Лістинг – Схема дерева поведінки

```
using StandWorld.Characters.AI.Jobs;
using StandWorld.Definitions;
using UnityEngine;

namespace StandWorld.Characters.AI
{
 public enum TaskState
 {
  Running
```

```
  Success,
   Failed,
}

public enum TaskType
{
   Idle,
   Sleep,
   Eat,
   Cut,
   Harvest,
   Sow,
   Dirt,
   HaulRecipe,
}

public class Task
{
   public TargetList targets;
   public TaskState state;
   public TaskDef def;
   public TaskBase taskBase;
   public int ticksToPerform;

   public Task(TaskDef def)
   {
      this.def = def;
      ticksToPerform = def.ticksToPerform;
   }

   public Task(TaskDef def, TargetList targets) : this(def)
   {
      this.targets = targets;
      ticksToPerform = def.ticksToPerform;
   }

   public Task(TaskDef def, TargetList targets, int ticksToPerform) : this(def, targets)
   {
      this.ticksToPerform = ticksToPerform;
   }

   public Task(TaskDef def, int ticksToPerform) : this(def)
   {
      this.ticksToPerform = ticksToPerform;
   }

   public void GetTaskClass(BaseCharacter character)
   {
      switch (def.taskType)
      {
         case TaskType.Cut:
```

```
                taskBase = new TaskCut(character, this);
                break;
            case TaskType.Dirt:
                taskBase = new TaskDirt(character, this);
                break;
            case TaskType.Sow:
                taskBase = new TaskSow(character, this);
                break;
            case TaskType.Eat:
                taskBase = new TaskEat(character, this);
                break;
            case TaskType.Sleep:
                taskBase = new TaskSleep(character, this);
                break;
            case TaskType.Idle:
                taskBase = new TaskIdle(character, this);
                break;
            case TaskType.HaulRecipe:
                taskBase = new HaulRecipeJob(character, this);
                break;
        }
    }
```

Кінець лістингу 2