Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет інформаційних радіотехнологій та технічного захисту інформації
(повна назва)

Кафедра медіаінженерії та інформаційних радіоелектронних систем
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти *другий (магістерський)*
(позначення документа)

Комплексна тема. Розробка ігрового контенту.
Програмна частина на ігровому рушії Unity
(тема)

Виконав:
студент 2 курсу, групи *СТМм-22-1*
Михайло МЕДВЕДЄВ
(прізвище, ініціали)

Спеціальність *171 Електроніка*
(код і повна назва спеціальності)

Тип програми *освітньо-професійна*
(освітньо-професійна або освітньо-наукова)

Освітня програма *Системи, технології і комп'ютерні засоби мультимедіа*
(повна назва освітньої програми)

Керівник *ст.викл. Олександр ЯКОВЧЕНКО*
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Володимир КАРТАШОВ
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційних радіотехнологій та технічного захисту інформації
Кафедра Медіаінженерії та інформаційних радіоелектронних систем

Рівень вищої освіти другий (магістерський)
Спеціальність 171 Електроніка
(код і повна назва)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма "Системи, технології і комп'ютерні засоби мультимедіа"

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Студентові *Медведєву Михайлу Михайловичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Комплексна тема. Розробка ігрового контенту. Програмна частина на ігровому рушії Unity*
затверджена наказом по університету від " *20* " *11* *2023 р. №* *1371 СТ*
2. Термін подання студентом роботи *08.01.2024 р.*
3. Вихідні дані до проекту (роботи)
1. Аналіз програмно-апаратних засобів створення ігрового контенту
2. Створення концепту ігрового контенту
3. Розробка ігровогу додатку з використанням можливостей ігрового рушія Unity 3D

4. Перелік питань, що потрібно опрацювати в роботі
ВСТУП
1. Аналітичний огляд та обгрунтування вибору ігрового рушія
2. Розробка концепт документу ігрового додатку
3. Розробка алгоритму ігрового контенту
ВИСНОВКИ
ПЕРЕЛІК ПОСИЛАНЬ
ДОДАТКИ

5. Перелік графічного матеріалу із зазначенням обов'язкових креслеників, схем, плакатів, комп'ютерних ілюстрацій:
1. Постановка задачі, 2. Обгрунтуваня вибору ігрового рушія, 3. Використання додаткових плагінів проекті, 4. Приклад використання Zenject, 5. Приклад використання DoTween, 6. Приклад компонентів, 7. Приклад конфігурації, 8. Вигляд ігрової сцени в едіторі, 9. Вигляд ігрової сцени в плеймоді, 10. Можливості для будівництва, 11. Можливості для розширення.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Аналітичний огляд та обгрунтування вибору ігрового рушія	20.11.23–28.11.23	
2.	Розробка концепт документу ігрового додатку	21.11.23–28.11.23	
3.	Розробка алгоритму ігрвого контенту	23.11.23–02.12.23	
4.	Графічна частина роботи	15.12.21–20.12.23	
5.	Перевірка керівником	20.12.23-24.12.23	
6.	Перевірка на академічний плагіат	24.12.23-26.12.23	
7.	Перевірка завідувачем кафедри, рецензування	27.12.23-31.12.23	

Дата видачі завдання *20.11.2023 р.*

Студент *Михайло МЕДВЕДЄВ*
(підпис)

Керівник роботи _____ *ст.викл. Олександр ЯКОВЧЕНКО*
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи має: 69 с., 52 рис., 37 джерел.

МОБІЛЬНІ ПРИСТРОЇ, ІНТЕРФЕЙС, РЕАЛІЗАЦІЯ, ІНІЦІАЛІЗАЦІЯ, ГЕЙМПЛЕЙ, РУШІЙ, РОЗРОБКА, ТЕСТУВАННЯ.

Об'єкт дослідження – розробка гри на мобільних пристроях.

Предмет дослідження – інтерфейс та геймплей мобільних ігор.

Мета кваліфікаційної роботи – створити оптимізований та захоплюючий геймплей для мобільної гри засобами ігрового рушія Unity.

Методи дослідження – аналіз сучасних трендів у галузі мобільних ігор, дослідження популярних гейм-движків, тестування ефективності інтерфейсу та геймплею на різних пристроях, аналіз зворотного зв'язку від гравців.

У даній роботі надано огляд класифікації та особливостей мобільних ігор, аналіз популярних гейм-движків, вивчено особливості інтерфейсу та геймплею на мобільних пристроях, розроблено оптимізований геймплей для мобільної гри, включаючи адаптивний інтерфейс та взаємодію з різними аспектами мобільних платформ. Використані методи дозволили створити ефективний геймплей для мобільних пристроїв, враховуючи особливості їхнього функціонування та екранний розмір.

ABSTRACT

Explanatory note of the qualification work has: 69 p., 52 figures, 37 sources.

MOBILE DEVICES, INTERFACE, IMPLEMENTATION, INITIALIZATION, GAMEPLAY, ENGINE, DEVELOPMENT, TESTING.

The object of research – development of a game on mobile devices.

The subject of research – interface and gameplay of mobile games.

The purpose of the qualification work – create an optimized and exciting gameplay for a mobile game.

Research methods – analysis of modern trends in the field of mobile games, research of popular game engines, testing the effectiveness of the interface and gameplay on different devices, analysis of player feedback.

This paper provides an overview of the classification and features of mobile games, an analysis of popular game engines, a study of interface and gameplay features on mobile devices, and the development of optimized gameplay for mobile games, including an adaptive interface and interaction with various aspects of mobile platforms. The methods used made it possible to create an effective and engaging gameplay for mobile devices, taking into account the peculiarities of their functioning and screen size.

