НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп‘ютерні системи та мережі ”

спеціальності 123 “Компьютерна інженерія”

на тему: Графічний модуль гри “Гетьманщина 1654”

Виконав : студент __4__ курсу, групи _ІВ-91_
(шифр групи)

_____Микитенко Всеволод Олегович_____
(прізвище, ім’я, по батькові) (підпис)

Керівник ___ст. викладач, Виноградов Ю. М._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) ст. викладач, Виноградов Ю. М _
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

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

Київ – 2023 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп‘ютерні системи та мережі ”

спеціальності 123 “Комп’ютерна інженерія ”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2023 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Микитенко Всеволода Олеговича

1. Тема проєкту *Графічний модуль гри “Гетьманщина 1654”*
керівник проєкту Виноградов Ю. М, ст. викладач _____.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 31 травня 2023 року №2101-с

2. Термін здачі студентом закінченого проєкту *5 червня 2023 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Огляд і класифікація комп’ютерних ігор та їх графіки.

Розділ 2. Огляд ігрових рушіїв, алгоритмів та технологій для розробки системи.

Розділ 3. Деталі розробки системи.

Розділ 4. Дослідження та аналіз розробленої системи.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) Блок графічний (структурна схема), Блок графічний, діаграма класів (функціональна схема), Блок графічний, алгоритм дій програмного забезпечення (принципова схема).

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	ст. викладач, Виноградов Ю. М		

7. Дата видачі завдання «21» листопада 2022 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	21.11.2022 – 22.11.2022	
2.	*Вивчення та аналіз завдання*	23.11.2022 – 29.12.2022	
3.	*Розробка архітектури та загальної структури системи*	30.12.2022 – 07.01.2023	
4.	*Розробка структур окремих підсистем*	08.01.2023 – 15.01.2023	
5.	*Програмна реалізація системи*	16.01.2023 – 30.04.2023	
6.	*Оформлення пояснювальної записки*	15.05.2023 – 03.06.2023	
7.	*Захист програмного продукту*	05.06.2023	
8.	*Передзахист*	09.06.2023	
9.	*Захист*	21.06.2023	

Студент-дипломник _____ Всеволод МИКИТЕНКО
(підпис)

Керівник проєкту _____ Юрій ВИНОГРАДОВ
(підпис)

АНОТАЦІЯ

В бакалаврській дипломній роботі метою було створення графічного модуля для комп’ютерної гри «Гетьманщина 1654». Для досягнення мети було проаналізовано жанрову класифікацію ігор, проаналізовані правила настільного варіанта гри, а також були досліджені різні середи розробки ігор та ігрові рушії. На основі цього було реалізовано прототип ігрового поля. У ході розробки були написані скрипти, необхідні для роботи гри, та створенні або запозиченні графічні матеріали.

Для розробки відповідної програми використано ігровий рушій Unity 2021, що використовує мову C# та засобом програмування алгоритму є середа розробки JetBrains Rider.

ANNOTATION

In the bachelor thesis, the goal was to create a graphic module for the computer game "Hetmanate 1654". To achieve the goal, the genre classification of games was analyzed, the rules of the table version of the game were analyzed, as well as various game development environments and game engines were also researched. Based on this, a prototype of the playing field was implemented. In the course of the development, the scripts necessary for the operation of the game were written, and graphic materials were created or borrowed.

The game engine Unity 2021 using the C# language was used to develop the corresponding program, and the JetBrains Rider development environment was used to program the algorithm.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Графічний модуль гри	3		
			“Гетьманщина 1654”			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Графічний модуль гри	68		
			“Гетьманщина 1654”			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Графічний модуль гри	1		
			“Гетьманщина 1654”			
			Блок графічний			
			(структурна схема)			
	А4	***ІАЛЦ.4672008.005 Д2***	Графічний модуль гри	1		
			“Гетьманщина 1654”			
			Блок графічний, діаграма			
			класів			
			(функціональна схема)			
	А4	***ІАЛЦ.467200.006 Д3***	Графічний модуль гри	1		
			“Гетьманщина 1654”			
			Блок графічний, алгоритм			
			дій програмного забезпечення			
			(принципова схема)			
	А4	***ІАЛЦ.467200.007 Д4***	Графічний модуль гри	15		
			“Гетьманщина 1654”			
			Текст програмного коду			

					ІАЛЦ.467200.001 ОА			
Зм	*Лист*	*№ докум.*	*Підп*	*Дата*				
Розроб		Микитенко В.О.			*Графічний модуль гри “Гетьманщина 1654”* Опис альбому	Літ.	Аркуш	Аркушів
Перев		Виноградов Ю.М.					1	1
						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-91		

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Графічний модуль гри “Гетьманщина 1654”»*

Київ – 2023

ЗМІСТ

1. Найменування та область застосування

Дане технічне завдання стосується створення графічного модуля для комп'ютерної гри шляхом перенесення правил настольної гри «Гетьманщина 1654». Прототип гри, що розробляється, може бути використаний для загального користування або у статтях та інших проектах з посиланням на даний дипломний проект.

2. Підстави для розробки

Підставою для розробки є завдання на виконання бакалаврської дипломної роботи, затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут» імені Ігоря Сікорського.

3. Мета та призначення розробки

Метою розробки є графічний модуль прототиу гри «Гетьманщина 1654».

4. Джерела розробки

Джерелами розробки є науково-технічна література, технічна документація, публікації та статті у мережі Інтернет.

5. Технічні вимоги

Оскільки для розробки проекту був обраний ігровий рушій Unity, а для написання і редагування програмного коду використовується JetBrains Rider, то знадобиться обладнання, що володіє технічними характеристиками не нижче мінімально необхідних для коректної роботи даних засобів розробки. Тому для реалізації проекту був використаний ноутбук DELL Inspirion 5559 з наступними технічними характеристиками:

- Операційна система Windows 10;
- Процесор Intel Core i5-U6200, 2.4Ghz;
- Об'єм оперативної пам'яті 4 GB
- Відеокарта AMD Radeon M335 4 GB VRAM.

6. Етапи розробки

Назва етапів виконання	Термін виконання
Затвердження теми роботи	21.11.2022 – 22.11.2022
Вивчення та аналіз завдання	23.11.2022 – 29.12.2022
Розробка архітектури та загальної структури системи	30.12.2022 – 07.01.2023
Розробка структур окремих частин системи	08.01.2023 – 15.01.2023
Програмна реалізація системи	16.01.2023 – 30.04.2023
Виправлення помилок	15.05.2023 – 03.06.2023
Оформлення пояснювальної записки	05.06.2023
Захист дипломної бакалаврської роботи	21.06.2023

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Графічний модуль гри "Гетьманщина 1654"*»

Київ – 2023

ЗМІСТ

ВСТУП

У сучасному світі комп'ютерні ігри поступово займають все більш вагоме й впливове місце серед безлічі форм розваги, доступних нам сьогодні. Щохвилини прогресуючий технологічний розвиток та нескінченна творча свобода розробників дарують цим іграм нові горизонти, перетворюючи їх не просто на джерело розваги, але й на потужний засіб спілкування та взаємодії між гравцями. Комп'ютерні ігри відкривають нам широкі простори, у які ми можемо зануритися, відчути себе активними учасниками захоплюючих світів, віртуальних пригод і неповторного фантастичного досвіду.

Одним з ключових елементів комп'ютерних ігор є їх графічна складова. Вона є основою для створення візуального світу гри, який надає гравцеві можливість відчути атмосферу та реалістичність віртуального оточення. Графіка визначає якість зображень, деталізацію об'єктів, анімацію персонажів і ефекти, що роблять гру живою та захоплюючою. Вона передає неймовірну деталізацію, реалістичність об'єктів та середовища, а також викликає емоції та підсилює іммерсію гравців.

Завдяки постійному розвитку технологій комп'ютерної графіки, сьогодні ми можемо насолоджуватися вражаючою якістю графічних ефектів, які надають іграм реалістичний вигляд та відтворюють фізичні закони.

Графічна складова в комп'ютерних іграх є важливим фактором, що визначає їх успіх та популярність. Якщо гра не вражає гравців своїм візуальним виконанням, шанси на успіх значно зменшуються. Гравці все більше очікують від ігор захоплюючих графічних світів, які забезпечують їм відчуття реалізму та іммерсії.

Дана робота пропонує приклад створення основного графічного модуля для стратегічної настільної гри під назвою «Гетьманщина 1654».

РОЗДІЛ 1. Огляд і класифікація комп’ютерних ігор та їх графіки

1.1 Загальна характеристика комп’ютерних ігор

Комп'ютерна гра є програмою, яка організовує ігровий процес і може взаємодіяти з користувачем або виступати як його віртуальний партнер. Вона створює імітацію взаємодії у віртуальному просторі між ігровим персонажем та користувачем (або групою користувачів) за допомогою певного алгоритму. Гравець впливає на ігрову ситуацію на екрані монітора за допомогою клавіатури, миші, джойстика тощо.[1]

Багато комп'ютерних ігор базуються на сторонніх джерелах, таких як книги або фільми, що дозволяє розширити світ гри і привернути увагу шанувальників оригінального матеріалу. Цей підхід сприяє залученню нових аудиторій та викликає емоційний зв'язок з уже знайомими персонажами та уявним світом.

Крім того, в деяких випадках розробники ігор створюють додаткові матеріали на базі відомих ігрових серій, що розширюють їх світи й внесли глибше розуміння історії та персонажів. Це дозволяє фанатам ще більше поглибитися у улюблену гру та відкрити нові аспекти її всесвіту.

Не лише розвага, але й навчання відіграють важливу роль у світі комп'ютерних ігор. Спеціально розроблені навчальні ігри використовуються як ефективний інструмент для засвоєння знань та навичок. Вони сприяють активному навчанню, розвивають критичне мислення, логіку та співпрацю, стимулюючи ігровий процес для досягнення певних цілей.

Крім того, комп'ютерні ігри стали платформою для проведення кіберспортивних змагань. Змагання різного масштабу, від регіональних до світових, привертають увагу мільйонів глядачів та учасників, утворюючи

активну та конкурентну спільноту кіберспорту. Це відкриває нові можливості для професійного геймінгу та розвитку геймерської культури.

Вплив комп'ютерних ігор на сучасне суспільство настільки значний, що спостерігається стійка тенденція до гейміфікації в неігрових прикладних програмах. Гейміфікація, тобто використання елементів гри, таких як досягнення, рейтинги та конкурси, у навчанні, здоров'ї, бізнесі та інших сферах, стає ефективним інструментом мотивації та залучення користувачів.

Це свідчить про те, що комп'ютерні ігри вже не лише розважальні продукти, а й важливі інструменти для навчання, дослідження та взаємодії. Їх вплив на суспільство і культуру продовжує зростати, а майбутнє цієї галузі відкриває нові можливості і досягнення.

Існує загальна згода серед розробників комп'ютерних ігор щодо основних жанрів, хоча класифікація їх не є однозначною через розбіжності у критеріях жанрів у різних джерелах. Основний критерій поділу жанрів базується на типових діях, які здійснюються в іграх. Ігри поділяються на три великі групи: ігри контролю, ігри дій та ігри інформації. Хоча в межах однієї групи можуть бути схожості, вони також мають відмінності, що робить кожну гру унікальною.[2]

У загальному, будь-яка гра складається з 15 основних геймплейних елементів: навчання, загадки, спілкування, роль, вивчення, збирання, ухилення, знищення, змагання, техніка, турбота, розвиток, контроль, тактика, планування. Ці елементи поділені на 3 класифікації (Табл. 1.1).

Таблиця 1.1 – Класифікація комп’ютерних ігор по жанрам.

Категорія	Ігри інформації	Ігри дій	Ігри контролю
Гібридні жанри	• Action-RPG • Roguelike	• MMOFPS • Survival	• RTS • MOBA
5 елементів	• Open RPG	• Open Action	• Global Strategy

Продовження Таблиці 1.1

Категорія	Ігри інформації	Ігри дій	Ігри контролю
3 елемента	• RPG • MUD MMORPG	• Action • Slasher • Battle Racing	• Strategy • Sim Strateg
2 елемента	• Puzzle • Quest • Browse RPG • Adventure	• Platformer • Stealth-Action • Fighting • Racing	• Economical • Tower Defense • Wargame • Cardgame
1 елемент	• Education • Test • Contact • Hero • Toure	• Arcade • Horror • Shooter • Sport • Simulator	• Logic • Tactic • MicroControl • Building • Life Sim
Елементарні жанри	1.1. Навчання 1.2. Загадки 1.3.Спілкування 1.4. Роль 1.5. Вивчення	2.1. Збирання 2.2. Ухилення 2.3. Нищення 2.4. Змагання 2.5. Водіння	3.1. Турбота 3.2. Створення 3.3. Контроль 3.4. Тактика 3.5. Планування

Ігри інформації – тип комп'ютерних ігор, де основний акцент робиться на передачі інформації гравцеві. Ці ігри часто включають в себе головоломки, головним завданням яких є розв'язання різних завдань та логічних головоломок. Вони можуть бути базовими ігровими елементами, які допомагають гравцю отримати нові знання, виконувати різні завдання з орієнтацією в інформаційному середовищі або розвивати навички аналізу та логічного мислення. Ці ігри забезпечують не тільки розвагу, але і можуть мати освітній аспект, сприяючи навчанню та розширенню когнітивних здібностей гравця.

Ігри дії – тип комп'ютерних ігор, де основний фокус зосереджений на фізичних взаємодіях гравця з віртуальним світом. Вони включають в себе швидкість, реакцію, стратегію та координацію рухів. Головні завдання

полягають у боротьбі зі ворогами, виконанні місій, подоланні перешкод та досягненні цілей. Гравець взаємодіє з грою за допомогою реакційних рухів, стрілянини, бігу, скакання та інших активних дій. Ігри дії надають можливість відчути адреналін, динаміку та захоплення, дозволяючи гравцю стати героєм своєї власної пригоди.

Ігри контролю – тип комп'ютерних ігор, в яких головна мета полягає у плануванні подій і управлінні з метою отримання переваги в майбутньому. Цей жанр охоплює широкий спектр стратегічних ігор, економічних симуляцій, варгеймів і тактичних проектів. Він також включає "Strategy" – типову локальну стратегію, яка є золотою серединою цієї групи. У таких іграх гравці повинні розробляти ефективні стратегії, приймати рішення щодо алокації ресурсів, планувати ходи і прогнозувати наслідки своїх дій для досягнення перемоги. Ігри контролю розвивають аналітичне мислення та навички стратегічного планування.

1.2 Огляд і аналіз графічної складової компьютерних ігор

Далі буде розглянуто різні способи візуалізації ігрового процесу, зокрема текстові, 2-D, 3-D і VR. Буде проведено порівняння між цими способами з метою виявлення їх особливостей та впливу на ігровий досвід.[3]

1.2.1 Текстовий спосіб візуалізації

Текстовий спосіб візуалізації комп'ютерних ігор: магія словесного світу.

В світі комп'ютерних ігор візуалізація є одним з ключових елементів, що приваблюють та захоплюють гравців. Традиційно, графічна та аудіовізуальна складові використовуються для створення реалістичного та привабливого ігрового досвіду. Однак, існує ще один спосіб візуалізації, який може бути не менш захоплюючим і захоплюючим – текстовий спосіб.

Текстовий спосіб візуалізації комп'ютерних ігор базується на використанні тексту для передачі інформації, створення наративу та візуальності гри. Він використовується для опису об'єктів, персонажів, локацій, подій та інших елементів гри, дозволяючи гравцеві уявити унікальний світ гри та зануритися в нього. Через текстові описи гравець може побачити вигляд та особливості різноманітних об'єктів, відчути атмосферу різних локацій та взаємодіяти з персонажами через діалоги та логи.

Текстовий спосіб візуалізації гри має свої особливості та переваги порівняно з іншими способами візуалізації. По-перше, він дозволяє створювати глибину сюжету та персонажів. Детально описані діалоги, думки та емоції персонажів допомагають розкрити їхні характери, внутрішні конфлікти та мотивацію. Гравець може відчути емоційне зв'язок з персонажами та бути втягнутим у їхні історії.

По-друге, текстовий спосіб візуалізації дозволяє гравцеві власноручно уявити світ гри та його деталі. За допомогою тексту, гравець може створювати свої унікальні образи персонажів та об'єктів, відтворювати локації у своїй уяві та розширювати світ гри. Це дає гравцеві велику свободу творчості та особистого сприйняття гри.

По-третє, текстовий спосіб візуалізації може бути особливо ефективним у рольових іграх, де глибина сюжету та історії мають велике значення. Гравець може зануритися у довгі діалоги, вибирати варіанти відповідей та впливати на розвиток сюжету. Важливі події та рішення можуть бути описані через текст, надаючи гравцю можливість зануритися у світ гри та відчути його значення. (Рис 1.1)

В заключенні, текстовий спосіб візуалізації комп'ютерних ігор відіграє важливу роль у передачі інформації, створенні наративу та візуальності гри. Він дозволяє гравцеві власноручно утворювати образи та уяву про світ гри, розкриває глибину сюжету та персонажів, та надає велику свободу творчості та особистого сприйняття гри. Текстовий спосіб візуалізації комп'ютерних ігор –

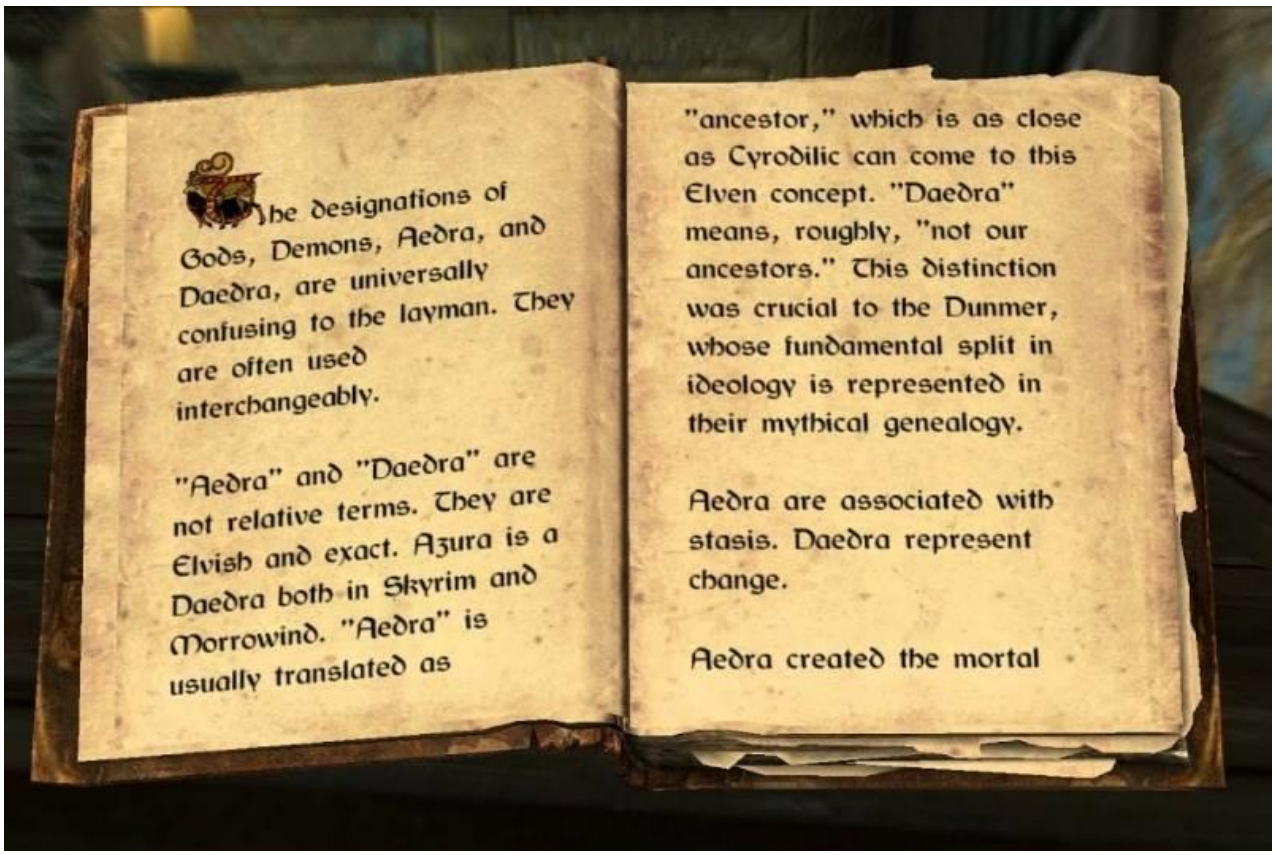


Рисунок 1.1 – Приклад подання історії гри TES:Skyrim через текст.