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

2D – двовимірний опис об’єкта;

3D – тривимірний опис об’єкта, геометрична модель матеріального світу;

4D – математичний об'єкт, що узагальнює властивості тривимірного простору;

Mesh – це набір полігонів, що визначає форму об'єкта у тривимірному просторі;

UE - Unreal Engine;

Ассет, asset - цифровий об'єкт, який представляє частину ігрового контенту і має деякі властивості;

ООП – об’єктно орієнтоване програмування;

Ріггінг - процес створення внутрішньої структури або "скелету" для 3D-моделі, який дозволяє контролювати рухи цієї моделі;

Семпл - це фрагмент аудіозапису, який використовується в новому контексті чи композиції;

Скрипт, script – це невелика програма, яка містить послідовність дій, створених;

Спрайт, sprite – двовимірне зображення в комп’ютерній графіці;

Тайл, tile – поодинока графічна плитка, що є частиною цілісного зображення;

Таймлайн, timeline — створена у хронологічному порядку подієва стрічка;

ШІ – штучний інтелект.

ЗМІСТ

ВСТУП

В сучасному інформаційному просторі розвиток відомостей та технологій набуває неабиякого обсягу, а ігрова індустрія виявляє тенденції стрімкого росту, привертаючи увагу не лише геймерів, а й розробників. У цьому контексті особливо важливим є використання сучасних інструментів для створення високоякісних та захоплюючих ігрових досвідів.

Серед таких інструментів виділяється Unity – потужне середовище для розробки ігор, яке знайшло широке визнання у галузі. Своєрідність Unity полягає в його доступності для розробників різного рівня досвіду та можливості створення якісних продуктів для різних платформ.

Метою даної кваліфікаційної роботи є розробка гри на платформі Unity з використанням сучасних технік та підходів. Досліджуючи сучасний стан індустрії розробки ігор та аналізуючи досягнення провідних наукових установ, ми ставимо перед собою завдання створити продукт, який не лише відповідає вимогам ринку, але й вносить свій унікальний внесок у світ геймдеву.

Ця дипломна робота спрямована на вивчення можливостей Unity для створення ігор та розробки власного ігрового продукту з використанням передових методик та рішень. Окрім того, вона взаємодіє з іншими дослідженнями в галузі та допомагає впроваджувати нові ідеї та технології в сучасний ландшафт ігрової розробки.

Графічні та технічні можливості Unity надають безмежні можливості для реалізації творчих ідей у галузі розробки ігор. Провідні гравці галузі вже визначили ключові тенденції, але існує великий простір для новаторських підходів та вдосконалення ігрових взаємодій, тому тема цієї дипломної работи є актуальною.

1 АНАЛІТИЧНИЙ ОГЛЯД ТА ОБГРУНТУВАННЯ ВИБОРУ ІГРОВОГО РУШІЯ

1.1 Аналіз ігрового рушія Unreal Engine

Unreal Engine - це високопродуктивний інтегрований графічний рушій та середовище розробки від Epic Games. Вперше представлений у 1998 році, Unreal Engine став одним з найпопулярніших інструментів для розробки відеоігор, візуальних ефектів, а також інтерактивних віртуальних просторів.

Його основна функціональність включає графічний рушій, який дозволяє створювати деталізовані та реалістичні графічні ефекти, фізичні розрахунки, штучний інтелект, аудіосистему, а також інструменти для роботи з анімацією та світловими ефектами. Unreal Engine підтримує багато платформ, включаючи ПК, консолі, мобільні пристрої та віртуальну реальність.

Unreal Engine (рис. 1.1) використовується як великими геймдев-студіями для розробки високобюджетних ігор, так і невеликими командами та індивідуальними розробниками для творення ігор різних жанрів та рівнів складності. Відомий своєю гнучкістю та потужністю, Unreal Engine забезпечує високий рівень якості графіки та ігрового досвіду [1].

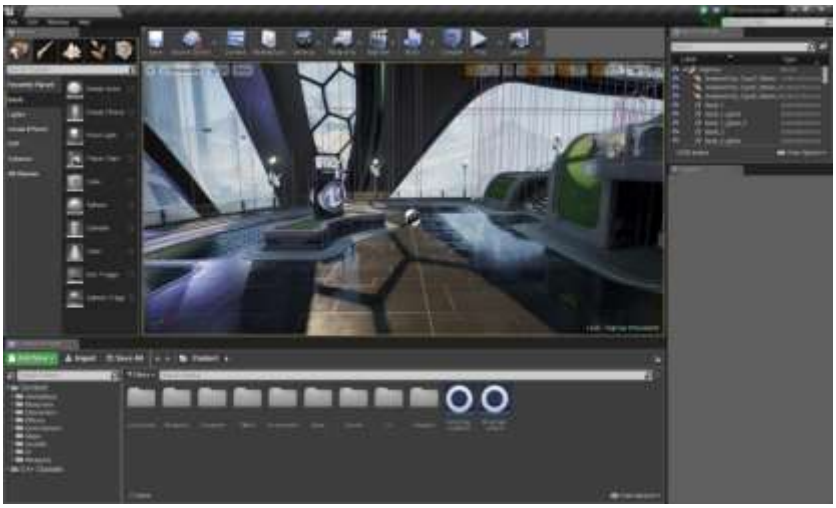


Рисунок 1.1 – Інтерфейс програми Unreal Engine[1]

Переваги Unreal Engine:

- графічний рушій високої якості. Unreal Engine вражає великою графічною якістю, дозволяючи розробникам створювати деталізовані та реалістичні візуальні ефекти;
- реалістична фізика. Рушій надає потужні засоби для реалізації фізичної поведінки об'єктів, що сприяє створенню реалістичних ігрових світів;
- широкий функціонал. Unreal Engine має різноманітні вбудовані інструменти для створення різноманітних ігрових елементів, включаючи штучний інтелект, фізичні ефекти, звук та інше;
- підтримка великих проектів. Інженерна структура Unreal Engine дозволяє ефективно працювати над великими та складними проектами, що робить його популярним для великих геймдев-студій;
- спільнота та документація. Unreal Engine має активну спільноту розробників та надає широкий спектр документації та навчальних ресурсів.

Недоліки Unreal Engine:

- складність вивчення. Для новачків Unreal Engine може здаватися складним у вивченні через велику кількість функцій та можливостей;
- великий обсяг файлів. Проекти, розроблені в Unreal Engine, можуть мати великий обсяг файлів, що може впливати на час завантаження та роботи над проектом;
- ліцензійні витрати. Для комерційних проектів можуть виникнути ліцензійні витрати, що робить Unreal Engine менш доступним для невеликих розробників;
- системні вимоги. Для ефективної роботи з Unreal Engine потрібні потужні обчислювальні ресурси, що може бути стримуючим фактором для користувачів зі слабкими комп’ютерами. Мінімальні системні вимоги для Unreal Engine 4: операційна система Windows 7 64-біт або

macOS 10.15.6; процесор Quad-core Intel or AMD, 2.5 GHz чи швидше; оперативна пам'ять 8 GB RAM; графічна карта: NVIDIA GeForce 470 GTX або AMD Radeon 6870 HD серії або еквівалент, з DX10 підтримкою та 1 GB VRAM; DirectX. Версія 11; [1].

1.2 Аналіз ігрового рушія Godot Engine

Godot Engine - це відкритий інтегрований інструмент та рушій для розробки ігор та інших інтерактивних застосунків (рис 1.2). Розроблений Juan Linietsky у 2007 році, Godot широко використовується в галузі геймдеву, а також для створення анімацій, симуляцій та інших інтерактивних візуальних проектів [2].

Рисунок 1.2 – Інтерфейс Godot Engine [2]

Переваги Godot Engine:

– потужний інструментарій моделювання. 3ds Max пропонує широкий набір інструментів для створення складних 3D-моделей, включаючи підтримку полігонального, сабдивізійного та об'ємного моделювання. Це робить його придатним до створення різноманітних об'єктів, від персонажів до архітектурних конструкцій;

- кросплатформенність. Розроблені в Godot ігри можуть бути легко виведені на різні платформи, включаючи ПК, консолі, мобільні пристрої та веб;
- легкий для вивчення. Godot відомий своєю простотою та зрозумілістю, що полегшує вивчення новачкам у галузі геймдеву;
- вбудовані інструменти. Godot надає ряд вбудованих інструментів для створення графіки, анімацій, фізичної моделювання та штучного інтелекту;

Недоліки Godot Engine:

- менша кількість ресурсів для навчання. У порівнянні з деякими іншими рушіями, Godot може мати меншу кількість доступних ресурсів та допомоги в Інтернеті;
- обмежена графічна якість. Графічні можливості Godot можуть бути меншими порівняно з деякими іншими рушіями, що може вплинути на створення великої кількості деталізованих графічних ефектів;
- не такий популярний серед великих студій. Godot не так часто використовується великими геймдев-студіями порівняно з іншими рушіями, хоча він стає все більш популярним.

Мінімальні системні вимоги для Godot Engine: операційна система Windows 7/8/10 64-біт; macOS 10.11+ або будь-яка сучасна дистрибуція Linux; процесор Dual-core 64-bit чи еквівалент; оперативна пам'ять 4 GB RAM; графічна карта, що підтримує OpenGL 3.0; простір на жорсткому диску 2 GB [2].

1.3 Аналіз інтегрованої розробницької середи Unity 3D

Unity 3D - це інтегрована розробницька середа та графічний рушій, розроблений компанією Unity Technologies (рис 1.3). Цей інструмент використовується для створення ігор та інтерактивного візуального контенту

на різних платформах. Unity 3D є відомим своєю гнучкістю, масштабованістю та зручним інтерфейсом для розробників.

Основні характеристики Unity 3D включають у себе графічний рушій, систему фізичного моделювання, засоби роботи з анімацією, різноманітними ефектами, партіклами, інтерфейсами, звуками, а також підтримку різних платформ, таких як ПК, консолі, мобільні пристрої та веб. Unity 3D є популярним серед розробників як для невеликих інді-проектів, так і для великих геймдев-студій, завдяки своїй універсальності, простоті та широкому функціоналу [3].

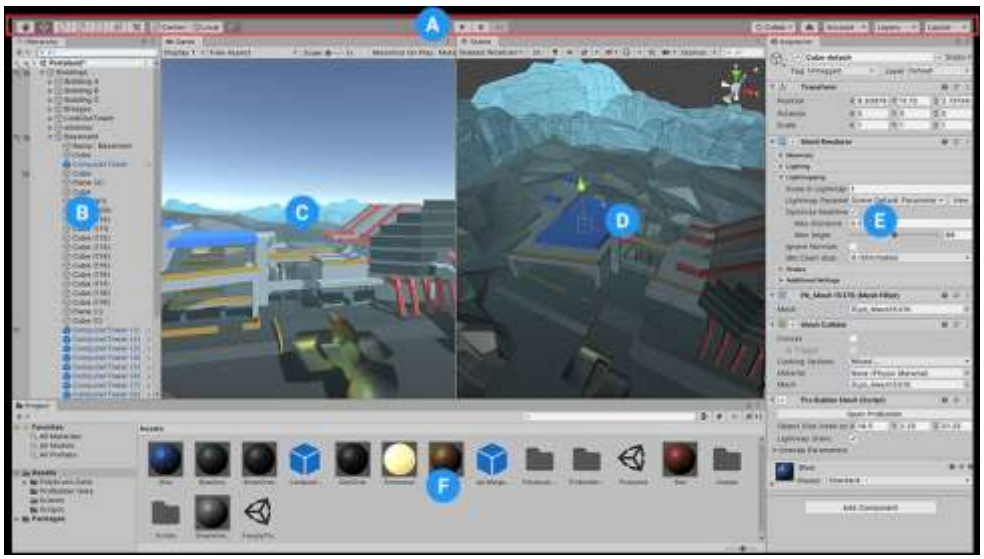


Рисунок 1.3 – Інтерфейс Unity 3D [3]

Переваги Unity 3D:

- широкий функціонал. Unity 3D має величезний набір інструментів для розробки ігор та інтерактивних застосунків, включаючи графіку, анімації, фізику, штучний інтелект та багато іншого;
- масштабованість. Від інді-розробок до великих проектів, Unity відмінно справляється з масштабуванням проектів різної складності;
- активна спільнота. Unity має велику та активну спільноту розробників, що сприяє обміну досвідом та розв'язанню проблем;
- широкий вибір платформ. Розроблені в Unity ігри можуть бути виведені на різні платформи, включаючи ПК, консолі, мобільні пристрої та веб.