це магія словесного світу, яка здатна занурити гравця у неповторну атмосферу та створити незабутній ігровий досвід.

1.2.2 Спосіб візуалізації через 2-D графіку

Спосіб візуалізації через 2-D графіку є одним із найпоширеніших у комп'ютерних іграх. Він використовує двовимірні графічні елементи, такі як спрайти, текстури та фони, для створення візуального середовища гри. Порівняно з текстовим способом візуалізації, 2-D графіка має свої особливості та переваги.[4]

По-перше, 2-D графіка надає гравцеві більш реалістичну та деталізовану візуальну інформацію. Використання спрайтів та текстур дозволяє створювати вигляд персонажів, об'єктів та фонів з більшою кількістю деталей та кольорів.

Це допомагає створити більш привабливе та реалістичне середовище гри, яке може захопити гравця та спонукати його до взаємодії.

По-друге, 2-D графіка дає можливість створювати візуальні ефекти та анімацію. Завдяки використанню спеціальних ефектів, таких як зміна кольору, переходи, рух та зміна розміру спрайтів, можна створювати рухливі та динамічні об'єкти в грі. [5] Це додає відчуття живості та активності до геймплею, покращуючи візуальний досвід гравця. (Рис 1.2)

Рисунок 1.2 – Приклад використання 2-D графіки у грі Terraria.

По-третє, 2-D графіка дозволяє створювати більш складні та деталізовані рівні та локації. Використання текстур та фонів дозволяє створювати різноманітні тематичні світи та візуальні стилі, що поглиблює іммерсію гравця. Крім того, 2-D графіка зазвичай використовується в платформерах та ретро-стильових іграх, де спрощена та плоска графіка може надати гравцям ностальгічні враження та сприяти швидкому геймплею.[6]

Однак, порівняно з текстовим способом візуалізації, 2-D графіка може обмежувати гравцеву уяву та інтерпретацію. Вона вже має визначений вигляд та стиль, що може зменшити гнучкість та креативність гравця у створенні власного образу світу гри. Крім того, залежно від якості графіки та художнього стилю, 2-D візуалізація може виглядати менш привабливою порівняно з високоякісною 3-D графікою.

В загальному, 2-D графіка є популярним способом візуалізації комп'ютерних ігор, який надає багато можливостей для створення реалістичного та привабливого ігрового досвіду. Вона надає більш деталізовану та рухливу візуальну інформацію, дозволяє створювати візуальні ефекти та анімацію, а також створювати різноманітні рівні та локації. Однак, в порівнянні з текстовим способом візуалізації, вона може обмежувати гравцеву уяву та креативність.

1.2.3 Спосіб візуалізації через 3-D графіку

Спосіб візуалізації через 3-D графіку відрізняється від попередніх методів і є одним з найбільш передових та реалістичних візуальних способів в комп'ютерних іграх. Використання 3-D графіки дозволяє створювати тривимірні об'єкти, середовища та персонажів, що надає гравцеві більшу глибину та іммерсію.

По-перше, 3-D графіка надає гравцям більшу реалістичність та деталізацію. Завдяки використанню тривимірних моделей та текстур, гравці можуть бачити об'єкти та персонажів з різними кутами огляду, використовувати світло-тіньові ефекти та отримувати більш реалістичні відчуття простору та присутності в грі.[7]

По-друге, 3-D графіка дозволяє створювати великі та деталізовані ігрові світи. Гравці можуть досліджувати великі локації, пересуватися вздовж осей X, Y та Z, і взаємодіяти з різними об'єктами та середовищем.[8] Це робить

геймплей більш відкритим та свобідним, а також дозволяє реалістично відтворювати природні елементи, архітектуру та інші деталі. (Рис. 1.3)

Рисунок 1.3 – Приклад використання 3-D графіки у грі Horizon Zero Dawn.

По-третє, 3-D графіка дозволяє створювати більш динамічні та спектакулярні ефекти. За допомогою розрахунку фізики, анімацій та частинок, можна створити рух, вибухи, руйнування та інші вражаючі візуальні ефекти, які підсилюють досвід гравця та роблять його більш захоплюючим.[9]

Однак, порівняно з попередніми способами візуалізації, 3-D графіка вимагає більших обчислювальних ресурсів та потужностей комп'ютера. Вона також може бути більш складною у розробці та вимагати великої команди художників, моделювальників та програмістів для створення високоякісних графічних об'єктів та ефектів.

Висновуючи, 3-D графіка є передовим та реалістичним способом візуалізації комп'ютерних ігор. Вона надає гравцям більшу реалістичність, деталізацію та глибину, дозволяє створювати великі та деталізовані ігрові світи,

а також створювати спектакулярні візуальні ефекти. Однак, вона вимагає більших обчислювальних ресурсів та команди фахівців для розробки.

1.2.4 Спосіб візуалізації через VR (virtual reality)

Спосіб візуалізації через віртуальну реальність (VR) відрізняється від попередніх методів тим, що надає гравцеві повний іммерсивний досвід, погружаючи його у віртуальне середовище. Порівняно з текстовим, 2-D та 3-D графікою, VR візуалізація забезпечує новий рівень взаємодії та сприйняття.

По-перше, VR візуалізація надає гравцю можливість бути в самому середині гри. Завдяки спеціальним гарнітурам VR, гравець може перенестися до віртуального світу та бачити його з усіх кутів, обертатися та рухатися, відчуваючи простір і присутність. Це дозволяє досягти високого рівня іммерсії та занурення, створюючи відчуття, що гравець дійсно перебуває у грі.

По-друге, VR візуалізація надає можливість взаємодії з віртуальним середовищем у набагато більш природний спосіб. Гравець може використовувати контролери руху або рухи тіла для взаємодії з об'єктами та персонажами в грі. Це робить ігровий процес більш реалістичним та інтуїтивно зрозумілим, оскільки гравець може фізично взаємодіяти з об'єктами у віртуальному світі. (Рис. 1.4)

По-третє, VR візуалізація надає можливість досліджувати простір у тривимірному форматі, подібно до 3-D графіки, але з вищим рівнем реалізму та присутності. Гравці можуть спостерігати деталізовані тривимірні об'єкти, взаємодіяти з ними та рухатися у тривимірному просторі, що надає ще більш реалістичне відчуття.

Однак, в порівнянні з попередніми методами, VR візуалізація вимагає додаткового обладнання (гарнітури VR, контролери тощо) та потужних обчислювальних ресурсів, що може бути обмежувальним для деяких гравців.

Рисунок 1.4 – Приклад використання VR у грі Phasmophobia.

Крім того, наявність фізичних обмежень (наприклад, обмежена площа для руху) може впливати на повноту іммерсії.

Висновуючи, VR візуалізація є найбільш іммерсивним та взаємодійним способом візуалізації комп'ютерних ігор. Вона надає гравцям можливість погрузнутися в віртуальний світ та взаємодіяти з ним у природний спосіб. Однак, вона вимагає спеціального обладнання та потужних ресурсів.

ВИСНОВОК ДО РОЗДІЛУ 1

Комп'ютерні ігри – це форма розваги, яка поєднує в собі різноманітні елементи, включаючи геймплей, сюжет, звуковий супровід та візуальну складову. Різні способи візуалізації грають ключову роль у створенні іммерсивного досвіду для гравців. У даному тексті ми порівняли текстову, 2-D графіку, 3-D графіку та віртуальну реальність (VR) як способи візуалізації комп'ютерних ігор.

Текстовий спосіб візуалізації відрізняється своєю абстрактною та уявністю формою. Хоча текстова складова може запропонувати глибокий наратив та вираження ідей, вона має свої обмеження у передачі реалістичності та деталізації візуального світу.

2-D графіка використовує двомірні об'єкти та площини для створення візуальних складових ігор. Вона надає можливість розробляти деталізовані та естетичні образи, але має обмежену просторову глибину та перспективу.

3-D графіка є передовим способом візуалізації, який надає реалістичність, глибину та деталізацію. Вона дозволяє створювати тривимірні об'єкти та середовища, що підвищує іммерсію та відчуття присутності в грі.

VR візуалізація є найбільш іммерсивним способом візуалізації. Вона дозволяє гравцеві погрузнутися у віртуальний світ та взаємодіяти з ним, створюючи реалістичний досвід та повний контроль над грою.

Загалом, вибір способу візуалізації залежить від конкретних потреб розробника та вподобань гравців. Кожен спосіб має свої унікальні переваги та внесок у створення захоплюючих та незабутніх ігрових досвідів. Розвиток технологій візуалізації продовжується, пропонуючи ще більш інноваційні та захоплюючі способи візуалізації комп'ютерних ігор у майбутньому.

РОЗДІЛ 2. ОГЛЯД АЛГОРИТМІВ ТА ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ

2.1 Аналіз середовищ розробки та їх ліцензій

Ігровий рушій – програмне забезпечення, призначене для розробки комп’ютерних ігор та інших інтерактивних програм, що обробляють графіку у режимі реального часу.

Для аналізу було взято на розгляд декілька ігрових рушіїв – Unreal Engine 5, Cry Engine 5, Source Engine, Unity 2021. [10]

2.1.1 Unreal Engine 5

Unreal Engine 5 (UE5) є потужним інструментом для розробки комп'ютерних ігор, який надає широкий спектр можливостей щодо роботи з графікою. UE5 відкриває безліч можливостей для розробників, надаючи їм потужний рушій інтерфейсу, редактор вмісту та інструментарій, що дозволяють створювати вражаючі візуальні ефекти та деталізовані графічні образи.[11] Ось основні особливості Unreal Engine 5 з точки зору роботи з графікою:

1. Високоякісна 3-D графіка: UE5 пропонує високоякісну рендерингову систему, яка дозволяє створювати деталізовані та реалістичні тривимірні об'єкти та середовища. Рушій Unreal Engine використовує сучасні алгоритми освітлення, тіней та текстури, що дозволяє досягти вражаючого рівня графічної якості.
2. Можливості фізики: UE5 має вбудовану систему фізики, яка дозволяє моделювати реалістичні фізичні взаємодії між об'єктами в грі. Це дозволяє створювати рухомі об'єкти, симулювати реальну поведінку тіл, реалістичні колізії та руйнування об'єктів.

3. Візуальні ефекти: UE5 надає широкий спектр візуальних ефектів, які можуть покращити графічний досвід гравців. Це включає реалістичну воду, вогонь, дим, пил, світлові ефекти, частинки та багато інших ефектів, що додають реалізм та атмосферність до гри.
4. Візуальні редактори та інструменти (Рис. 2.1): UE5 має потужний візуальний редактор, який дозволяє розробникам створювати та налаштовувати графічні елементи гри без необхідності програмування. Це включає редагування матеріалів, освітлення, камери, анімацію та інші важливі аспекти графіки.

Рисунок 2.1 – Редактор Unreal Engine 5

5. Масштабованість та оптимізація: UE5 пропонує інструменти для оптимізації графіки, що дозволяють розробникам забезпечити високу продуктивність гри навіть на менш потужних пристроях. UE5 має системи, такі як Level of Detail (LOD), які дозволяють автоматично знижувати рівень деталізації моделей та текстур для забезпечення плавної роботи гри.

6. Мультиплатформеність: Unreal Engine 5 підтримує розробку гри для різних платформ, включаючи ПК, консолі, мобільні пристрої та віртуальну реальність. Це дозволяє розробникам створювати гру один раз і випускати її на різних платформах без значних зусиль.

Загалом, Unreal Engine 5 є потужним інструментом для розробки графіки в комп'ютерних іграх. Він надає розробникам широкі можливості для створення деталізованих, реалістичних та іммерсивних графічних ефектів, які допомагають створити незабутній ігровий досвід для гравців.

Офіційний веб-сайт рушія Unreal Engine пропонує розгалужену документацію та навчальну інформацію для початківців розробників. Тут також є магазин, який надає різноманітні ресурси для розробки, включаючи моделі персонажів, об'єктів, скрипти, звуки, анімацію та додаткові плагіни для рушія.

З 2015 року Unreal Engine надає безкоштовну ліцензію, яку можуть використовувати будь-які розробники. Однак, якщо проект, створений на основі рушія, заробляє більше ніж 3 тисячі доларів за квартал, розробник повинен сплачувати 5% від прибутку компанії Epic Games.

Unreal Engine більш придатний для розробки глобальних 3D-ігор, ніж 2D. Втім, він все ж має широкий спектр платформ, які підтримують готові проекти та розробку нових ігор.

2.1.2 Cry Engine 5

Cry Engine 5 є потужним двигуном розробки комп'ютерних ігор, який славиться своєю вражаючою графікою та фотореалістичними візуальними ефектами. З точки зору роботи з графікою, Cry Engine 5 має кілька особливостей, які роблять його привабливим для розробників.[12] Давайте розглянемо основні особливості Cry Engine 5 з точки зору роботи з графікою.

1. Фотореалістична графіка: Однією з основних особливостей Cry Engine 5 є його здатність створювати фотореалістичну графіку. Двигун використовує передові технології рендерингу, такі як Global Illumination, Real-Time Reflections, Screen Space Ambient Occlusion та інші, щоб створити деталізовані та живі ігрові світи. Це дозволяє розробникам створювати іммерсивні ігрові досвіди, які наближаються до реальності.
2. Розширені візуальні ефекти: Cry Engine 5 надає широкі можливості для створення різноманітних візуальних ефектів. Він підтримує високоякісне освітлення, реалістичні тіні, частинки, руйнування та багато інших ефектів. Це дозволяє розробникам додавати спеціальні ефекти, які покращують візуальну привабливість ігри і створюють більш реалістичний світ.
3. Гнучкість у роботі з матеріалами (Рис. 2.2): Cry Engine 5 має потужну систему матеріалів, яка дозволяє розробникам створювати складні матеріали з різноманітними властивостями. Це включає текстури, бамп-мапи, спекулярні картки та інші параметри, що дають змогу створювати реалістичні матеріали для об'єктів у грі. Розробники можуть точно налаштувати вигляд об'єктів, що дозволяє досягти високої деталізації та реалістичності.
4. Розширена фізична модель: Cry Engine 5 включає розширену фізичну модель, яка дозволяє створювати реалістичні фізичні ефекти у грі. Розробники можуть моделювати рух об'єктів, колізії, руйнування та взаємодію з навколишнім середовищем. Це створює динамічні та реалістичні сцени, які підвищують іммерсію та взаємодію гравця з оточуючим світом.
5. Інтеграція з іншими інструментами: Cry Engine 5 підтримує інтеграцію з різними програмами та редакторами для спрощення роботи з графікою. Розробники можуть використовувати зовнішні програми для створення та редагування моделей, текстур, анімації та інших елементів графіки, а

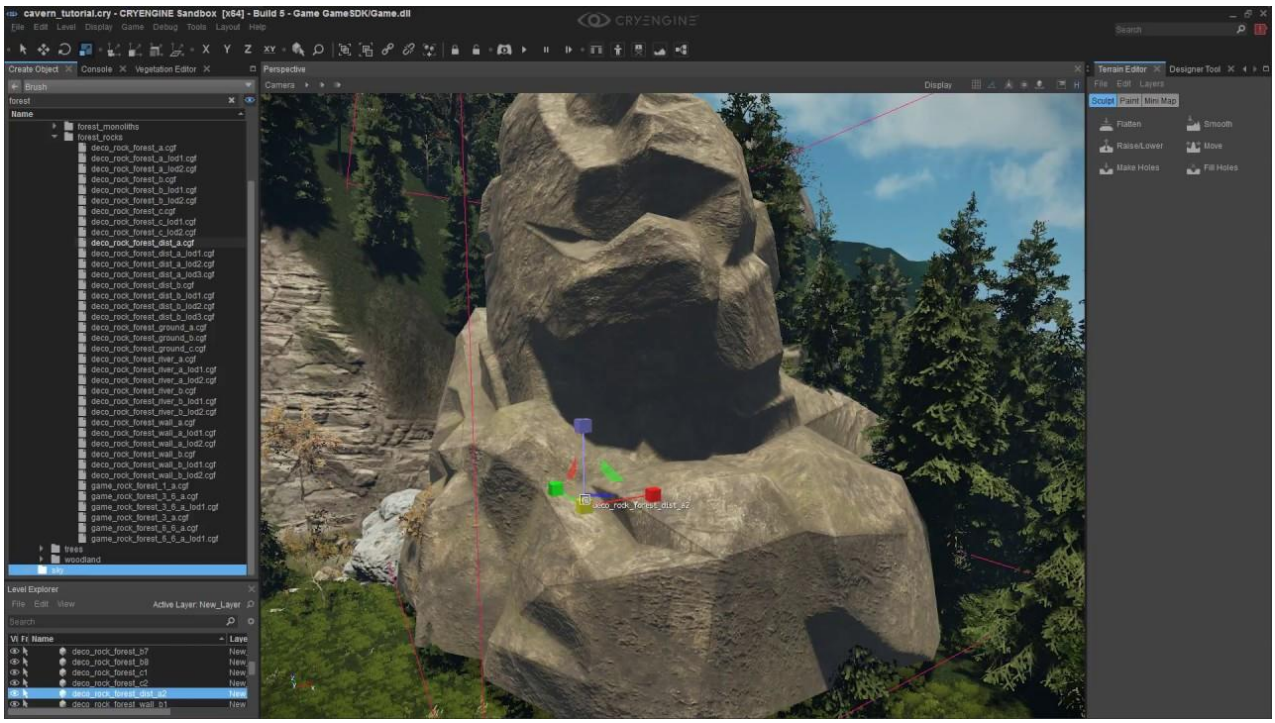


Рисунок 2.2 – Редактор Cry Engine 5

потім імпортувати їх в Cry Engine 5 для подальшої розробки ігрового світу.