- надійність та стабільність. Unity відомий своєю надійністю та стабільністю під час роботи над великими проектами;

Недоліки Unity 3D:

- ліцензійні витрати. Для великих комерційних проектів можуть виникнути значні ліцензійні витрати;
- закритий код. Деякі аспекти Unity можуть бути закритими для редагування, що обмежує повну свободу налаштувань для деяких розробників;
- високі вимоги до апаратного забезпечення. Для роботи з Unity ефективно потрібні потужні обчислювальні ресурси, що може бути обмежуючим для окремих користувачів;

Мінімальні системні вимоги для Unity 3D: операційна система Windows 7 SP1+, 8, 10, або macOS 10.12+; процесор Quad-core Intel or AMD, 2.5 GHz чи швидше; оперативна пам'ять 8 GB RAM; графічна карта з підтримкою DirectX 11 або Metal для macOS; є простір на жорсткому диску 20 GB [3].

1.4 Аналіз розробницької середи GameMaker

GameMaker - це інтегрована розробницька середа та рушій для створення 2D ігор (рис 1.4). Розроблений компанією YoYo Games, GameMaker дозволяє користувачам швидко і ефективно розробляти власні ігри без глибоких знань програмування [4].

Рисунок 1.4 - Інтерфейс GameMaker [4]

Переваги GameMaker:

- простий вивчення та використання. GameMaker славиться своєю легкістю вивчення, що робить його ідеальним для новачків у галузі геймдеву;
- ефективний для 2D ігор. Спрямований переважно на 2D ігри, GameMaker надає потужні інструменти для роботи з двомірною графікою та анімацією;
- швидкість розробки. Завдяки вбудованим функціям та готовим шаблонам, GameMaker дозволяє прискорити процес розробки ігор;
- спрощений процес програмування. Інтерфейс GameMaker дозволяє створювати логіку гри за допомогою блок-схем та обробників подій, що полегшує програмування;

Недоліки GameMaker 4D:

- обмежені можливості для 3D. GameMaker не є ідеальним вибором для розробки складних тривимірних ігор;

- менша гнучкість для великих проектів. Хоча GameMaker чудовий для невеликих та середніх проектів, він може стати менш практичним для великих та складних ігор;
- залежність від мови GML. Використання мови програмування GML (GameMaker Language) може бути обмежуючим для деяких розробників, зокрема для тих, хто звик до інших мов програмування;
- менше спільнота порівняно з іншими рушіями. GameMaker має меншу активну спільноту порівняно з іншими ігровими рушіями, що може вплинути на доступність ресурсів та підтримки в Інтернеті;

Мінімальні системні вимоги для GameMaker: операційна система Windows 7 SP1+, 8, 10, або macOS 10.11+; процесор Intel Dual Core 2GHz чи еквівалентний; оперативна пам'ять 2 GB RAM; графічна карта з підтримкою DirectX 11 або OpenGL 4.3; є простір на жорсткому диску 3 GB [4].

1.5 Аналіз пакету AppGameKit

AppGameKit - це пакет та рушій для створення ігор, розроблений компанією The Game Creators (рис. 1.5). Цей інструмент покликаний забезпечити легкість використання та швидкість розробки ігор, особливо для мобільних платформ [5].

Рисунок 1.5 – Інтерфейс AppGameKit [5]

Переваги AppGameKit:

- простий для вивчення та використання.AppGameKit спроектований з орієнтацією на простоту, що дозволяє навіть новачкам швидко вивчати його та розпочинати розробку;
- кросплатформенність. AppGameKit дозволяє створювати ігри, які можна легко експортувати на різні платформи, включаючи iOS, Android, Windows, macOS та інші;
- підтримка мови AGK Script. Мова AGK Script, яку використовує AppGameKit, спрощує процес програмування, зокрема для тих, хто не є досвідченим програмістом;
- швидкість розробки.AppGameKit надає готові рішення та шаблони, що допомагають прискорити розробку ігор;

Недоліки AppGameKit:

- обмежена графічна якість. У порівнянні з деякими іншими рушіями, AppGameKit може мати обмежені можливості для створення високоякісної графіки;
- відсутність великої спільноти. AppGameKit має меншу активну спільноту порівняно з іншими популярними ігровими рушіями, що може впливати на доступність ресурсів та підтримки;
- менше функціональності порівняно з конкурентами. Деякі інструменти та можливості, які пропонують конкуренти, можуть бути менше розвиненими або відсутніми в AppGameKit.
- обмежена підтримка сторонніх плагінів. У порівнянні з деякими конкурентами, AppGameKit може мати обмежену підтримку сторонніх плагінів та розширень;

Мінімальні системні вимоги для AppGameKit: операційна система Windows 7, 8, 10, або macOS 10.9+; процесор Intel Core 2 Duo чи еквівалентний;

оперативна пам'ять 2 GB RAM; графічна карта з підтримкою OpenGL 2.0 або вище; є простір на жорсткому диску 700 MB [5].