Однак, в порівнянні з Unreal Engine 5 та Source Engine, Cry Engine може відрізнятися захоплюючою графікою та потужною реалістичністю, але він може мати меншу кількість ресурсів та інструментів, доступних для розробників.

Рушій Cry Engine, так само як його попередник, має офіційний веб-сайт, де розміщена навчальна інформація та магазин з різноманітними ресурсами для розробки. Цей рушій також надає безкоштовну ліцензію, яку можуть використовувати будь-які розробники. Проте, якщо продукт, створений на основі Cry Engine, приносить компанії або фізичній особі прибуток понад 5 тисяч доларів на рік, то 5% цього прибутку має бути сплачено компанії Crytek.

2.1.3 Source Engine

Source Engine є відомим двигуном розробки комп'ютерних ігор, який використовується для створення вражаючої графіки. З точки зору роботи з

графікою, Source Engine має декілька особливостей, які роблять його популярним серед розробників.[13] Давайте розглянемо основні особливості Source Engine з точки зору роботи з графікою:

1. Візуальна якість: Source Engine відомий своєю високою якістю графіки. Він підтримує різні технології, такі як шейдери, освітлення, тіні та покращену текстурування, що дозволяє розробникам створювати реалістичні та деталізовані ігрові світи. Гладкість зображення та багатий колірний діапазон допомагають створювати іммерсивний досвід для гравців.
2. Фізична модель: Source Engine має потужну фізичну модель, що дозволяє реалістично моделювати рух об'єктів, колізії та взаємодію з навколишнім середовищем. Це дозволяє створювати динамічні та інтерактивні сцени з реалістичною фізикою, яка відображає поведінку об'єктів у реальному світі.
3. Моделювання та редагування (Рис. 2.3): Source Engine надає зручні інструменти для моделювання та редагування графічних об'єктів. Розробники можуть створювати та редагувати 3D-моделі, текстури, освітлення та ефекти безпосередньо в середовищі двигуна. Інтеграція з зовнішніми програмами, такими як Blender або 3ds Max, також підтримується, що дозволяє розширити можливості створення графіки.
4. Зручний розробницький інтерфейс: Source Engine має інтуїтивно зрозумілий та зручний розробницький інтерфейс, який полегшує роботу з графікою. Розробники можуть швидко створювати та редагувати матеріали, освітлення, анімацію та ефекти, зосереджуючись на креативному процесі без зайвих технічних складнощів.
5. Підтримка модифікацій: Source Engine відомий своєю відкритістю для модифікацій. Розробники можуть створювати та впроваджувати власні модифікації, змінюючи графіку, механіку гри та інші аспекти. Це надає широкі можливості для творчості та індивідуалізації гри.

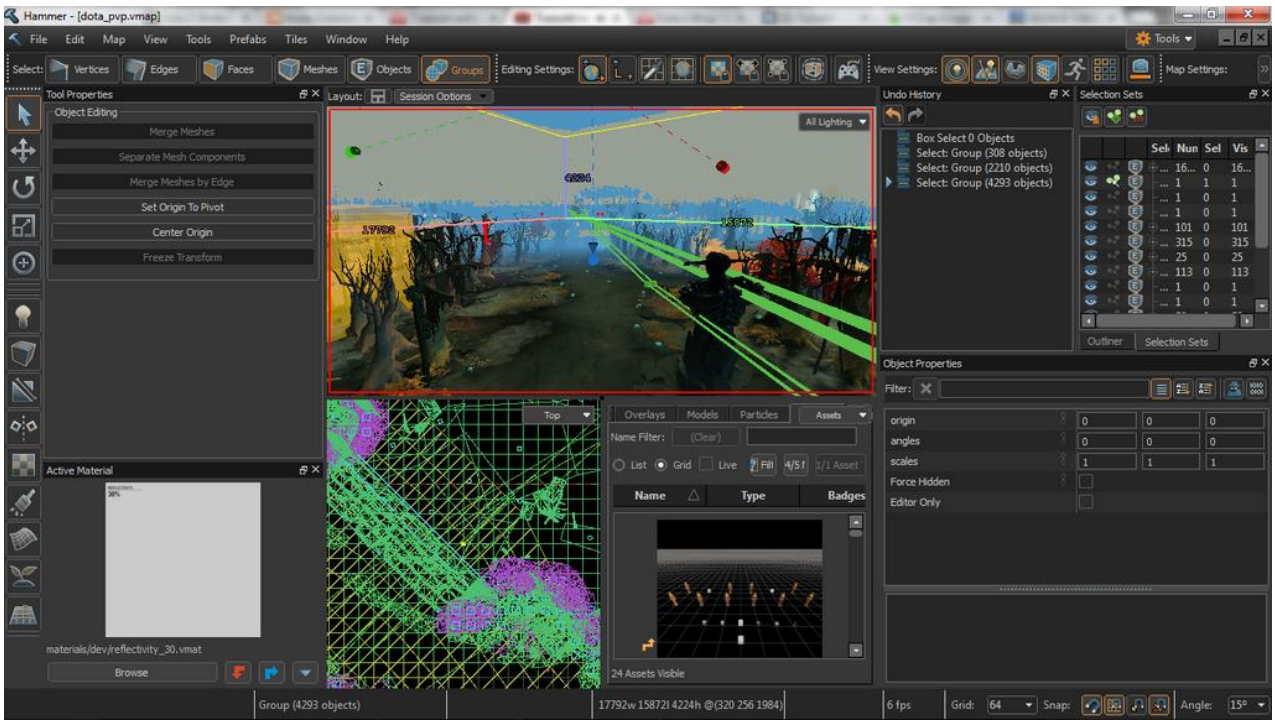


Рисунок 2.3 – Редактор Source Engine

Порівнюючи Source Engine з Unreal Engine 5, Cry Engine 5 та Unity, варто зазначити, що кожен з цих двигунів має свої особливості та переваги у роботі з графікою. Однак, Source Engine відомий своєю високою якістю графіки, потужною фізичною моделлю та зручним інтерфейсом розробки. Він також має довгу історію в розробці ігор і популярний серед багатьох відомих ігор.

Ліцензія Source Engine має різні форми та умови використання. Перш за все, важливо зазначити, що Source Engine є комерційним продуктом, тому він не надає безкоштовну ліцензію для загального використання, як це може бути у випадку інших рушіїв, таких як Unreal Engine чи Unity.

Оскільки доступ до ліцензії Source Engine є обмеженим, деталі та ціни ліцензування можуть бути встановлені лише через прямий контакт з Valve Corporation або їхнім представником. Компанія пропонує індивідуальні угоди та умови ліцензування, які можуть варіюватися залежно від потреб розробників і характеристик проекту.

2.1.4 Unity 2021

Unity є одним з найпопулярніших двигунів розробки ігор[14], який має багато особливостей з точки зору роботи з графікою. Основні особливості Unity включають:

1. Широкий спектр платформ: Unity підтримує багато платформ, включаючи комп'ютери, консолі, мобільні пристрої та віртуальну реальність. Це дозволяє розробникам створювати гри для різних пристроїв і ринків.
2. Візуальний редактор (Рис. 2.4): Unity має потужний візуальний редактор, який дозволяє розробникам створювати та редагувати графіку безпосередньо у середовищі розробки. Розробники можуть швидко розміщати об'єкти, налаштовувати матеріали, освітлення, анімацію та інші властивості, зокрема, шляхом використання drag-and-drop.

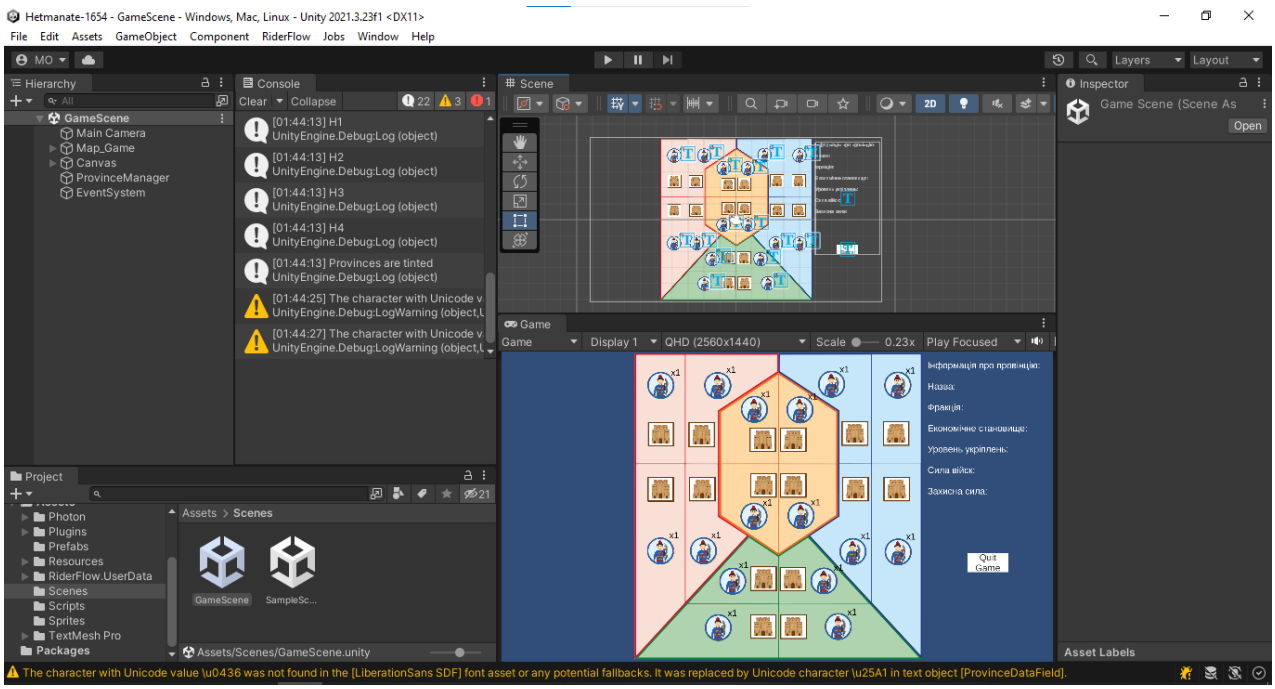


Рисунок 2.4 – Редактор Unity 2021

3. Розширена система шейдерів: Unity надає потужну систему шейдерів, яка дозволяє розробникам створювати різноманітні візуальні ефекти та матеріали. Розробники можуть програмувати власні шейдери або використовувати готові рішення з Asset Store, щоб створювати привабливі та реалістичні графічні ефекти.
4. Система частинок: Unity має вбудовану систему частинок, яка дозволяє створювати реалістичні ефекти, такі як дим, вогонь, вибухи та багато інших. Розробники можуть налаштовувати поведінку та вигляд частинок, що дозволяє створювати вражаючі візуальні ефекти у грі.
5. Підтримка VR і AR: Unity має вбудовану підтримку віртуальної реальності (VR) та доповненої реальності (AR). Це означає, що розробники можуть створювати ігри та додатки для платформ, таких як Oculus Rift, HTC Vive, Microsoft HoloLens та інших. Unity надає потужні інструменти та ресурси для розробки інтерактивних віртуальних світів.
6. Asset Store: Unity має велику онлайн-бібліотеку Asset Store, де розробники можуть знайти готові моделі, текстури, ефекти, анімацію та інші ресурси для використання у своїх проектах. Це значно спрощує процес розробки графіки, оскільки розробники можуть використовувати готові елементи або знайти інспірацію для власних створень.

Порівнюючи Unity з Unreal Engine 5, Cry Engine 5 та Source Engine з точки зору роботи з графікою, Unity є дуже доступним та розширюваним інструментом. Він надає широкий спектр можливостей у роботі з графічними ефектами, матеріалами, анімацією та іншими візуальними аспектами. Unity також відрізняється великою спільнотою розробників та активною підтримкою з боку компанії, що дозволяє розробникам швидко знаходити відповіді на свої запитання та розв'язувати проблеми.[15] Однак, вибір двигуна залежить від конкретних потреб проекту, рівня експертизи розробника та особистих уподобань.

Рушій Unity 2021 пропонує три версії для придбання: Personal (безкоштовна), Plus (35$/місяць) та Pro (125$/місяць). Кожна версія має свої особливості та межі максимально допустимого річного прибутку, які становлять 100 тисяч доларів для версії Personal, 200 тисяч доларів для версії Plus і не має обмежень для версії Pro. Це дозволяє розробникам обрати підходящу версію, враховуючи їхні потреби та бюджет.

Основними критеріями для вибору середовища розробки для нашого проекту є вільна ліцензія, наявність навчальної документації, зручний та зрозумілий інтерфейс, низькі системні вимоги та наявність інструментів для розробки 2D проектів. З урахуванням цих факторів, ми вирішили обрати ігровий рушій Unity.

Реалізовувати програмний код в Unity я буду через середовище розробки JetBrains Rider (Рис. 2.5), яке має можливість інтеграції додаткового функціоналу для Unity.

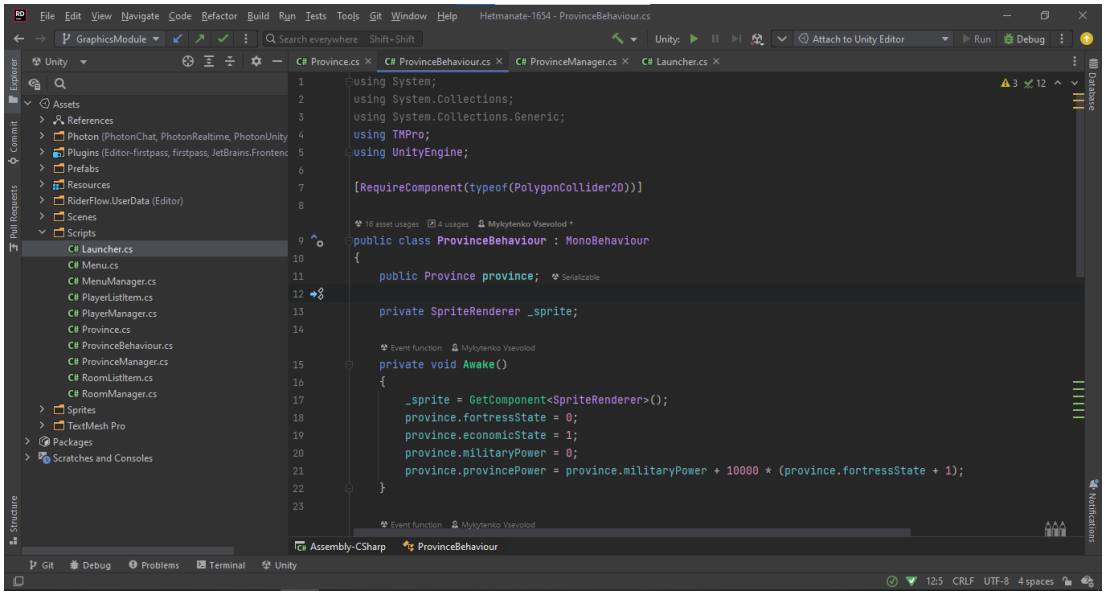


Рисунок 2.5 – Редактор JetBrains Rider

JetBrains Rider – потужне IDE для розробки ігор на платформі Unity. Забезпечує повну підтримку Unity API, зручне редагування шейдерів, відлагодження коду в реальному часі та підтримку юніт-тестування. Rider

пропонує розширену функціональність для розробки Unity-проектів, таку як автодоповнення, візуальне редагування інтерфейсу, керування сценами, ресурсами та імпорт/експорт проектів. Його потужні інструменти допомагають розробникам створювати високоякісні ігрові проекти, забезпечуючи зручну і продуктивну робочу область.

2.2 Загальний алгоритм реалізації проекту

Алгоритм розробки комп'ютерної гри мало чим відрізняється від алгоритму для будь-якого іншого програмного продукту. Розробка комп'ютерної гри включає такі 3 етапи:

- Проектування;
- Розробка;
- Видання та підтримка.

2.2.1 Проектування

На етапі проектування визначаються мета гри та засоби її розробки.[16]

При визначенні мети гри виділяються ідея, жанр і сеттинг. Ідея гри тісно пов'язана з жанром і мотивує гравців грати. Наприклад, основна ідея шутерів полягає в можливості брати участь у реальних або вигаданих бойових діях, тоді як у RPG основна ідея полягає в тому, щоб дозволити гравцеві відігравати роль по-своєму. Таким чином, після визначення основних ідей гри жанр можна практично вибрати відразу. Ідея спонукає вибрати відповідний жанр, який найкраще втілить цю ідею і приверне увагу гравців.

Сеттинг гри – це світ, в якому відбувається дія гри, його атмосфера та характеристики. Вибір сеттингу важливий, оскільки він впливає на настрій гравців та їх іммерсію у гру. Наприклад, сеттинг може бути футуристичним

науково-фантастичним світом, середньовічним фентезійним умовами або сучасним міським оточенням.

При виборі сеттингу важливо врахувати взаємодію між ідеєю гри, жанром і атмосферою сеттингу. Гармонійна сполучення всіх цих елементів допоможе створити захоплюючий та цікавий світ для гравців.

Таким чином, визначившись з жанром і ідеєю гри, вибір відповідного сеттингу допоможе створити унікальний досвід для гравців і підкреслить концепцію гри.

Вибір засобів розробки має велике значення для швидкості та ефективності процесу розробки комп'ютерних ігор.[17] Код, написаний розробником, залежить від платформи, на якій гра буде запускатися. Наприклад, для браузерних ігор можуть використовуватись мови програмування, такі як Java або Flash. У той же час, для створення ігор для персональних комп'ютерів оптимальним вибором може бути мова програмування C#.

Ігровий рушій, як компонентом розробки, відповідає за опис фізики об'єктів, правила рендерингу графіки та інші аспекти гри. Існує кілька різних ігрових рушіїв, наприклад рушії з бешкоштовною ліцензією – Unity, Unreal Engine. В основному, вони надають потужні інструменти для розробки графіки, фізики та інших аспектів гри.

При виборі ігрового рушія розробники спочатку оцінюють його доступність та мови програмування, які ним підтримуються. Наприклад, ігровий рушій Unity дозволяє розробляти гри з використанням мови програмування C# і підтримує інтеграції з середовищами розробки для цієї мови.

Отже, вибір ігрового рушія має велике значення для успішної розробки гри, залежно від його функціональності, підтримки мов програмування та доступності для розробників.

2.2.2 Розробка

Після визначення мети та ідеї гри, наступним кроком алгоритму є розробка, де розробники створюють елементи, необхідні для реалізації гри.[18]

Розробка комп'ютерної гри є найбільшим та найдовшим етапом у процесі реалізації проекту. Вона включає в себе багато різноманітних кроків, які є необхідними для створення працездатного та захоплюючого продукту, що задовольнятиме потреби гравців.

Початок цього етапу полягає у визначенні ігрової механіки та сюжету, які відіграють ключову роль у створенні неповторного ігрового досвіду. Ігрова механіка охоплює всі аспекти взаємодії гравця з об'єктами та правила, які регулюють цю взаємодію, тоді як сюжет впливає на зацікавленість та емоційну залученість гравця до гри. Сюжет може бути представлений у формі літературного або режисерського сценарію, які детально описують основні події, характери персонажів та розвиток сюжетної лінії гри. Комбінація ігрової механіки та захопливого сюжету створює унікальну ігральну ситуацію, яка приваблює та утримує увагу гравців, спонукаючи їх бажати продовжувати грати та досліджувати ігральний світ.

На цьому етапі розробники також відводять час на раннє концептуальне розроблення графічного стилю та дизайну гри. Шляхом створення концепт-артів вони можуть виявити та визначити візуальні концепції гри та персонажів, встановлюючи загальний вигляд гри та впливаючи на естетичне враження, яке вона справить на гравців. Концепт-арт дозволяє розробникам експериментувати з різними стилістичними підходами, колоритом, освітленням та іншими візуальними елементами, що допомагають створити неповторний та привабливий світ гри.

На основі сюжету та концепт-артів розпочинається процес створення персонажів та об'єктів гри, одночасно з цим розпочинається розробка ігрових рівнів. Першим етапом при створенні рівнів є створення їх спрощеного плану,

де схематично зображений сам рівень та предмети, з якими буде взаємодіяти гравець. Після цього настає момент, коли створюється перша версія рівня, яка зазвичай є простою локацією з мінімумом необхідних предметів для проходження. Ця версія рівня використовується для тестування його прохідності. Поступово рівень заповнюється іншими об'єктами та елементами після успішного проходження тестування.

Після створення початкових рівнів починається складання першого прототипу гри, відомого як альфа-версія.[19] Ця версія є важливою для проведення альфа-тестування, яке спрямоване на перевірку основної механіки гри та відповідність її заявленим вимогам. Зазвичай, в альфа-версіях гри об'єкти не мають текстур або вони представлені у вигляді абстрактних об'єктів, що дозволяє зосередитися на функціональності та геймплеї. Таке підходження допомагає виявити та виправити можливі проблеми ще на ранній стадії розробки гри.

Після успішного проходження тестування альфа-версії гри настає етап опрацювання механіки та об'єктів гри. На цьому етапі розробники працюють над вдосконаленням рівнів та механіки гри, а також починають впроваджувати перші сюжетні події, такі як відеоролики, сюжетні діалоги та кат-сцени. Паралельно з цим здійснюється виправлення помилок та несправностей, виявлених під час тестування альфа-версії гри, що забезпечує подальшу покращену якість та функціональність гри.

Після виконання попередніх кроків настає етап розробки другого прототипу гри, відомий як бета-версія. Бета-версія призначена для інтенсивного тестування гри з метою виявлення будь-яких несправностей та вдосконалення її перед остаточним випуском. У бета-версії гри, яка вже майже готова до випуску, можуть відсутні незначні елементи, що не впливають на основний геймплей. Під час тестування бета-версії гри проводиться комплексна перевірка всіх її аспектів. Зокрема, у випадку мультиплеєрних ігор, звичайних гравців

запрошують для активної участі в тестуванні, що сприяє прискоренню процесу тестування та зменшенню навантаження на розробників.

2.2.2 Видання і підтримка

Якщо гра успішно пройшла бета-тестування, наступним кроком є проведення остаточного доопрацювання та виправлення критичних помилок. Після цього відбувається збірка фінальної версії гри, підготовка до релізу та сам реліз гри. Також на цьому етапі визначається чи буде виходити гра на різні платформи.[20] Після релізу починається фаза підтримки гри, яка включає випуск патчів – файлів, що виправляють помилки у готовому продукті. Крім того, для продовження життєвого циклу гри розробники можуть випускати додатковий контент у вигляді DLC (завантажуваних змістових пакетів), які додають нові предмети та можливості для гравця.

Аналізуючи процес розробки комп'ютерних ігор, стає очевидним, що його етапи мало відрізняються від етапів розробки інших програмних продуктів.

2.3 Аналіз потенційного попиту проекту

Для створення аналізу потенційного попиту проекту потрібно розглянути різні складові актуальності і конкурентноспроможності на ринку. Типічний аналіз має наступні складові:

- Аналіз потенційної аудиторії споживачів;
- Актуальність проекту;
- Вимоги до функціоналу.

2.3.1 Аналіз потенційної аудиторії споживачів

Характеристика цільової аудиторії є надзвичайно важливим етапом у розробці будь-якого проекту, оскільки вона допомагає розробнику краще зрозуміти свою цільову аудиторію і звернути більшу увагу до деталей продукту, що розробляється. Тому в рамках даного проекту була детально описана цільова аудиторія.

Для цього проекту була обрана широка цільова аудиторія віком від 15 до 40 років. Вибір такої широкої категорії аудиторії обумовлений кількома факторами. По-перше, це середній вік активних гравців комп'ютерних ігор, який в країнах СНД стабільно коливається у діапазоні від 30 до 40 років. Саме з цих причин була встановлена верхня межа віку цільової аудиторії гри. Цільовою аудиторією гри були обрані люди, які цікавляться даним жанром. Прості правила гри створюють малий рівень входження, але виличезна кількість розвитків ігрової сесії і механіки гри зав'язані на комунікації між людьми створюють реалістичну симуляцію стратегії, що в свою чергу може зацікавити більш вибагливих гравців. Загалом, цей проект може зацікавити різноманітну аудиторію з різними інтересами та вподобаннями.

2.3.2 Актуальність проекту

У сучасних реаліях ігрової індустрії можна спостерігати швидкий та стійкий розвиток. З кожним наступним роком збільшується не тільки загальна кількість активних гравців, але й розширюється аудиторія ігрових проектів. Разом з цим з'являються нові студії, що створюють унікальні ігри. Цей ринок приносить величезний прибуток.

У сфері розробки ігор, зокрема в жанрі стратегій, спостерігається високий рівень вимог до гравців. Відтак, для того, щоб успішно осягнути геймплей та впоратися з викликами, необхідне глибоке розуміння механік гри. Стратегічні

ігри вимагають від гравців уважності, аналітичних навичок та здатності приймати важливі рішення на основі доступної інформації.

Водночас, варто зазначити, що стратегічні ігри не є легкими для початківців. Вони можуть мати складну систему правил та інтерфейс, що вимагають від гравця певного часу та зусиль, щоб осягнути їх суть. Тим не менш, саме ця складність привертає гравців, які цінують виклики та глибину геймплею, бажають побудувати власну стратегію та вплинути на віртуальний світ.

В порівнянні з великими проектами, що вимагають великої кількості годин інвестованого часу, стратегічні ігри можуть бути привабливими для гравців, які хочуть насолодитися грою, не витрачаючи безліч часу на її проходження. Вони дозволяють гравцям мислити стратегічно, приймати рішення і спостерігати за результатами своїх дій у відносно короткий період часу.

Загалом, ігри в жанрі стратегій залишаються привабливими для гравців, які прагнуть до викликів, глибини та можливості впливати на розвиток віртуального світу з використанням своїх стратегічних навичок та розумового потенціалу.

2.3.3 Вимоги до функціоналу

Проект представляє собою комп'ютерну гру в жанрі двовимірної стратегії, яка має на меті створити засіб для розваги та задоволення від проведення часу.

Можна вивести деякі вимоги, яким повинен відповідати проект гри стратегії:

- Мапа та територія: Реалістична та деталізована мапа, на якій гравець може взаємодіяти з різними територіями, розміщувати свої поселення, виробляти ресурси та будувати споруди.

- Ресурси та економіка: Система ресурсів, яка дозволяє гравцеві збирати, виробляти та управляти різними видами ресурсів. Економічна система, яка враховує витрати та доходи, можливість торгівлі та розвитку інфраструктури.
- Управління персоналом: Можливість наймати та керувати персоналом, включаючи робітників, військових одиниць, науковців тощо. Кожен персонаж може мати свої унікальні вміння та характеристики.
- Дипломатія та війна: Система взаємодії з іншими гравцями або комп'ютерними противниками, включаючи можливість укладати альянси, вести переговори, торгувати та вести військові операції.

Дані вимоги до функціональної частини є одними з говних при реалізації гри стратегії.

ВИСНОВОК ДО РОЗДІЛУ 2

Усі розглянуті рушії – Unreal Engine 5, Cry Engine 5, Source Engine та Unity – є популярними та потужними інструментами для розробки комп'ютерних ігор. Кожен з них має свої особливості та переваги, які залежать від потреб розробника та характеристик проекту.

Unreal Engine 5 відрізняється високою якістю графіки та візуальних ефектів, зосередженими на 3D-розробці, а також має широкий вибір платформ та підтримку VR-технологій.

Cry Engine 5 також вражає своїм візуальним рівнем та потужностями для створення реалістичної 3D-графіки, зокрема у сфері освітлення та фізичної симуляції.

Source Engine відзначається швидкодією та низькими системними вимогами, що робить його ефективним для розробки 2D-проектів, але також підтримує 3D-графіку.

Unity є універсальним інструментом. Він приваблює своєю безкоштовною ліцензією, доступною навчальною документацією, зручним інтерфейсом та широким спектром інструментів.

Так як для реалізації цього проекту головними критеріями вибору середи розробки є вільна ліцензія, навчальна документація, зручний та зрозумілий інтерфейс, низькі системні вимоги та інструменти для розробки 2D проектів, нашим вибором стане ігровий рушій Unity.

Після ретельного дослідження середовища розробки, був визначений основний алгоритм розробки комп'ютерної гри. В рамках цього процесу були розглянуті ключові етапи створення будь-якого проекту, від написання сценарію до підтримки гри після випуску. Далі був розглянутий аналіз цільової аудиторії нашого проекту, а також визначенню актуальності та мети проекту. Завершенням другого розділу є визначення основного функціоналу проекту, яким він повинен володіти після завершення розробки.

РОЗДІЛ 3. ДЕТАЛІ РОЗРОБКИ СИСТЕМИ

3.1 Опис правил гри «Гетьманщина 1654»

Гетьманщина 1654 – Гра присвячена подіям і рішення, які приймали ключові країни Східної Європи в середині 17 століття, що на багато років визначили перебіг історії. Мета гри: За декілька наступних років (у грі відображені ходами) збудувати найпотужнішу і найбільшу державу у Східній Європі. Змагаючись з іншими гравцями вам потрібно дипломатичним і військовим шляхом досягти першості на зламі історії. У грі присутня ігрова мапа, яка поділена чотирма державами за кольорами: Московія – фіолетовий, Жовтий – Гетьманщина, Рожевий – Польща, Зеленуватий – Татарське ханство. Кожна держава має у своєму складі по 4 території. (фото поля). У грі кожен рік – це 1 хід, який в свою чергу поділяється на 3 фази. Перша – фаза Перемовин. Вона складається з трьох раундів перемовин, де гравці домовляються між собою про підписання договору. (фото про перемовини). Друга – фаза Воєнних дій. На основі підписаних договорів та власних стратегічних ходів держав здійснюються військові дії на ігровому полі. Та остання, третя фаза – Економічна. В ній кожен гравець збирає прибуток з утримуваних територій, купує війська та фортеці, розміщуючи їх на підвладних територіях.

3.1.1 Фаза Перемовин

Гравці на цій фазі ведуть між собою перемовини та заключають договір. Договори бувають двох типів: Таємний та Звичайний. Таємний договір – має найбільшу силу і перебиває дію звичайного договору і/або стратегічних ходів. Це дуже важливий договір і у вас всього 3 штуки на руках. Якщо ви вдало підпишете таємний договір, ви можете обманути гравця, з яким ви підписали звичайний договір. Звичайний договір – має середню силу, є найбільш

використовуваним типом договорів. Стратегічні ходи – мають найнижчу силу, підпорядковані і не можуть суперечити звичайним і таємним договорам.

3.1.2 Фаза Воєнних дій

Гравці на цій фазі вже бачать таємні договори, звичайні договори та стратегічні ходи. Якщо звичайні договори вступають у конфлікт з таємними – перші анулюються, а другі продовжують діяти. Після зачитування договорів відбувається переміщення всіх зазначених військ. Далі на кожній території, де відбуваються бойові дії ведеться підрахунок втрат і визначається переможець. Загиблі війська прибираються з ігрової мапи і на територію ставиться маркер випаленої землі.

3.1.3 Фаза Економіки

Гравці збирають зі своїх територій дохід. Далі кожен гравець купує війська і/або фортеці. У кінці фази є 2 хвилини щоб кожен гравець розставив війська по своїх територіях. На цьому хід завершується.

3.1.4 Закінчення гри

Гра може тривати від 5 до 7 ходів(років). Після 5 ходу кидається кубик(random від 1 до 6) : якщо випадає 4,5 або 6, тоді 6-ий хід відбувається. Після 6 ходу знову кидається кубик: якщо випадає 5 або 6, 7-ий хід відбувається. Більше 7 ходів гра тривати не може.

Критерії перемоги за порядком значимості:

1. Збережені власні землі
2. Кількість ворожих земель під вашим контролем
3. Сукупний дохід всіх підвладних територій

Якщо двома або більше державами виконується перший пункт, тоді переходять до другого, якщо знову дві або більше країн мають однакові результати, тоді переходять до 3 пункту.

Гра може закінчитися спільною перемогою 2 сторін, але якщо жодна держава не зберегла власні території – переможець визначається за кількістю територій і сукупним доходом.

3.2 Розробка модуля, робота зі спрайтами

Основною складовою частиною будь-якої комп’ютерної гри є її візуальна складова. Розробка графічного модулю для комп’ютерної гри основується на розміщені і взаемодії спрайтів на ігровій сцені, тому далі розберемо поняття спрайту:

Спрайт (англ. Sprite) - у двовимірних іграх є графічним зображенням якогось об'єкту. Він може містити кілька зображень, з яких можна створити анімацію для об'єкту. Спрайти використовуються для представлення персонажів, предметів, тлів та інших елементів гри.

Кожен спрайт має свої координати, що визначають його положення на екрані. Вони використовуються для розміщення спрайту на певній позиції на ігровій сцені. Крім того, спрайти можуть мати інші властивості, такі як розмір, швидкість руху, стан (наприклад, спрайт може бути активним або неактивним), а також можуть взаємодіяти з іншими спрайтами або об'єктами у грі.

Для створення анімації спрайта використовуються різні кадри зображень, які змінюються впродовж часу. Наприклад, для анімації ходьби персонажа можуть використовуватись послідовні зображення його руху.

Для розробки графічного модуля гри «Гетьманьщина 1654», автором оригинальних правил настільної гри було надано картинки необхідних ігрвих елементів (Рис. 3.1), які і були перенесені в гру у вигляді спрайтів.

Рисунок 3.1 – Оригінальні зображення для настільної версії гри

3.2.1 Спрайтами карти і провінцій

Робота над спрайтами карти і провінцій почалась з редагування оригинального файлу, для виокремлення з основної карти картинку кожної провінції окремо (Рис. 3.2).

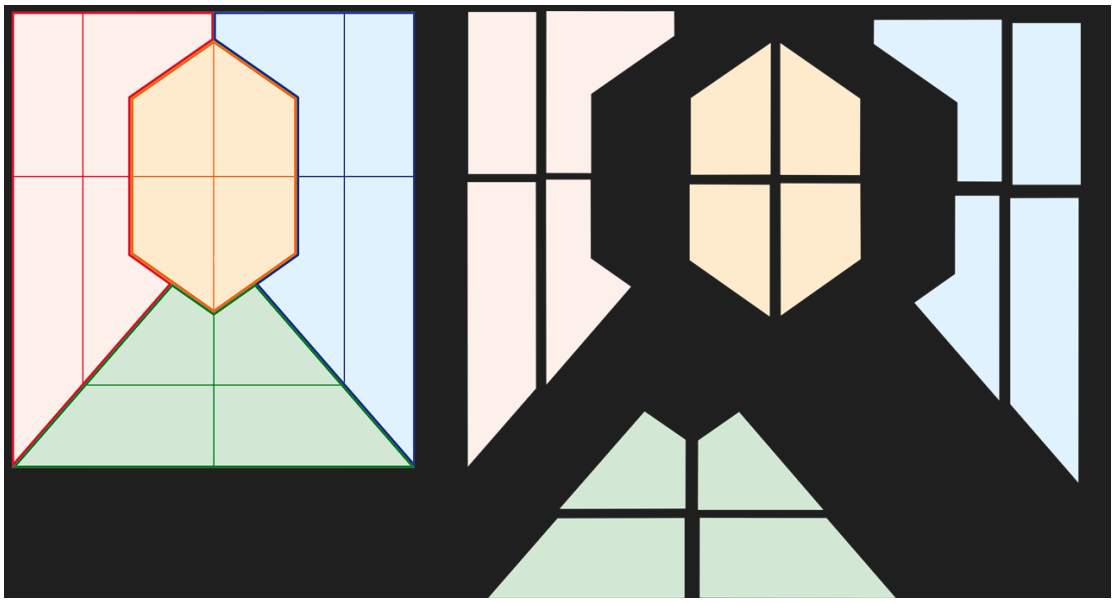


Рисунок 3.2 – Файл Карта_final.png після пертворень

Для перетворення зображень у спрайти Unity надає інструмент обробки спрайтиів Sprite Editor.

Sprite Editor - це вбудована функція в Unity, що надає можливість редагувати та маніпулювати спрайтами, які використовуються у грі або

додатку. Він надає зручний інтерфейс для роботи з графічними ресурсами та дозволяє виконувати різноманітні операції зі спрайтами, такі як розбиття на окремі частини, зміна розмірів, обрізка, вирівнювання та інші.

У випадку розбиття зображення на колекцію спрайтів, потрібно відкрити файл у Sprite Editor. За допомогою інструментів, які надає редактор, виділити окремі частини зображення, які будуть пізніше перетворені на окремі спрайти. Це дозволяє розбити зображення на більшість роздільних елементів або графічних об'єктів, які можуть бути незалежно використані в грі.