1.6 Аналіз ігрового рушія CryEngine

CryEngine - це високопродуктивний графічний рушій та інтегрована розробницька середа, розроблена компанією Crytek (рис. 1.6). Спочатку розроблений для гри "Far Cry," рушій отримав широке застосування в інших відомих іграх, таких як "Crysis" і "Star Citizen."[6].

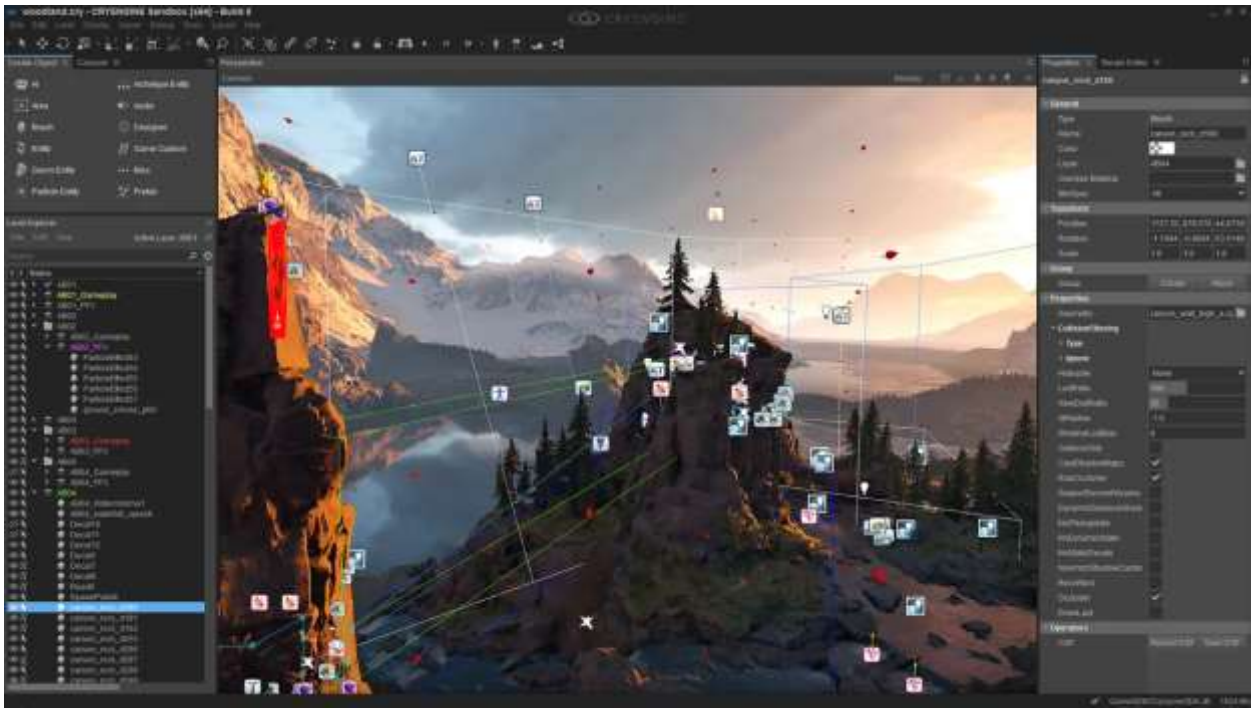


Рисунок 1.6 – Інтерфейс CryEngine [6]

Переваги CryEngine:

- графічна потужність. CryEngine славиться своєю вражаючою графікою та потужними графічними можливостями, що робить його відмінним вибором для створення високоякісних графічно продвинутих ігор;
- фізичний рушій. CryEngine має вбудований фізичний рушій, який надає реалістичну поведінку об'єктів, води та інших елементів у грі;

- широкі можливості. Рушій надає велику кількість інструментів для роботи з штучним інтелектом, анімацією, звуком та іншими аспектами гри;
- кросплатформенність. Хоча підтримка кросплатформенності була покращена, CryEngine залишається дещо менш універсальним порівняно з іншими рушіями;

Недоліки CryEngine:

- складність вивчення. CryEngine може бути складним для вивчення, особливо для новачків у галузі геймдеву, через свою розширену функціональність та складний інтерфейс;
- високі вимоги до обладнання. Для роботи з CryEngine ефективно потрібні потужні обчислювальні ресурси, що може становити виклик для менших розробників або проектів;
- ліцензійні обмеження. Використання CryEngine може пов'язуватися з високими ліцензійними обмеженнями, особливо для комерційних проектів;

Мінімальні системні вимоги для CryEngine: операційна система Windows 7 SP1 (64-bit) або вище; процесор Intel Dual-Core 2.5 GHz або еквівалентний; оперативна пам'ять 8 GB RAM; графічна карта: NVIDIA GeForce GTX 670 / AMD Radeon HD 7870 або еквівалент, з DX11 підтримкою та 2 GB VRAM; є простір на жорсткому диску 8 GB; DirectX Версія 11 [6].

1.7 Аналіз програмного пакету Amazon Lumberyard

Amazon Lumberyard - це відкритий ігровий рушій, розроблений компанією Amazon та базується на технологіях CryEngine (рис. 1.7). Він призначений для розробки ігор і включає в себе широкий спектр інструментів та послуг від Amazon Web Services (AWS) [7].