Після вирівнювання, обрізка зайвого простору, редагування розмірів та інших візуальних змін, отримаємо колекцію окремих спрайтів, які тепер можуть бути використані окремо або комбіновано для створення різних графічних ефектів та анімацій у грі. (Рис. 3.3)

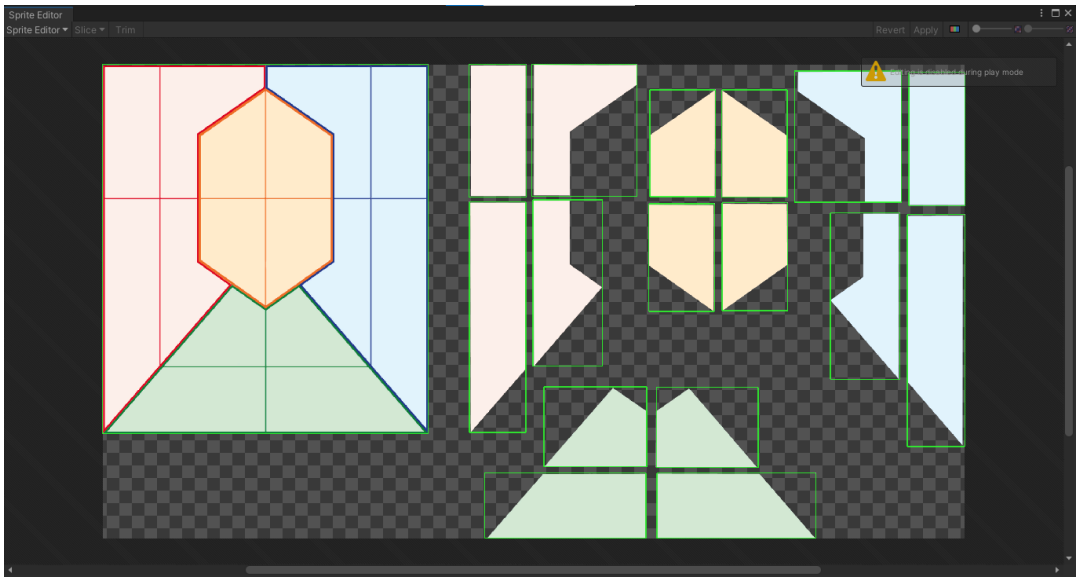


Рисунок 3.3 – Робиття картинки на колекцію спрайтів в Sprite Editor

Останім кроком роботи з завантаженням срайтів карти та провінцій стало їх розміщення на ігровій сцені і вирівняня по кордорах оригінальної карти (Рис. 3.4).

Цей процес включав в себе ретельне розташування кожного срайта на відповідному місці на ігровій сцені. Використовуючи інструменти, доступні в

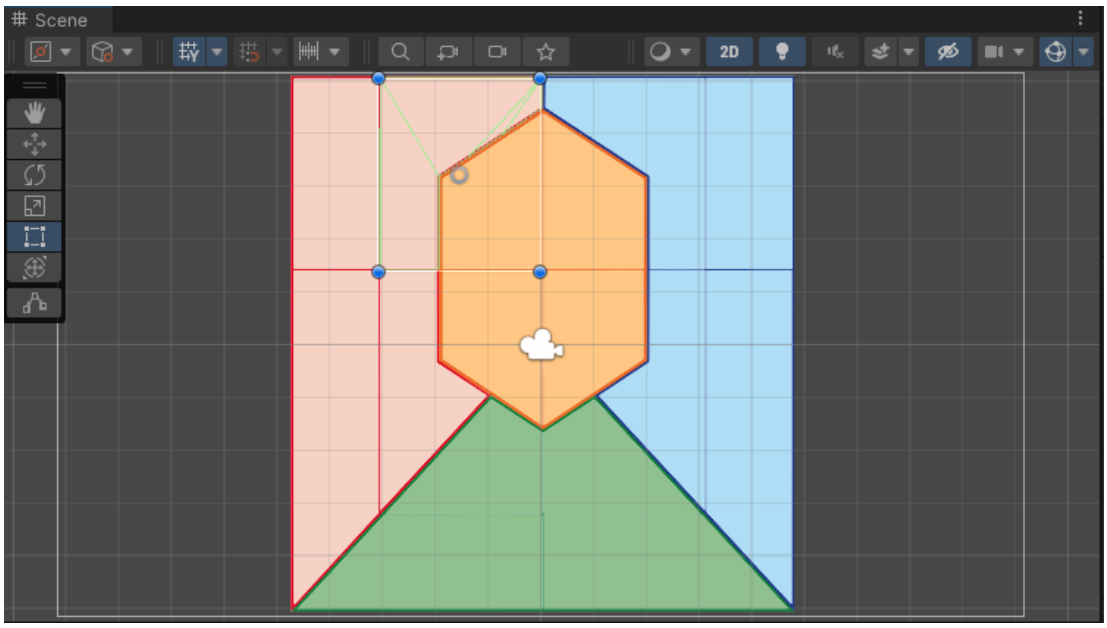


Рисунок 3.4 – Розміщення спрайтів карти і провінцій на ігровій сцені

редакторі гри, було перетягнуто та розміщено кожен срайт згідно з його відповідним положенням на оригінальній карті.

Цей процес включав в себе ретельне розташування кожного срайта на відповідному місці на ігровій сцені. Використовуючи інструменти, доступні в редакторі гри, було перетягнуто та розміщено кожен срайт згідно з його відповідним положенням на оригінальній карті.

Окрім розташування, важливим кроком було вирівнювання срайтів по кордорах оригінальної карти. Це означало, що кожен срайт мав бути точно розміщений відповідно до шляхів і структури картографічного дизайну.

3.2.2 Спрайтами ігрових фішок

Для спрайтів ігрових фішок було достатньо використати оригинальні картинки надані розробником настільної версії гри. Кожна картинка була окремо завантажена і перетворена Sprite Editor в окремий спрайт. (Рис. 3.5)

Після завантаження спрайтів для фішок, наступним кроком було їх розміщення на відповідних місцях по провінціях. (Рис. 3.6) Так само, як і з

Рисунок 3.5 – Приклад обробки спрайту в Sprite Editor

картою і провінціями, цей процес був важливим для створення гармонійної ігрової сцени.

Після завантаження спрайтів для фішок, наступним кроком було їх розміщення на відповідних місцях по провінціях. (Рис. 3.6) Так само, як і з картою і провінціями, цей процес був важливим для створення гармонійної ігрової сцени.

Кожна провінція мала свій власний спрайт, на якому потрібно було розмістити фішки військ та фортець різних фракцій. Оскільки на кожній провінції може бути лише одна фішка військ і одна фішка фортеці одночасно, спрайти були накладені один на одного. Це означало, що було визначено правильний порядок накладання спрайтів так, щоб відображалась тільки актуальна фішка військ фракції і фішка фортеці. (Рис. 3.6)

Важливим елементом відображення зв'язаним з спрайтами фішок військ є лічильник кількості фішок на одній провінції. Одна фішка зображує 5000 військ, а на провінції може знаходитись значно більша кількість солдатів, тому було вирішено додати лічильник біля кожної фішки військ. Це було реалізовано за допомогою текстового елемента відображення TextMeshPro.

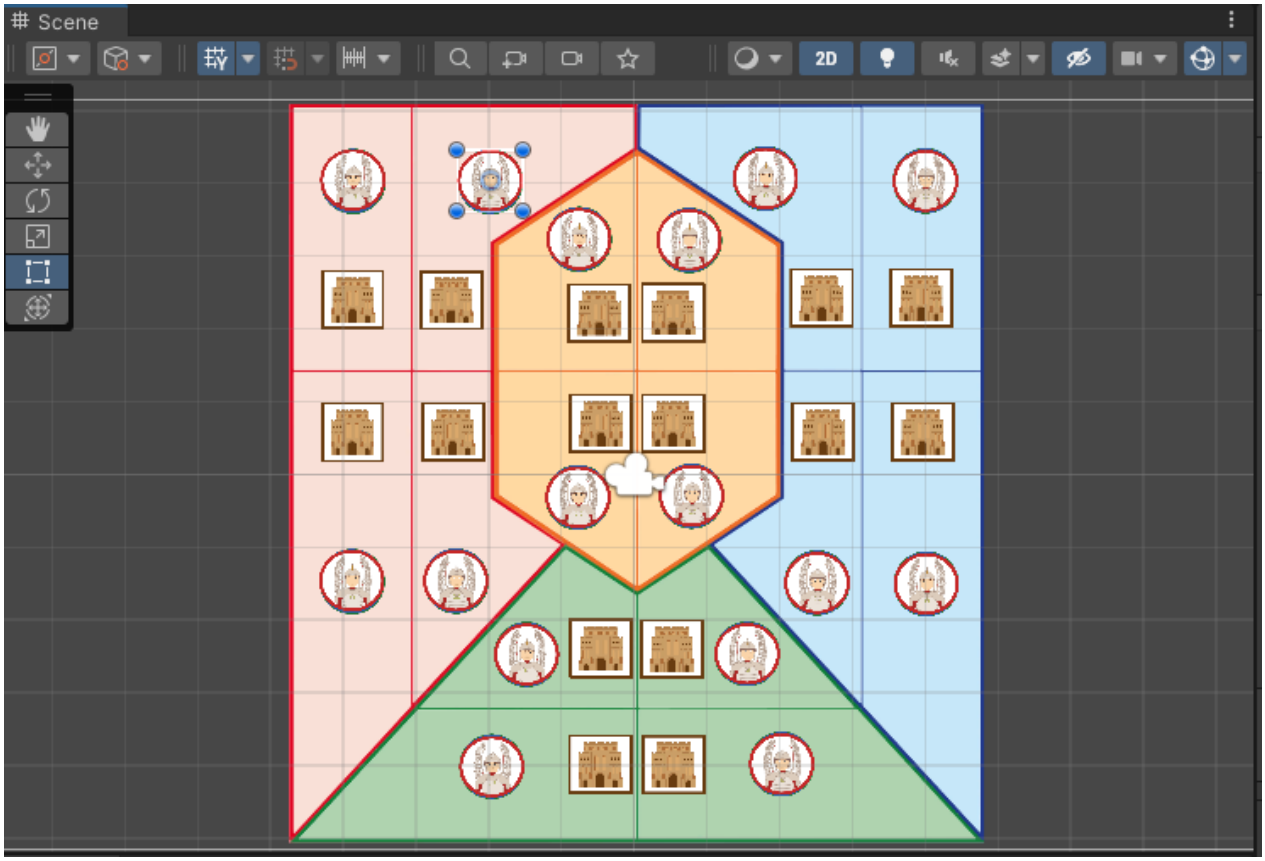


Рисунок 3.6 – Розміщеня спрайтів фішок на ігровому полі

Для відображення кількості фішок військ на одній провінції було вирішено використати текстовий елемент відображення TextMeshPro. Цей елемент реалізовував функцію - показувати кількість солдатів, яку відповідав кожен спрайт фішки військ.

Оскільки одна фішка військ зображувала 5000 солдатів, а на провінції могло бути значно більше солдатів, було важливо мати можливість точно відображати цю кількість. Тому, біля кожної фішки військ було додано текстовий елемент, що використовував TextMeshPro. (Рис. 3.7)

3.2.3 Елементи інтерфейсу

З метою покращення візуальної привабливості і поліпшення взаємодії гравця з ігровою сценою було використано ігровий інтерфейс. Головною метою

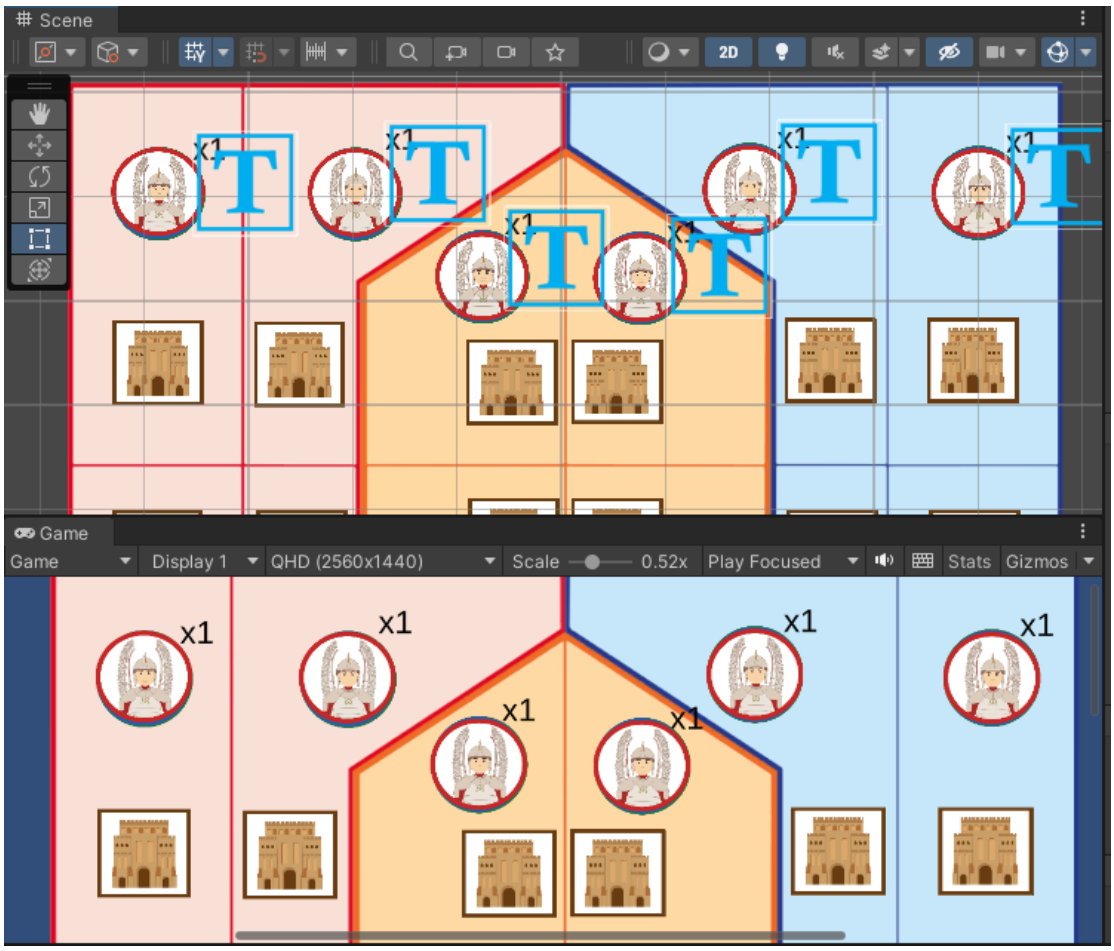


Рисунок 3.7 – Зображення лічильників біля фішок військ в редакторі і на ігровому полі

цього інтерфейсу було забезпечення чіткого та доступного способу відображення інформації на полі гри.

Для досягнення цієї мети, було прийнято рішення розмістити вивід інформації про провінцію з правого боку поля гри. (Рис. 3.8) Це дозволило гравцеві швидко та зрозуміло оцінити стан та властивості кожної провінції. Інформаційний вивід включав різноманітні дані, такі як назва провінції, належність, військові сили та інші важливі характеристики, які гравець міг використовувати для прийняття рішень у грі.

Крім того, була додана кнопка виходу з гри, щоб забезпечити гравцеві зручний спосіб завершення сеансу гри. Ця кнопка дозволяла гравцеві швидко припинити гру та повернутися до головного меню або вийти з програми. Це

забезпечувало зручність і контроль над процесом гри, дозволяючи гравцеві в будь-який момент вибрати оптимальний сценарій завершення гри.

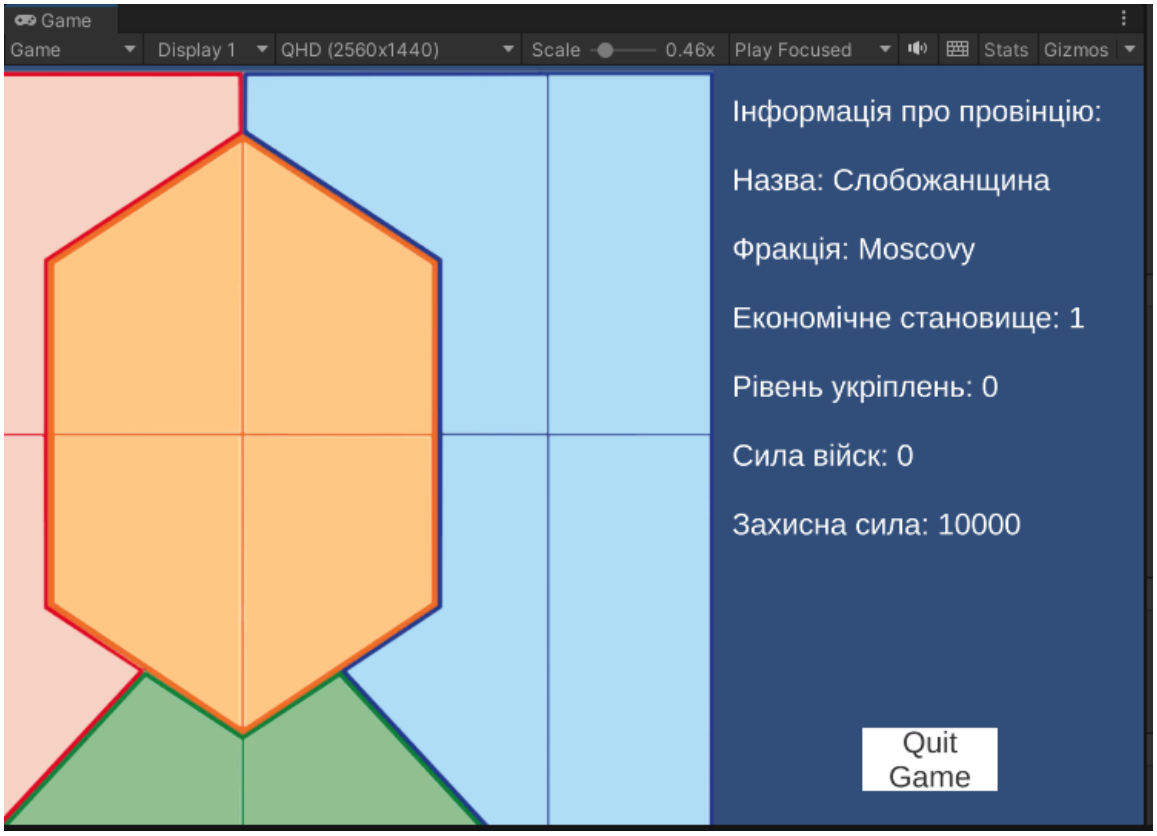


Рисунок 3.8 – Вигляд виводу інформації про провінцію і кнопка виходу

3.3 Розробка модуля, створення логіки відображення

Для обробки графічної інформації, та забезпечення правильного відображення ситуації на ігровому полі було створено такі скрипти графічного модуля:

- Province
- ProvinceBehavior
- ProvinceManager

3.3.1 Клас Province

Для коректного відображення спрайтів і тексту на ігровому полі були розроблені клас зберігання та обробки данних провінцій.

Інформація кожної провінції зберігається в класі Province. Це інформативний клас, який створює змінні для подальшого запису даних. (Рис. 3.9)

```
public class Province
{
    public string name;  Serializable

    9 usages  Mykytenko Vsevolod  5 exposing APIs
    public enum Factions
    {
        RichPospolita,
        Hetmanate,
        Moscovy,
        Khanate
    }

    public Factions faction;  Serializable

    public int economicState;  Serializable
    public int fortressState;  Serializable
    public int militaryPower;  Serializable
    public int provincePower;  Serializable
```

Рисунок 3.9 – Поля класу Province

Екземпляри цього класу можуть містити в собі дані про назву провінції – “name”, дані про фракцію яка займає цю провінцію – “Faction”, дані про економічну ситуацію – “economicState”, дані про рівень укріплень провінції – “fortressState”, дані про кількість військ у провінції – “militaryPower”, та дані про захисну силу провінції – “provincePower”.

Цей клас також має метод getStringName(), що повертає повну назву провінції в залежності від назви її шифру. (Рис. 3.10)

Рисунок 3.10 – Метод getStringName() класу Province

3.3.2 Клас ProvinceBehavior

Для обробки інформації з класу Province, програмного доступу до спарйта провінції та для створення спеціальної логіки в її межах було розроблено клас ProvinceBehavior.