Рисунок 1.7 – Інтерфейс Amazon Lumberyard [7]

Переваги Amazon Lumberyard:

- інтеграція з AWS. Lumberyard використовується з інфраструктурою AWS, надаючи розробникам доступ до обчислювальних та облікових ресурсів Amazon;
- графічна якість. Оснований на технологіях CryEngine, Lumberyard забезпечує високу графічну якість та фотореалістичні можливості;
- інтеграція з Twitch. Lumberyard має вбудовану підтримку для інтеграції з платформою стрімінгу Twitch, що робить його привабливим для розробників, спрямованих на ігри зі стрімінгом;
- надійність Amazon. Використання послуг AWS в Lumberyard дозволяє розробникам користуватися надійністю та масштабованістю Amazon в хмарному середовищі;

Недоліки Amazon Lumberyard:

- складність вивчення. Як із CryEngine, Lumberyard може бути складним для вивчення через розширену функціональність та великий обсяг інструментів;

– обмежена спільнота. Lumberyard має меншу активну спільноту порівняно з іншими ігровими рушіями, що може впливати на доступність ресурсів та підтримки;
– обмежені платформи. Хоча Lumberyard підтримує кілька платформ, він може бути менш універсальним порівняно з деякими конкурентами;

Мінімальні системні вимоги для Amazon Lumberyard: операційна система Windows 10 (64-bit); процесор Intel Core i5-6600K або AMD Ryzen 5 1600X; оперативна пам'ять 8 GB RAM; графічна карта: NVIDIA GeForce GTX 1060 (3GB VRAM) або AMD Radeon RX 570; є простір на жорсткому диску 230 GB; DirectX Версія 11 [7].

1.8 Аналіз ігрового рушія RPG Maker

RPG Maker (рис 1.8) - це серія програм для створення рольових ігор (RPG). Ці інструменти дозволяють користувачам створювати свої власні гри з мінімальними навичками програмування, використовуючи вбудовані ресурси та інтерфейс [8].

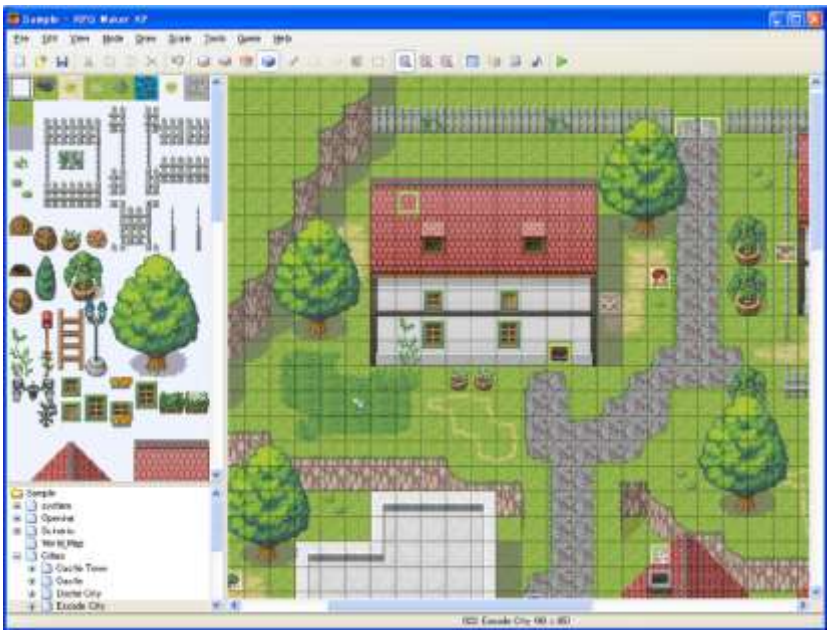


Рисунок 1.8 – Інтерфейс RPG Maker [8]

Переваги RPG Maker:

- простота використання. RPG Maker славиться своєю легкістю вивчення та використання, що робить його ідеальним для початківців у галузі геймдеву;
- швидкість розробки. Завдяки готовим шаблонам та вбудованим ресурсам, RPG Maker дозволяє швидко створювати прості RPG-ігри;
- комунітет та обмін ресурсами. Існує активний комунітет користувачів RPG Maker, який дозволяє обмінюватися ресурсами, ідеями та допомагати один одному у розробці гри;
- можливість власних інструментів. Інші версії RPG Maker дозволяють розробникам використовувати власні ресурси та скрипти для створення унікальних гравецьких досвідів;

Недоліки RPG Maker:

- обмежена гнучкість. RPG Maker, в першу чергу, орієнтований на створення певного типу ігор (RPG) і може бути обмеженим для інших жанрів;
- графічна якість. Хоча RPG Maker надає вбудовані графічні ресурси, графічна якість може бути меншою порівняно з більш потужними рушіями;
- неіндівідуалізований стиль. У зв'язку з використанням стандартних ресурсів, гри, створені за допомогою RPG Maker, можуть мають подібний стиль та вигляд;
- обмежені можливості програмування. Для більш досвідчених розробників, обмежені можливості програмування можуть бути обмеженням у реалізації складних ідей та функціоналу;

Мінімальні системні вимоги для RPG Maker: операційна система Windows 7/8/8.1/10 (32 або 64-біт); процесор Intel Core 2 Duo або краще; оперативна пам'ять 2 GB RAM; графічна карта: DirectX 9/OpenGL 4.1 сумісна

з NVIDIA GeForce GTX 500 Series/AMD Radeon HD 6000 Series (1 GB VRAM); є простір на жорсткому диску 700 MB; DirectX Версія 9.0c [8].

1.9 Аналіз LibGDX

LibGDX (рис. 1.9) - це відкрите програмне забезпечення для розробки кросплатформених мобільних та стільникових ігор. Він надає розробникам інструменти для створення гри на різних платформах, таких як Android, iOS, Desktop і веб [9].

Переваги LibGDX:

- кросплатформенність. LibGDX підтримує кросплатформенність, дозволяючи розробникам створювати ігри, які можна виводити на різних пристроях;
- відкрите програмне забезпечення. Бути відкритим проектом, LibGDX дозволяє розробникам вільно користуватися та модифікувати його з врахуванням власних потреб;
- широкий функціонал. LibGDX має величезний функціонал, включаючи роботу з графікою, фізикою, анімацією, звуком та іншими аспектами геймдеву;
- активна спільнота. За рахунок свого відкритого характеру LibGDX має активну спільноту розробників, яка активно допомагає та обмінюється ідеями;

Недоліки LibGDX:

- високий поріг входу. Для новачків може бути складним розпочати роботу з LibGDX через велику кількість функцій та налаштувань;
- не такий простий, як інші фреймворки. У порівнянні з деякими іншими інструментами, LibGDX може вимагати більше часу та зусиль для вивчення та використання;

– менше графічних ресурсів. Менша кількість вбудованих графічних ресурсів порівняно з іншими рушіями;

Мінімальні системні вимоги для LibGDX не застосовуються, оскільки LibGDX є бібліотекою розробки ігор для Java, а не окремим ігровим двигуном з власними системними вимогами. Однак для розробки на LibGDX потрібно мати встановлене програмне забезпечення для роботи з мовою програмування Java, таке як JDK (Java Development Kit), а також обране інтегроване середовище розробки (IDE), наприклад, Eclipse чи IntelliJ IDEA [9].