При запуску сцени Unity визиває вбудований метод Awake(), де задається спрайт провінції та початкова її інформація через екземпляр класу Province. (Рис. 3.11)

```
public class ProvinceBehaviour : MonoBehaviour
{
    public Province province;  Serializable

    private SpriteRenderer _sprite;

    Event function  Mykytenko Vsevolod
    private void Awake()
    {
        _sprite = GetComponent<SpriteRenderer>();
        province.fortressState = 0;
        province.economicState = 1;
        province.militaryPower = 0;
        province.provincePower = province.militaryPower + 10000 * (province.fortressState + 1);
    }
```

Рисунок 3.11 – Метод Awake() класу ProvinceBehavior

Наступний метод цього класу TintColor (Color32 color) реалізовує зміну кольору спрайту провінції на той колір що він отримує як зміну Color32. (Рис. 3.12)

```
4 usages  new *  More...
public void TintColor(Color32 color)
{
    _sprite.color = (Color) color;
    _sprite.material.color = (Color) color;
}
Event function  Mykytenko Vsevolod
```

Рисунок 3.12 – Метод TintColor() класу ProvinceBehavior

Останій метод класу ShowProvinceInfo() реалізовує заповненя інформаційного поля інтерфейсу справа від ігрового поля інформацією з екземпляру класу Province. (Рис. 3.13)

Рисунок 3.13 – Метод ShowProvinceInfo() класу ProvinceBehavior

3.3.3 Клас ProvinceManager

Для керуваням змінами між всіма провінціями і між користувачами було написано клас ProvinceManager.

Як і в прикладі класу ProvinceBehavior при запуску ігрової сцени Unity викликає вбудований метод Awake(), а потім метод Start(), де створюються пустий список для подальшого заповненя об’єктами привінцій та екземпляр PhotonView(). Після чого викликається метод AddProvinceData().(Рис. 3.14)

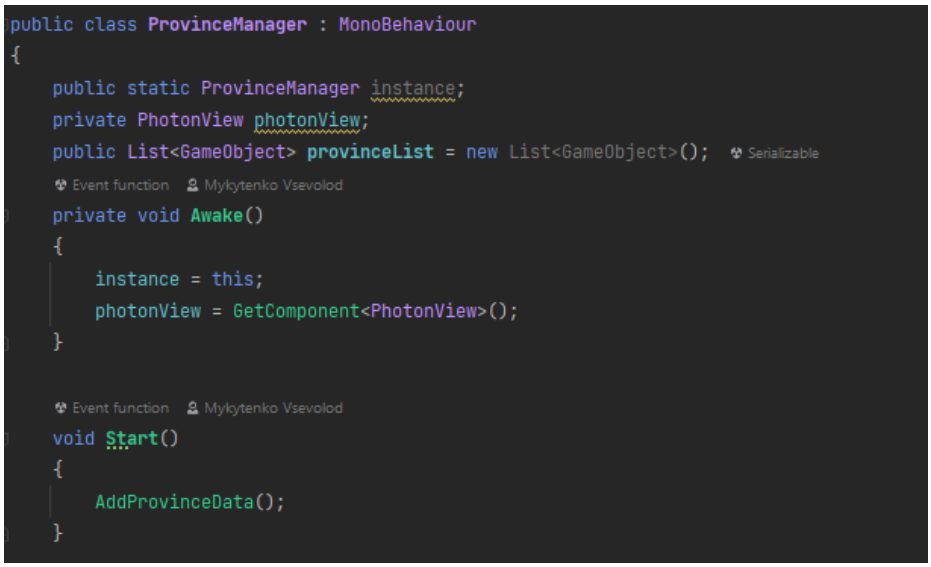


Рисунок 3.14 – Методи Awake() та Start() класу ProvinceManager

Метод AddProvinceData() в свою чергу заповнює список ігрових об’єктів знаходячи їх по списку назв спрайтів провінцій. Потім викликає метод TintProvinces().(Рис. 3.15)

```
1 usage   new *
void AddProvinceData ()
{
    string[] nameArray = {"RP1","RP2","RP3","RP4","G1","G2","G3","G4","M1","M2","M3","M4","H1","H2","H3","H4"};
    for(int i = 0; i < 16; i++)
    {
        provinceList.Add( item: GameObject.Find(nameArray[i]));
    }
    TintProvinces();
}
```

Рисунок 3.15 – Метод AddProvinceData() класу ProvinceManager

Метод Update() – це вбудований метод Unity, що визивається кожне обновленя фрейма. В цій реалізації він зчитує чи гравець натиснув кнопку пробілу і тоді через PhotonView() на одному клієнті гереруєтся масив randomArray, а на всіх клієнатах визиваеться методи RandomProvinceDataFill(int[] randomArray), та TintProvinces().(Рис. 3.16)

```
private void Update()
 {
    if (Input.GetKeyDown(KeyCode.Space))
    {
        int [] randomArray = new int[64];
        for (int i = 0; i < 16; i++)
        {
            randomArray[4*i] = Random.Range(0, 3);
            randomArray[4*i+1] = Random.Range(0, 3);
            randomArray[4*i+2] = Random.Range(0, 6);
            randomArray[4*i+3] = Random.Range(0, 4);
        }

        StringBuilder aaaaaaaa = new StringBuilder();
        foreach (var i int in randomArray)
        {
            aaaaaaaa.Append(i + " ");
        }
        Debug.Log( message: aaaaaaaa.ToString());
        photonView.RPC( methodName: "RandomProvinceDataFill", RpcTarget.All, randomArray);
        photonView.RPC( methodName: "TintProvinces", RpcTarget.All,  parameters: null);
        Debug.Log( message: "Provinces are tinted");
    }
}
```

Рисунок 3.16 – Метод Update() класу ProvinceManager

Метод RandomProvinceDataFill(int[] randomArray) проходиться по списку всіх об’єктів провінцій і заповнює інформації про провінції з випадкового

масиву randomArray. (Рис. 3.17)

```
[PunRPC]
Mykytenko Vsevolod
public void RandomProvinceDataFill(int[] randomArray)
{
    for (int i = 0; i < provinceList.Count; i++)
    {
        ProvinceBehaviour provinceBehaviour = provinceList[i].GetComponent<ProvinceBehaviour>();
        Debug.Log( message: provinceBehaviour.province.name.ToString());
        provinceBehaviour.province.fortressState = randomArray[4*i];
        provinceBehaviour.province.economicState = randomArray[4*i+1];
        provinceBehaviour.province.militaryPower = randomArray[4*i+2] * 5000;
        provinceBehaviour.province.provincePower = provinceBehaviour.province.militaryPower +
                                                   10000 * (provinceBehaviour.province.fortressState + 1);

        switch (randomArray[4*i+3])
        {
            case 0:
                provinceBehaviour.province.faction = Province.Factions.RichPospolita;
                break;
            case 1:
                provinceBehaviour.province.faction = Province.Factions.Hetmanate;
                break;
            case 2:
                provinceBehaviour.province.faction = Province.Factions.Moscovy;
                break;
            case 3:
                provinceBehaviour.province.faction = Province.Factions.Khanate;
                break;
        }
    }

}
```

Рисунок 3.17 – Метод RandomProvinceDataFill класу ProvinceManager

Метод TintProvinces() проходить по списку всіх об’єктів провінції. Для кожного з них отримує об’єкт ProvinceManager, та знаходить всі спрайти фішок. Далі приховує всі знайдені спрайти фішок і лічильника. (Рис. 3.18)

```
public void TintProvinces()
{
    for (int i = 0; i < provinceList.Count; i++)
    {
        ProvinceBehaviour provinceBehaviour = provinceList[i].GetComponent<ProvinceBehaviour>();
        GameObject soldierRP = GameObject.Find(provinceBehaviour.province.name + "_SoldierRP");
        GameObject soldierG = GameObject.Find(provinceBehaviour.province.name + "_SoldierG");
        GameObject soldierM = GameObject.Find(provinceBehaviour.province.name + "_SoldierM");
        GameObject soldierH = GameObject.Find(provinceBehaviour.province.name + "_SoldierH");
        GameObject soldierCounter = GameObject.Find(provinceBehaviour.province.name + "_SoldierCounter");
        soldierCounter.GetComponent<TextMeshProUGUI>().enabled = false;
        soldierRP.GetComponent<SpriteRenderer>().enabled = false;
        soldierG.GetComponent<SpriteRenderer>().enabled = false;
        soldierM.GetComponent<SpriteRenderer>().enabled = false;
        soldierH.GetComponent<SpriteRenderer>().enabled = false;
```

Рисунок 3.18 – Знаходженя об’єктів спрайтів в методі TintProvinces()

Далі цей метод залежно від інформації вказаної в провінції змінює колір спрайту і фішку військ залежно від фракції. (Рис. 3.19)

```
switch (provinceBehaviour.province.faction)
{
    case Province.Factions.RichPospolita:
        provinceBehaviour.TintColor(new Color32( r: 252, g: 239, b: 234, a: 255));
        if (provinceBehaviour.province.militaryPower != 0)
        {
            soldierRP.GetComponent<SpriteRenderer>().enabled = true;
        }
        break;

    case Province.Factions.Hetmanate:
        provinceBehaviour.TintColor(new Color32( r: 255, g: 235, b: 205, a: 255));
        if (provinceBehaviour.province.militaryPower != 0)
        {
            soldierG.GetComponent<SpriteRenderer>().enabled = true;
        }
        break;

    case Province.Factions.Moscovy:
        provinceBehaviour.TintColor(new Color32( r: 225, g: 243, b: 252, a: 255));
        if (provinceBehaviour.province.militaryPower != 0)
        {
            soldierM.GetComponent<SpriteRenderer>().enabled = true;
        }
        break;

    case Province.Factions.Khanate:
        provinceBehaviour.TintColor(new Color32( r: 211, g: 232, b: 211, a: 255));
        if (provinceBehaviour.province.militaryPower != 0)
        {
            soldierH.GetComponent<SpriteRenderer>().enabled = true;
        }
        break;
}
```

Рисунок 3.19 – Зміненя спрайтів залежно від фракції в методі TintProvinces()

Нарешті, цей агоритм завершує роботу виводячи лічильник військ та необхідну фішку фортеці. (Рис. 3.20)

```
if (provinceBehaviour.province.militaryPower != 0)
{
    soldierCounter.GetComponent<TextMeshProUGUI>().enabled = true;
    soldierCounter.GetComponent<TextMeshProUGUI>().text = "x" + provinceBehaviour.province.militaryPower / 5000;
}

GameObject tower1 = GameObject.Find(provinceBehaviour.province.name + "_Tower1");
GameObject tower2 = GameObject.Find(provinceBehaviour.province.name + "_Tower2");

switch (provinceBehaviour.province.fortressState)
{
    case 0:
        tower1.GetComponent<SpriteRenderer>().enabled = false;
        tower2.GetComponent<SpriteRenderer>().enabled = false;
        break;
    case 1:
        tower1.GetComponent<SpriteRenderer>().enabled = true;
        tower2.GetComponent<SpriteRenderer>().enabled = false;
        break;
    case 2:
        tower1.GetComponent<SpriteRenderer>().enabled = false;
        tower2.GetComponent<SpriteRenderer>().enabled = true;
        break;
}
```

Рисунок 3.20 – Вивід лічильника та спрайтів фортець в методі TintProvinces()

ВИСНОВОК ДО РОЗДІЛУ 3

У цьому розділі було детально розглянуто правила ігри "Гетьманщина 1654" та її основні механіки. Було проаналізовано процес перенесення оригінальних картинок в спрайти, використовуючи графічний редактор Sprite Editor. Цей процес включав завантаження окремих картинок та їх подальше розбиття на окремі спрайти.

Далі, було розглянуто механізм відображення спрайтів у відповідних провінціях гри. Кожна провінція мала власний спрайт, на якому розміщувалися фішки військ різних фракцій і фортець. З метою візуальної узгодженості і обмеження кількості фішок, які можуть знаходитись на одній провінції одночасно, спрайти було накладено один на одного.

Для кращого візуального представлення інформації про кількість фішок на провінції, було використано текстовий елемент відображення - TextMeshPro. Цей елемент дозволяв виводити лічильник кількості фішок неподалік кожної фішки військ, допомагаючи гравцеві легко орієнтуватись у ситуації.

Крім того, було додано ігровий інтерфейс, який забезпечував зручне взаємодію гравця з грою. Інтерфейс включав вивід інформації про провінцію, кнопку виходу з гри та інші функціональні елементи.

Для ефективної обробки графічної інформації та забезпечення правильного відображення ситуації на ігровому полі, було розроблено та використано кілька скриптів у графічному модулі:

1. Province (Провінція): Цей скрипт був відповідальний за представлення окремої провінції у грі.
2. ProvinceBehavior (Поведінка провінції): Цей скрипт визначав різні аспекти поведінки провінції на ігровій сцені.
3. ProvinceManager (Менеджер провінцій): Цей скрипт виконував функцію керування усіма провінціями у грі.

РОЗДІЛ 4. ДОСЛІДЖЕННЯ ТА АНАЛІЗ РОЗРОБЛЕНОЇ СИСТЕМИ

4.1 Інструкція користувача

При роботі з грою гравець може виконувати різні дії, реалізовані натисканям на інтерфейс гри чи натисканя клавіш на клавіатурі.Дії поділяються залежно від сцени де знаходиться користувач наступним чином:

- Головне меню гри:
- Меню приєднання до існуючих ігрових онлайн кімнат;
- Меню створення ігрової онлайн кімнати;
- Меню ігрової онлайн кімнати;
- Поле ігрової сессії.

4.1.1 Головне меню гри

Після завантаження гри, користувачу відкривається "Головне меню" ігри, де він зустрічає три доступні кнопки для взаємодії.

Перша кнопка називається "Find room" і дозволяє користувачеві переглянути список доступних онлайн-кімнат для приєднання з іншими гравцями.

Друга кнопка позначена як "Create room" і при натисканні на неї користувач відкриває меню для створення власної ігрової кімнати.

Нарешті, третя кнопка підписана як "Quit" і при натисканні на неї користувач завершує роботу гри і виходить з неї.

4.1.2 Меню приєднання до існуючих ігрових онлайн кімнат

Після відкриття меню, користувачу відображається список доступних ігрових кімнат, які були створені іншими гравцями. У цьому списку вказані назви кімнат, які були створені цими гравцями.

Вгорі списку є назва вкладки "Find Room", що позначає відкриту вкладку.

Нижче назви вкладки знаходиться сам список доступних ігрових онлайн кімнат, створених іншими гравцями, з вказаними назвами кімнат.

Користувач може приєднатися до обраної ігрової онлайн кімнати, натиснувши на її назву і стаючи її учасником.

Під списком доступних кімнат користувач може побачити кнопку "Back to menu", натискання якої поверне його до "Головного меню" гри.

4.1.3 Меню створення ігрової онлайн кімнати

У цьому меню, користувач має можливість створити свою власну ігрову онлайн кімнату, до якої зможуть приєднатись інші гравці.

У центрі екрану розміщене поле з написом "Room name...". Користувач може ввести назву своєї ігрової онлайн кімнати в цьому полі, і ця назва буде відображатись у списку доступних ігрових онлайн кімнат для інших користувачів. Поле дозволяє використовувати різні символи, включаючи верхній та нижній регістр літер, кирилицю, латиницю та цифри.

Якщо користувач бажає скинути назву своєї ігрової онлайн кімнати, він може натиснути клавішу ESC на клавіатурі. Якщо в поточній сесії користувач вже створював ігрову онлайн кімнату, то замість напису "Room name...", він побачить назву своєї попередньо створеної ігрової онлайн кімнати.

Нижче поля для введення назви ігрової онлайн кімнати розташована кнопка "Create room". Після натискання цієї кнопки, користувач створить власну ігрову онлайн кімнату з назвою, введеною у полі для назви кімнати, і ця

кімната буде відображатись у списку доступних ігрових онлайн кімнат для інших користувачів. Сам користувач стане учасником цієї ігрової онлайн кімнати після натискання кнопки "Create room" і потрапить в меню ігрової онлайн кімнати як "учасник, що створив кімнату". Якщо користувач не введе назву своєї ігрової онлайн кімнати, то створення кімнати не відбудеться при взаємодії з кнопкою "Create room".

4.1.4 Меню ігрової онлайн кімнати

Якщо користувач приєднався до ігрової онлайн кімнати як учасник, він побачить назву кімнати, до якої приєднався, вгорі екрану. Під назвою кімнати буде список інших учасників, де будуть відображені ніки інших користувачів. Інші учасники також побачать нік користувача, який приєднався. Нижче списку учасників буде розташована кнопка "Leave Room", яку користувач може натиснути, щоб повернутися до головного меню гри і покинути ігрову онлайн кімнату.

Якщо користувач приєднався до ігрової онлайн кімнати як "учасник, що створив кімнату", основне меню виглядатиме так само, як для учасника. Єдиною зміною буде наявність додаткової кнопки "Start Game" в правому нижньому куті екрану. Натиснувши на цю кнопку, користувач розпочне ігрову сесію для всіх гравців, які знаходяться в цій ігровій онлайн кімнаті.

Якщо учасник, який приєднався як "учасник, що створив кімнату", покине ігрову онлайн кімнату, натиснувши "Leave Room", і якщо він був єдиним учасником кімнати, то ця ігрова онлайн кімната буде видалена. У разі наявності інших учасників кімнати, вони зможуть продовжувати гру без нього.

4.1.5 Поле ігрової сессії

Після завантаження всіх учасників ігрової онлайн кімнати розпочинається гра в ігровій сесії.

На екрані користувач може побачити наступні елементи:

1. Фракції, представлені на ігровому полі:
    - RichPospolita (рожевий)
    - Moskovy (блакитний)
    - Hetmanate (помаранчевий)
    - Khanate (зелений)
2. На початку гри провінції розподілені між фракціями наступним чином:
    - RichPospolita: Варшава, Волинь, Бессарабія, Поділля
    - Moskovy: Смоленщина, Москва, Слобожанщина, Поволжя
    - Hetmanate: Київщина, Чернігівщина, Запорізька січ, Полтавщина
    - Khanate: Перекоп, Дике поле, Бахчисарай, Керч
3. З правого боку екрану користувач може побачити вікно "Інформація про провінцію", де надана детальна інформація про вибрану ним провінцію, включаючи:
    - Назва провінції
    - Фракція, якій належить провінція
    - Економічний стан провінції
    - Рівень укріплень провінції
    - Сила військ у провінції
    - Захисна сила провінції (базова захисна сила становить 10 000)

4.2 Подальша розробка гри

На даний момент версія гри має такі значні упущення, як:

- Відсутня логіка гри;
- Відсутнє поняття «гравець»;
- Відсутній ігровий текстовий/голосовий чат;
- Поле не поділено на фази гри;
- Відсутність можливості налаштування для пристроїв;
- Відсутність мультиплатформеності;
- Відсутність анімацій.

4.2.1 Логіка гри

Для забезпечення насолоди гравців активною грою у комп'ютерній грі жанру двовимірної стратегії, необхідно ретельно розробити складну та досконалу логіку та механіки, які відтворять їх ходи та дії з точністю та проникненням. У конкретному випадку гри, заснованої на правилах "Гетьманщини 1654", важливо виділити три основні фази, що надають їй глибину та реалістичність: фаза перемовин, воєнна фаза та економічна фаза, кожна з яких виконує свої функції та пропонує різні виклики та можливості для гравців.

Фаза перемовин - це дефінітивний період, коли гравці мають можливість взаємодіяти і спілкуватися один з одним, вести перемовини, проводити дипломатичні переговори та укладати союзи або домовленості. Ця фаза вимагає впровадження механізмів комунікації, забезпечення можливості обміну повідомленнями, пропозиціями та взаємодії з іншими гравцями, що реалістично відтворює процеси дипломатії та переговорів.

Воєнна фаза - це інтенсивний період, коли гравці займаються веденням військових дій та стратегічних битв, дотримуючись правил гри. Для досягнення успіху в цій фазі необхідно розробити складні механіки переміщення військ, проведення атак та оборони, розрахунок результатів битви, враховуючи різні фактори, такі як сила військ, територіальні переваги та стратегічні можливості.

Економічна фаза - це період, коли гравці займаються управлінням своїми ресурсами, грошима та територіями. Вони здобувають прибуток, розвивають економіку, займаються набором військ та плануванням стратегічних дій. У цій фазі необхідно реалізувати складну систему підрахунку грошових потоків, враховуючи витрати та доходи в кінці кожного ходу, а також впровадити механіки, що дозволяють гравцям ефективно управляти своїми ресурсами.

Основна мета розбивки гри на фази та розробки відповідних скриптів та механік полягає в тому, щоб надати гравцям можливість активно взаємодіяти з грою та між собою, використовуючи різноманітні стратегії та тактики, відповідно до правил гри "Гетьманщина 1654". Така розбивка на фази додає глибину та реалізм до геймплею, надаючи гравцям широкий спектр можливостей для стратегічного мислення та прийняття рішень у світі, що пронизаний історичними контекстами та викликами.

4.2.2 Відсутність поняття гравця

В даному моменті у грі відсутній інтерфейс, який обмежує можливості взаємодії та візуалізації для гравця. Щоб виправити цю ситуацію та забезпечити зручну геймплейну платформу, можна розробити індивідуальний інтерфейс гравця, що включатиме різноманітні елементи та функціональні можливості.

По-перше, створення інтерфейсу гравця може включати відображення грошових коштів, які гравець має на рахунку. Це дозволить гравцю вести контроль над своїми фінансовими ресурсами та приймати обґрунтовані рішення щодо їх використання.

По-друге, інтерфейс може надавати інформацію про підписані договори та статус відносин між країнами (гравцями). Наприклад, гравець може бачити, з якими країнами він уклав договори про військове союзництво, а також контролювати їх виконання та розробляти стратегію на основі цієї інформації.

По-третє, на ігровому полі можна розробити синхронізовану гру країн, що означає, що гравець зможе керувати лише арміями та фортецями на своїй власній території. Це створить більш реалістичне та стратегічне середовище, де гравець зосереджений на управлінні власною країною та розвитку її ресурсів.

Всі ці розширення інтерфейсу гравця сприятимуть покращенню його занурення в гру, забезпечуючи зручну навігацію, доступ до важливої інформації та можливість приймати стратегічні рішення на основі комплексного аналізу грошових ресурсів та міжнародних відносин.

4.2.3 Відсутність ігрового текстового/голосового чату

Для забезпечення коректної роботи першої фази гри, яка включає фазу перемовин між гравцями, важливим кроком буде розробка розширеного комунікаційного модуля. Такий модуль може включати текстовий чат, який дасть гравцям змогу спілкуватися та домовлятися відповідно до черги перемовин. Це створить можливість для гравців обмінюватися ідеями, задавати питання та узгоджувати свої дії у зручний та організований спосіб.

Крім текстового чату, додавання голосового чату в гру буде ще більш ефективним кроком у покращенні комунікації. Голосовий чат дозволить гравцям спілкуватися голосом, обговорювати стратегії та детально розробляти плани на партію. Це дозволить гравцям надавати більш точні та докладні команди, створюючи більш іммерсивну та реалістичну атмосферу гри.

Застосування розширеного комунікаційного модуля з текстовим та голосовим чатом у грі "Гетьманщина 1654" сприятиме покращенню комунікації між гравцями, підвищенню рівня співпраці та створенню більш глибокого та соціального геймплею. Такий модуль дозволить гравцям більш ефективно спілкуватися, координувати свої дії та насолоджуватися багатогранністю стратегічної гри.

4.2.4 Відсутність налаштувань у грі

Для досягнення більшої автентичності та іммерсії в грі "Гетьманщина 1654" є необхідність додати музичний супровід, який створить підходящу атмосферу та підсилить занурення гравців у гру. Розробка розширеного меню налаштувань є важливим кроком, оскільки дозволить гравцям налаштовувати звукові параметри за своїми потребами та уподобаннями зручним для них способом.

У меню налаштувань можна додати різноманітні опції для регулювання гучності голосового чату та музики. Гравці зможуть вибрати оптимальні налаштування, що дозволить їм збалансувати звукові елементи гри відповідно до їх власних уподобань та потреб.

Помимо цього, можна впровадити різноманітні звукові ефекти, пов'язані з діями гравців, такі як звуки переміщення військ, натискання кнопок миші або будівництво фортець. Ці звуки додадуть реалізму та доповнять візуальні ефекти та дії гравців. Регулювання гучності цих звукових ефектів також може бути доступне через меню налаштувань.

Додатково, можна розробити опцію вибору режиму гри на повний екран або у вікні, а також можливість регулювання розширення екрану. Це дозволить гравцям налаштовувати ігральне вікно згідно з власними вимогами та умовами їх комп'ютерної системи, забезпечуючи комфортну геймплейну зону.

Впровадження цих додаткових звукових та візуальних налаштувань через розширене меню гри "Гетьманщина 1654" покращить автентичність, зручність та налаштовування геймплею, надаючи гравцям більше контролю над звуковими та візуальними аспектами гри.

4.2.5 Усунення інших недоліків

Додавання анімацій подій на ігровому полі може принести значні покращення до гри, оскільки ці анімації викликають рух та візуальні ефекти, які сприяють кращому сприйняттю та розумінню подій, що відбуваються на полі гри. Крім того, анімації подій додають не лише функціональний аспект, а й естетичну складову до гри, оскільки візуальні ефекти та рухи створюють атмосферу, реалістичність та глибину, що робить геймплей більш насиченим та задовольняючим для гравців.

Додавання мультиплатформеності до мультиплеєрної стратегічної гри має велике значення та приносить численні переваги. Однією з головних переваг є збільшення аудиторії гри. Підтримка різних платформ, таких як ПК, консолі та мобільні пристрої, дозволяє привернути більше гравців до гри, оскільки кожен може обрати найзручнішу для себе платформу.

Мультиплатформеність забезпечує можливість взаємодії та гри разом незалежно від платформи, на якій знаходяться гравці. Це відкриває широкі можливості для збору гравців з різних пристроїв та об'єднання їх у спільних ігрових сесіях.

Загалом, мультиплатформеність є надзвичайно важливим аспектом розробки мультиплеєрних стратегічних ігор, оскільки вона забезпечує розширення аудиторії, покращення соціальної взаємодії, розширені можливості геймплею та зручність для гравців у виборі платформи, що викликає позитивний вплив на весь ігровий процес.

ВИСНОВОК ДО РОЗДІЛУ 4

В цьому розділі розглядалися аспекти взаємодії користувача з грою, зокрема процес приєднання до ігрової онлайн кімнати, перегляд інформації про учасників та покидання кімнати. Також було описано елементи гри, зокрема ігрове поле з провінціями та фракціями, і вікно з детальною інформацією про провінцію.

За допомогою цих описаних елементів, користувач може відчути себе часткою ігрового світу і залучитись до гри. Він може обирати роль учасника чи творця ігрової кімнати, бачити назву кімнати, список учасників та їх нікнейми. Кнопка "Leave Room" надає можливість зручно покинути кімнату і повернутись до головного меню.

Також в розділі вказується, що на даний момент версія гри має деякі значні недоліки, які необхідно врахувати для подальшого вдосконалення ігрового процесу та забезпечення насолоди гравців. Основні упущення включають відсутність логіки гри, відсутність поняття "гравець", відсутність ігрового текстового/голосового чату, відсутність розбиття поля на фази гри, відсутність можливості налаштування для пристроїв, відсутність мультиплатформеності та відсутність анімацій.

ВИСНОВКИ

Дана робота досліджує гозробку графічного модуля для гри стратегії «Гетьманщина 1654». Вона складається з 4 різних розділів.

Перший розділ містить інформацію про загальне поняття комп’ютерних ігор та їх класифікацію. Оскільки тема розробки графічний модуль, було розглянуто різні способи візуалізації інформації в різних іграх. Зокрема, текстовий спосіб візуалізації відрізняється своєю уявною формою, а 2-D графіка дозволяє створювати деталізовані образи. 3-D графіка забезпечує реалістичність та глибину, а VR візуалізація надає найбільш іммерсивний досвід.

У другому розділі були розглянуті рушії для розробки комп'ютерних ігор - Unreal Engine 5, Cry Engine 5, Source Engine та Unity. Unreal Engine 5 славиться високою якістю графіки та підтримкою VR, Cry Engine 5 - реалістичною 3D-графікою, Source Engine - швидкодією та низькими вимогами, а Unity - універсальністю та доступністю. Враховуючи вимоги проекту, такі як вільна ліцензія, навчальна документація, зручний інтерфейс та підтримка 2D проектів, найкращим вибором став Unity. Далі було проведено дослідження алгоритму створення проекту комп’ютерної гри. Розглянуто його ключові етапи, аналіз цільової аудиторії та визначення функціоналу проекту.

Третій розділ оглявав правила та механіки гри "Гетьманщина 1654". Процес перенесення картинок в спрайти було аналізовано, включаючи використання графічного редактора Sprite Editor. Механізм відображення спрайтів у провінціях гри було пояснено, включаючи накладання спрайтів один на одного. Для візуального представлення інформації про кількість фішок використовувався текстовий елемент TextMeshPro. Також були розроблені скрипти у графічному модулі, включаючи Province, ProvinceBehavior та ProvinceManager, для керування провінціями та їх поведінкою на ігровій сцені.

В останьому розділі були розглянуті аспекти взаємодії користувача з грою, зокрема приєднання до онлайн кімнати, перегляд інформації про учасників та покидання кімнати. Були описані елементи гри, такі як ігрове поле з провінціями та фракціями, а також вікно з детальною інформацією про провінцію. Проте, зазначено, що поточна версія гри має деякі значні недоліки, які потребують вдосконалення. Відсутність логіки гри, поняття "гравець", ігрового текстового/голосового чату, розбиття поля на фази гри, можливості налаштування для пристроїв, мультиплатформеності та анімацій є основними упущеннями, які потрібно врахувати для подальшого поліпшення ігрового процесу та задоволення гравців.

Завдання на дипломний бакалаврський проєкт виконано повністю. Матеріали відповідають вимогам, які висуваються до робіт такого роду на кафедрі ОТ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Комп'ютерна гра [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/PC_game.
2. Nystrom R. Game Programming Patterns / Robert Nystrom., 2014 – 354 с.
3. Візуалізація даних у іграх [Електронний ресурс] – Режим доступу до ресурсу: https://gameanalytics.com/blog/data-visualization-games/.
4. Halpern J. Developing 2D Games with Unity: Independent Game Programming with C# / Jared Halpern., 2018. – 408 с.
5. Jackson S. Mastering Unity 2D Game Development / Simon Jackson., 2014 – 474 с.
6. Ferrone H. Pro Unity 2D Game Development / Harrison Ferrone., 2018 – 405 с.
7. Smith G. Basic Math for Game Development with Unity 3D / G. Smith, K. Sung., 2018. – 279 с.
8. Ferrone H. Learning C# by Developing Games with Unity 2019: Code in C# and build 3D games with Unity, 4th Edition / Harrison Ferrone., 2019. – 342 с.
9. 3DTotal. Digital Painting Techniques / 3DTotal. // 3DTotal Publishing. – 2009. – №1. – С. 286.
10. Найкращі ігрові рушії [Електронний ресурс] – Режим доступу до ресурсу: https://www.gamedesigning.org/career/video-game-engines.
11. Unreal Engine [Електронний ресурс] – Режим доступу до ресурсу: https://www.unrealengine.com/.
12. Cry Engine [Електронний ресурс] – Режим доступу до ресурсу: https://www.cryengine.com/.
13. Source Engine [Електронний ресурс] – Режим доступу до ресурсу: https://developer.valvesoftware.com/wiki/Source

14. Unity Game Engine [Електронний ресурс] – Режим доступу до ресурсу: https://unity.com.
15. Unity Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unity3d.com/Manual/index.html.
16. David B. Hands-On Game Development Patterns with Unity / Baron David., 2019. – 116 с.
17. Buttfield-Addison P. Unity Game Development Cookbook: Essentials for Every Game / Paris Buttfield-Addison, Jon Manning, Tim Nugent., 2019 – 404 с.
18. Thorn A. Mastering Unity Scripting / Alan Thorn., 2015. – 380 с.
19. Ferrone H. Mastering Unity Scripting / Harrison Ferrone., 2016. – 379 с.
20. Hocking J. Unity in Action. Multiplatform game development in C# with Unity 5 / Joseph Hocking., 2015. – 352 с.

ДОДАТОК 1

Графічний модуль гри “Гетьманщина 1654”

Блок графічний (структурна схема)

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2023 р

					ІАЛЦ.467200.004 Д1	
		№ докум.	Підпис	Дата		
Розробив		Микитенко В. О.			**Графічний модуль гри “Гетьманщина 1654”** Блок графічний (структурна схема)	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Виноградов Ю.М.				
Н. Контр.		Виноградов Ю.М.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-91
Затвердив		Стіренко С. Г.				

ДОДАТОК 2

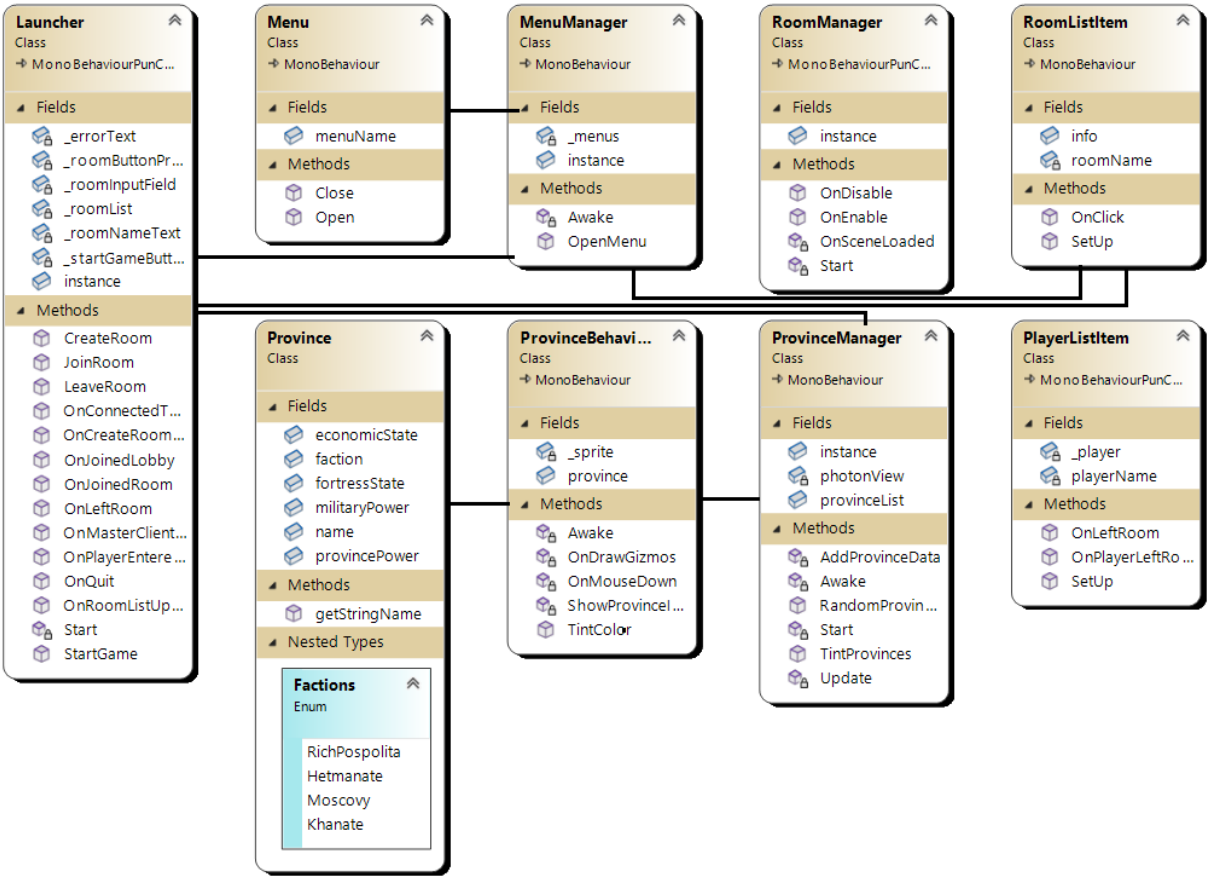
Графічний модуль гри “Гетьманщина 1654”

Блок графічний, діаграма класів
(функціональна схема)

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2023 р



					ІАЛЦ.467200.005 Д2	
		№ докум.	Підпис	Дата		
Розробив		Микитенко В.О.			**Графічний модуль гри «Гетьманщина 1654»** Блок графічний, діаграма класів (функціональна схема)	Літ. / Аркуш: 1 / Аркушів: 1
Перевірив		Виноградов Ю. М.				
Н. Контр.		Виноградов Ю. М.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-91
Затвердив		Стіренко С. Г.				