1.10 Висновки до розділу

Аналітичний огляд різних ігрових рушіїв, показав: які з них можуть бути використані для створення ігрового контенту. Кожен з рушіїв володіє унікальними характеристиками та можливостями для розробки віртуальних світів. Від індустріальних стандартів, таких як Unreal Engine, до популярних та доступних рішень, як Godot, ці програми пропонують різноманітні інструменти, придатні для різних сфер творчості та професійної діяльності.

Аналізуючи існуючі засоби для створення 3D об’єктів, в даній роботі будуть використовуватись дві програми: Unity 3D та JetBrains Rider.

Unity 3D є визнаним лідером у галузі розробки відмінних ігрових додатків. Його потужний інструментарій дозволяє створювати якісні та креативні 3D ігри для різних платформ, включаючи мобільні, настільні, та веб-платформи. Unity 3D володіє широким спектром можливостей у сфері анімації, фізики, штучного інтелекту та інших аспектів геймдеву. Безкоштовний доступ та велика активна спільнота додають йому конкурентну перевагу.

JetBrains Rider є інтегрованою середою розробки, спеціально призначеною для підтримки роботи з .NET-проектами, включаючи розробку ігор в Unity. Його продвинуті можливості редагування коду, рефакторінг та

підтримка роботи з Unity API роблять його відмінним інструментом для програмістів, що працюють над проектами у галузі геймдеву.

Unity 3D та JetBrains Rider взаємодіють між собою ефективно, надаючи команді розробників потужні та гнучкі засоби для творчого процесу. Їхні спільні можливості дозволяють створювати якісні та інноваційні 3D ігри, а також ефективно керувати та вдосконалювати кодову базу проекту.

Додатково, для ефективного керування версіями коду та спільної роботи над проектом використовуватиметься система керування версіями Git. Git надає зручні засоби для відстеження змін, створення гілок розробки, а також об'єднання різних гілок. Використання Git у поєднанні з Unity 3D та JetBrains Rider дозволяє зберігати та контролювати історію змін в коді, полегшує роботу з командою розробників та сприяє стабільності та надійності проекту. Такий підхід дозволяє зберігати інтегровану та організовану систему розробки, а також спрощує виявлення та вирішення конфліктів у випадку одночасної роботи над кодом кількох розробників.

2 РОЗРОБКА КОНЦЕПТ ДОКУМЕНТУ ІГРОВОГО ДОДАТКУ

2.1 Концептуалізація та визначення вимог

Основний жанр гри, це гіперкежуал сімулятор ферми, де гравець перевтілюється в медвідя, який готується до сплячки та будує свою власну ферму. Атмосфера гри спрощена та мила, спрямована на створення приємного враження, ідеально підходить для обхвату великої аудиторії.

Цілі гри - збирати ресурси, будувати виробничі будівлі та наймати інших тварин для допомоги. Ваш медвідь веде активний спосіб життя, підготовлюючись до зимового сну та створюючи процвітаючу ферму.

У грі ми створили неперевершений світ, де медвідь готується до сплячки, а гравець керує його життям на фермі. Головні герої - медвідь і його асистенти - привертають увагу та створюють унікальний геймплей.

Було вивчено різноманітні концепції та створено мокапи, щоб передати ідею гри. Кольорова палітра та дизайн інтерфейсу роблять гру привабливою для гравців.

Гра спрямована на аудиторію усіх вікових груп. Як гіперказуальний сімулятор, вона призначена для людей, які шукають простий та веселий спосіб провести час. Завдяки своєму привабливому сюжету та простим геймплейним механікам, гра сподобається широкому діапазону гравців.

Було вивчено сучасні тенденції, які були застосовані у грі. Здатність конкурувати на ринку визначена не тільки унікальністю, а й передовим підходом до розробки.

2.2 Проектування та створення архітектури

Створення програмної архітектури гри. Було проведено розробку структури та логічної архітектури гри в рамках середовища Unity 3D. Створена

дуже гнучка архітектура яку дуже легко розширювати і додавати новий функціонал. Визначено чіткі взаємозв'язки між об'єктами та компонентами, забезпечуючи оптимальну роботу гри.

Проектування Геймплейних Механік. Проведено визначення основних геймплейних механік та взаємодії гравця з оточенням. Розроблено ігрові сценарії та розгорнуті можливі варіанти розвитку подій для створення захоплюючого геймплею.

Дизайн Інтерфейсу. Здійснено розробку дизайну інтерфейсу, охоплюючи вигляд кнопок, меню, підказок та інших елементів. Реалізовано анімації та переходи між екранами для забезпечення інтуїтивного та привабливого користувацького досвіду. Інтерфейс виконано в мінімалістичному стилі для запобігання візуального перегруженя кількістю ігрових елементів які можуть одночасно знаходитись на екрані.

Визначення Логіки. Розроблено логічне ядро гри, включаючи обробку подій, систему взаємодії та управління головними героями та об'єктами. Логіка була детально визначена для ефективного функціонування гри.

Основний жанр гри, це гіперкежуал сімулятор ферми, де гравець перевтілюється в ведмедя, який готується до сплячки та будує свою власну ферму. Атмосфера гри мила та проста, спрямована на створення приємного враження, ідеально підходить для обхвату великої аудиторії.