ДОДАТОК З

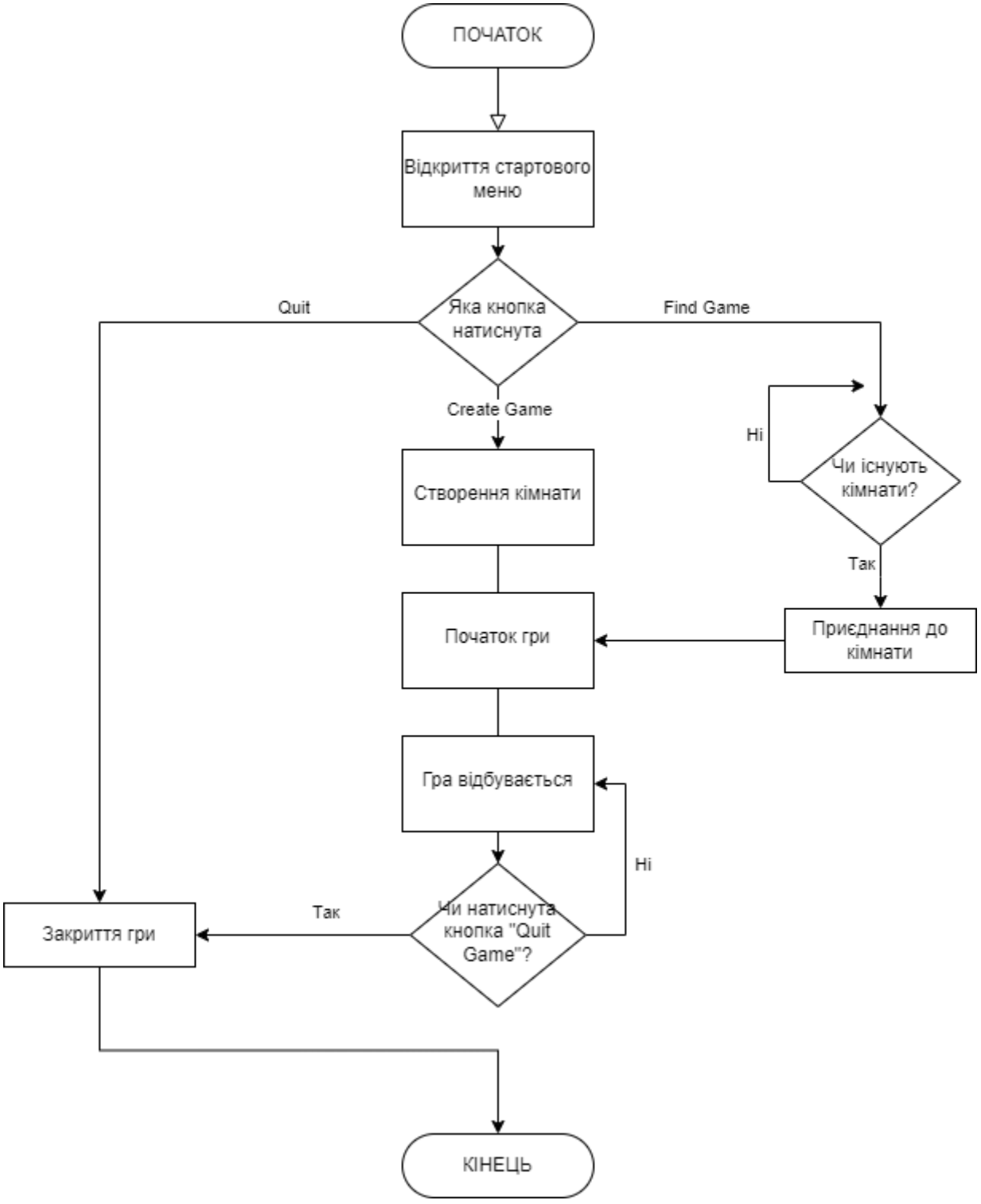
Графічний модуль гри “Гетьманщина 1654”

Блок графічний, алгоритм дій програмного забезпечення (принципова схема)

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2023 р



					ІАЛЦ.467200.006 Д3	
		№ докум.	Підпис	Дата		
Розробив		Микитенко В.О.			**Графічний модуль гри «Гетьманщина 1654»** Блок графічний, алгоритм дій програмного забезпечення (принципова схема)	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Виноградов Ю. М.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-91
Н. Контр.		Виноградов Ю. М.				
Затвердив		Стіренко С. Г.				

ДОДАТОК 4

Графічний модуль гри “Гетьманщина 1654”

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 15

Київ 2023 р

Файл Launcher.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Photon.Pun;
using Photon.Realtime;
using TMPro;
using Random = UnityEngine.Random;

public class Launcher : MonoBehaviourPunCallbacks
{
    public static Launcher instance;



    [SerializeField] private TMP_InputField _roomInputField;
    [SerializeField] private TMP_Text _errorText;
    [SerializeField] private TMP_Text _roomNameText;
    [SerializeField] private Transform _roomList;
    [SerializeField] private GameObject _roomButtonPrefab;
    [SerializeField] private Transform _playerList;
    [SerializeField] private GameObject _playerNamePrefab;
    [SerializeField] private GameObject _startGameButton;
    private void Start()
    {
        instance = this;
        Debug.Log("Joining Master-server");
        PhotonNetwork.ConnectUsingSettings();
        MenuManager.instance.OpenMenu("loading");
    }

    public void OnQuit()
    {
        Debug.Log("Quiting Master-server");
        Application.Quit();
    }
```

```
public override void OnConnectedToMaster()
{
    Debug.Log("Joined to Master-server");
    PhotonNetwork.JoinLobby();
    PhotonNetwork.AutomaticallySyncScene = true;
}

public override void OnJoinedLobby()
{
    Debug.Log("Joined Lobby");
    MenuManager.instance.OpenMenu("title");
    PhotonNetwork.NickName = "Player " + Random.Range(0, 1000).ToString("0000");
}

public void StartGame()
{
    PhotonNetwork.LoadLevel(1);
}

public void CreateRoom()
{
    if (string.IsNullOrEmpty(_roomInputField.text))
    {
        return;
    }

    RoomOptions roomOptions = new RoomOptions();
    roomOptions.MaxPlayers = 4;

    PhotonNetwork.CreateRoom(_roomInputField.text, roomOptions);
    MenuManager.instance.OpenMenu("loading");
}

public override void OnJoinedRoom()
{
    _roomNameText.text = PhotonNetwork.CurrentRoom.Name;
    MenuManager.instance.OpenMenu("room");

    Player[] players = PhotonNetwork.PlayerList;

    for (int i = 0; i < _playerList.childCount; i++)
    {
        Destroy(_playerList.GetChild(i).gameObject);
```

```
            }

            for (int i = 0; i < players.Length; i++)
            {
                Instantiate(_playerNamePrefab,
_playerList).GetComponent<PlayerListItem>().SetUp(players[i]);
            }

            _startGameButton.SetActive(PhotonNetwork.IsMasterClient);
        }

        public override void OnMasterClientSwitched(Player newMasterClient)
        {
            _startGameButton.SetActive(PhotonNetwork.IsMasterClient);
        }

        public override void OnCreateRoomFailed(short returnCode, string message)
        {
            _errorText.text = "Error: " + message;
            MenuManager.instance.OpenMenu("error");
        }

        public void LeaveRoom()
        {
            PhotonNetwork.LeaveRoom();
            MenuManager.instance.OpenMenu("loading");
        }

        public override void OnLeftRoom()
        {
            MenuManager.instance.OpenMenu("title");
        }

        public void JoinRoom(RoomInfo info)
        {
            PhotonNetwork.JoinRoom(info.Name);
            MenuManager.instance.OpenMenu("loading");
        }

        public override void OnRoomListUpdate(List<RoomInfo> roomList)
        {
            for (int i = 0; i < _roomList.childCount; i++)
            {
```

```
                Destroy(_roomList.GetChild(i).gameObject);
            }
            for (int i = 0; i < roomList.Count; i++)
            {
                if(roomList[i].RemovedFromList)
                    continue;
                Instantiate(_roomButtonPrefab,
_roomList).GetComponent<RoomListItem>().SetUp(roomList[i]);
            }
        }

        public override void OnPlayerEnteredRoom(Player player)
        {
            Instantiate(_playerNamePrefab,
_playerList).GetComponent<PlayerListItem>().SetUp(player);
        }
    }
```

Файл Menu.cs

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

public class Menu : MonoBehaviour
{
    public string menuName;

    public void Open()
    {
        gameObject.SetActive(true);
    }

    public void Close()
    {
        gameObject.SetActive(false);
    }
}
```

Файл MenuManager.cs

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

```
public class MenuManager : MonoBehaviour
{
    public static MenuManager instance;

    [SerializeField] private List<Menu> _menus;

    public void OpenMenu(string menuName)
    {
        foreach (Menu menu in _menus)
        {
            if (menu.menuName == menuName)
            {
                menu.Open();
            }
            else
            {
                menu.Close();
            }
        }
    }

    private void Awake()
    {
        instance = this;
    }
}
```

Файл PlayerListItem.cs

```
using System.Collections;
using System.Collections.Generic;
using Photon.Pun;
using Photon.Realtime;
using TMPro;
using UnityEngine;

public class PlayerListItem : MonoBehaviourPunCallbacks
{
    private Player _player;
    [SerializeField] private TMP_Text playerName;

    public void SetUp(Player player)
    {
```

```
        _player = player;
        playerName.text = player.NickName;
    }

    public override void OnPlayerLeftRoom(Player otherPlayer)
    {
        if (_player == otherPlayer)
            Destroy(gameObject);
    }

    public override void OnLeftRoom()
    {
        Destroy(gameObject);
    }
}
```

Файл RoomListItem.cs

```
using System.Collections;
using System.Collections.Generic;
using Photon.Realtime;
using TMPro;
using UnityEngine;

public class RoomListItem : MonoBehaviour
{
    [SerializeField] private TMP_Text roomName;

    public RoomInfo info;


    public void SetUp(RoomInfo roomInfo)
    {
        info = roomInfo;
        roomName.text = info.Name + "  " + info.PlayerCount + "/" + info.MaxPlayers;
    }

    public void OnClick()
    {
        Launcher.instance.JoinRoom(info);
        if (info.PlayerCount == info.MaxPlayers)
        {
            MenuManager.instance.OpenMenu("error");
        }
```

```
        }
    }
```

Файл RoomManager.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using Photon.Realtime;
using Photon.Pun;
using UnityEngine;
using UnityEngine.SceneManagement;

public class RoomManager : MonoBehaviourPunCallbacks
{
    public static RoomManager instance;
    // Start is called before the first frame update
    void Start()
    {
        if (instance)
        {
            Destroy(gameObject);
            return;
        }
        DontDestroyOnLoad(gameObject);
        instance = this;
    }

    public override void OnEnable()
    {
        base.OnEnable();
        SceneManager.sceneLoaded += OnSceneLoaded;
    }

    private void OnSceneLoaded(Scene scene, LoadSceneMode loadSceneMode)
    {
        if (scene.buildIndex == 1)
        {
            PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs",  "PlayerManager"),
Vector3.zero, Quaternion.identity);
        }
    }
```

```
    public override void OnDisable()
    {
        base.OnDisable();
    }
}
```

Файл Province.cs

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

[System.Serializable]

public class Province
{
    public string name;

    public enum Factions
    {
        RichPospolita,
        Hetmanate,
        Moscovy,
        Khanate
    }

    public Factions faction;

    public int economicState;
    public int fortressState;
    public int militaryPower;
    public int provincePower;

    public string getStringName()
    {
        var names = new Dictionary<string, string>
        {
            {"RP1", "Варшава"},
            {"RP2", "Волинь"},
            {"RP3", "Бессарабія"},
            {"RP4", "Поділля"},
            {"G1", "Київщина"},
            {"G2", "Чернігівщина"},
            {"G3", "Запорізька січ"},
```

```
                {"G4", "Полтавщина"},
                {"M1", "Смоленщина"},
                {"M2", "Москва"},
                {"M3", "Слобожанщина"},
                {"M4", "Поволжя"},
                {"H1", "Перекоп"},
                {"H2", "Дике поле"},
                {"H3", "Бахчисарай"},
                {"H4", "Керч"}
            };
            return names[name];
        }
    }
```

Файл ProvinceBehaviour.cs

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

    [RequireComponent(typeof(PolygonCollider2D))]

    public class ProvinceBehaviour : MonoBehaviour
    {
        public Province province;

        private SpriteRenderer _sprite;

        private void Awake()
        {
            _sprite = GetComponent<SpriteRenderer>();
            province.fortressState = 0;
            province.economicState = 1;
            province.militaryPower = 0;
            province.provincePower     =     province.militaryPower     +     10000     *
(province.fortressState + 1);
        }

        public void TintColor(Color32 color)
        {
```

```
            _sprite.color = color;
            _sprite.material.color = color;
        }
        private void OnMouseDown()
        {
            ShowProvinceInfo();
       }


        private void ShowProvinceInfo()
        {
            GameObject provinceDataField = GameObject.Find("ProvinceDataField");
            provinceDataField.GetComponent<TextMeshProUGUI>().text   =   "Інформація   про
провінцію:<br><br>Назва: "+ province.getStringName() +
                                                                    "<br><br>Фракція: "+
province.faction +
                                                                    "<br><br>Економічне
становище: "+ province.economicState +
                                                                    "<br><br>Рівень
укріплень: "+ province.fortressState +
                                                                    "<br><br>Сила військ:
"+ province.militaryPower +
                                                                    "<br><br>Захисна
сила: "+province.provincePower;
        }
        private void OnDrawGizmos()
        {
            province.name = this.name;
            this.tag = "Province";
            province.provincePower     =     province.militaryPower     +     10000     *
(province.fortressState + 1);
        }
    }
```

Файл ProvinceManager.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Photon.Pun;
using Photon.Realtime;
using UnityEngine.Networking;
```

```
using TMPro;
using Unity.VisualScripting;
using UnityEngine;
using Random = UnityEngine.Random;


public class ProvinceManager : MonoBehaviour
{
    public static ProvinceManager instance;
    private PhotonView photonView;
    public List<GameObject> provinceList = new List<GameObject>();
    private void Awake()
    {
        instance = this;
        photonView = GetComponent<PhotonView>();
    }

    void Start()
    {
        AddProvinceData();
    }

    void AddProvinceData ()
    {
        string[] nameArray =
{"RP1","RP2","RP3","RP4","G1","G2","G3","G4","M1","M2","M3","M4","H1","H2","H3","H4"};
        for(int i = 0; i < 16; i++)
        {
            provinceList.Add(GameObject.Find(nameArray[i]));
        }
        TintProvinces();
    }

    private void Update()
     {
         if (Input.GetKeyDown(KeyCode.Space))
         {
             int [] randomArray = new int[64];
             for (int i = 0; i < 16; i++)
             {
                 randomArray[4*i] = Random.Range(0, 3);
                 randomArray[4*i+1] = Random.Range(0, 3);
                 randomArray[4*i+2] = Random.Range(0, 6);
```

```
                    randomArray[4*i+3] = Random.Range(0, 4);
                }

                StringBuilder aaaaaaaa = new StringBuilder();
                foreach (var i in randomArray)
                {
                    aaaaaaaa.Append(i + " ");
                }
                Debug.Log(aaaaaaaa.ToString());
                photonView.RPC("RandomProvinceDataFill", RpcTarget.All, randomArray);
                photonView.RPC("TintProvinces", RpcTarget.All, null);
                Debug.Log("Provinces are tinted");
            }
        }



        [PunRPC]
        public void RandomProvinceDataFill(int[] randomArray)
        {
            for (int i = 0; i < provinceList.Count; i++)
            {
                ProvinceBehaviour                    provinceBehaviour                    =
provinceList[i].GetComponent<ProvinceBehaviour>();
                Debug.Log(provinceBehaviour.province.name.ToString());
                provinceBehaviour.province.fortressState = randomArray[4*i];
                provinceBehaviour.province.economicState = randomArray[4*i+1];
                provinceBehaviour.province.militaryPower = randomArray[4*i+2] * 5000;
                provinceBehaviour.province.provincePower                                =
provinceBehaviour.province.militaryPower +
                                                                10000                       *
(provinceBehaviour.province.fortressState + 1);

                switch (randomArray[4*i+3])
                {
                    case 0:
                        provinceBehaviour.province.faction                              =
Province.Factions.RichPospolita;
                        break;
                    case 1:
                        provinceBehaviour.province.faction                              =
Province.Factions.Hetmanate;
                        break;
```

```
                    case 2:
                        provinceBehaviour.province.faction = Province.Factions.Moscovy;
                        break;
                    case 3:
                        provinceBehaviour.province.faction = Province.Factions.Khanate;
                        break;
                }
            }

        }

        [PunRPC]
        public void TintProvinces()
        {
            for (int i = 0; i < provinceList.Count; i++)
            {
                ProvinceBehaviour provinceBehaviour = provinceList[i].GetComponent<ProvinceBehaviour>();
                GameObject soldierRP = GameObject.Find(provinceBehaviour.province.name + "_SoldierRP");
                GameObject soldierG = GameObject.Find(provinceBehaviour.province.name + "_SoldierG");
                GameObject soldierM = GameObject.Find(provinceBehaviour.province.name + "_SoldierM");
                GameObject soldierH = GameObject.Find(provinceBehaviour.province.name + "_SoldierH");
                GameObject soldierCounter = GameObject.Find(provinceBehaviour.province.name + "_SoldierCounter");
                soldierCounter.GetComponent<TextMeshProUGUI>().enabled = false;
                soldierRP.GetComponent<SpriteRenderer>().enabled = false;
                soldierG.GetComponent<SpriteRenderer>().enabled = false;
                soldierM.GetComponent<SpriteRenderer>().enabled = false;
                soldierH.GetComponent<SpriteRenderer>().enabled = false;

                switch (provinceBehaviour.province.faction)
                {
                    case Province.Factions.RichPospolita:
                        provinceBehaviour.TintColor(new Color32(252, 239, 234, 255));
                        if (provinceBehaviour.province.militaryPower != 0)
                        {
                            soldierRP.GetComponent<SpriteRenderer>().enabled = true;
                        }
                        break;
```

```
                    case Province.Factions.Hetmanate:
                        provinceBehaviour.TintColor(new Color32(255, 235, 205, 255));
                        if (provinceBehaviour.province.militaryPower != 0)
                        {
                            soldierG.GetComponent<SpriteRenderer>().enabled = true;
                        }
                        break;

                    case Province.Factions.Moscovy:
                        provinceBehaviour.TintColor(new Color32(225, 243, 252, 255));
                        if (provinceBehaviour.province.militaryPower != 0)
                        {
                            soldierM.GetComponent<SpriteRenderer>().enabled = true;
                        }
                        break;

                    case Province.Factions.Khanate:
                        provinceBehaviour.TintColor(new Color32(211, 232, 211, 255));
                        if (provinceBehaviour.province.militaryPower != 0)
                        {
                            soldierH.GetComponent<SpriteRenderer>().enabled = true;
                        }
                        break;
                }

                if (provinceBehaviour.province.militaryPower != 0)
                {
                    soldierCounter.GetComponent<TextMeshProUGUI>().enabled = true;
                    soldierCounter.GetComponent<TextMeshProUGUI>().text    =    "x"    +
provinceBehaviour.province.militaryPower / 5000;
                }

                GameObject  tower1  =  GameObject.Find(provinceBehaviour.province.name  +
"_Tower1");
                GameObject  tower2  =  GameObject.Find(provinceBehaviour.province.name  +
"_Tower2");

                switch (provinceBehaviour.province.fortressState)
                {
                    case 0:
                        tower1.GetComponent<SpriteRenderer>().enabled = false;
                        tower2.GetComponent<SpriteRenderer>().enabled = false;
```

```
                    break;
                case 1:
                    tower1.GetComponent<SpriteRenderer>().enabled = true;
                    tower2.GetComponent<SpriteRenderer>().enabled = false;
                    break;
                case 2:
                    tower1.GetComponent<SpriteRenderer>().enabled = false;
                    tower2.GetComponent<SpriteRenderer>().enabled = true;
                    break;
            }
        }
    }
}
```