Цілі гри - збирати ресурси, будувати виробничі будівлі та наймати інших тварин для допомоги. Ваш медвідь веде активний спосіб життя, підготовлюючись до зимового сну та створюючи процвітаючу ферму.

У грі ми створили неперевершений світ, де медвідь готується до сплячки, а гравець керує його життям на фермі. Головні герої - медвідь і (в майбутньому) його асистенти - привертають увагу та створюють унікальний геймплей.

Була додана можливість пересуватись по мапі, збирати ресурси, зберігати їх та витрачати. Також була реалізована можливість побудови нових будівель з можливістю для їх покращення.

Кожен з цих етапів важливий для створення повноцінного та ефективного дизайну ігрового додатку в середовищі Unity 3D.

2.3 Розробка

Програмування Головної Логіки. Реалізація основної логіки гри реалізована за допомогою мови програмування C#. Розроблено основні алгоритми для головного героя (медвідя) та інших ігрових об'єктів, включаючи систему збору ресурсів, будівництво та взаємодію з іншими персонажами та об'єктами. Була створена логіка, яка визначає основні моменти гри та управління подіями.

Створення Графічних Елементів. Були використані інструменти Unity 3D для додавання графічних елементі, таких як об'єкти, персонажі та фони, і їхнього розташування на сцені. Дизайн графічних об'єктів відповідає концепції гри, де важливий акцент приділяється простоті та милому вигляду.

Робота із Звуком. Були вивчені та імплементовані звукові ефекти, які підсилюють атмосферу гри. Використовувалися звукові ресурси для створення реалістичної звукової картини, включаючи звуки дій гравця та оточення.

Імплементація Ігрових Механік. Імплементовано та протестовано різноманітні геймплейні механіки, які були визначені на етапі проектування. Проводилося тестування нових функцій, і вирішення проблем, що можуть виникнути під час імплементації.

Тестування та Відлагодження. Було проведено систематичне тестування для перевірки правильності функціонування всіх головних елементів гри. Виявлені помилки та недоліки були виправлені, а код був оптимізований для підвищення продуктивності.

Імплементація Штучного Інтелекту. Імплементовано систему штучного інтелекту для неперсонажних об'єктів, яка забезпечує реалістичну поведінку

взаємодії з навколишнім світом. Проведено тестування та оптимізацію штучного інтелекту для забезпечення ефективної та реалістичної гри.

Інтеграція Зовнішніх Сервісів. Було вивчено та інтегровано зовнішні сервіси для поліпшення функціональності гри та забезпечення взаємодії з користувацькими сервісами.

Забезпечено взаємодію гри з зовнішніми платформами для поліпшення загального досвіду користувачів.

Створення Механізмів Монетизації. Імплементовано механізми монетизації, такі як внутрішня валюта, платні елементи або рекламні блоки, якщо вони передбачені для гри. Забезпечено баланс між геймплейними можливостями та варіантами монетизації.

Підготовка до Тестування Користувачами. Впевнено було перевірено готовність всіх аспектів гри перед передачею на етап тестування користувачами.

Переконаємося, що всі елементи правильно функціонують та відповідають специфікаціям проекту.

2.4 Тестування

Підготовка Тестових Сценаріїв. Було розроблено комплекс тестових сценаріїв для всебічної валідації різноманітних аспектів гри. Це включало визначення ключових функціональностей, які вимагали особливої уваги та перевірки.

Функціональне Тестування. Було проведено ретельне функціональне тестування з метою перевірки відповідності реалізованих функцій вимогам. Детально складено та задокументовано виявлені дефекти чи невідповідності для подальшого виправлення.

Тестування Інтерфейсу та Взаємодії. Проведено валідацію коректності та зручності використання інтерфейсу гри, а також тестування взаємодії користувача з ігровим середовищем та іншими елементами.

Тестування Завантаження та Продуктивності. Було визначено швидкість завантаження та продуктивності гри на різних пристроях, а також вирішено можливі проблеми, пов'язані із завантаженням та продуктивністю.

Тестування Стабільності. Проведено тестування на стабільність гри з метою виявлення та виправлення можливих зависань чи збоїв. Тестування гри на різних пристроях та операційних системах.

Тестування Штучного Інтелекту. Проведено перевірку роботи штучного інтелекту в різних умовах та сценаріях гри, включаючи виявлення та виправлення аномалій чи неправильної поведінки ігрових об'єктів.

Тестування Сумісності. Визначено сумісність гри з різними версіями операційних систем та пристроями, а також проведено тестування на наявність конфліктів або проблем із сумісністю.

Підготовка Звіту та Виправлення Помилок. Складено докладний звіт про результати тестування, включаючи опис виявлених помилок. Перевірено та виправлено виявлені дефекти перед переходом до наступного етапу розробки.

2.5 Випуск та розгортання

Підготовка гри для релізу. Було завершено всі останні зміни та виправлення, враховані під час етапу тестування. Здійснена перевірка повноти та актуальності документації гри, забезпечивши, що всі важливі аспекти документації оновлені та відповідають останнім змінам у грі.

Створення Релізної Версії. Здійснено компіляцію остаточної версії гри, яка готова для публікації. Забезпечена перевірка того, щоб всі ресурси та файли були правильно включені у вихідний пакет гри, готовий для розповсюдження.

Вибір Платформ для Розгортання. Були визначені цільові платформи, на яких гра буде доступна. Це мобільні пристрої на базі Android та IOS. Здійснена підготовка специфічних ресурсів для кожної платформи, зокрема оптимізація гри для різних пристроїв.

Подача заявок на платформах розповсюдження. Здійснено подачу необхідних документів та заявок для публікації гри на платформах розповсюдження, таких як App Store иа Google Play. Забезпечено відповідність усіх вимог та процедур обраного середовища розповсюдження

Підтримка та оновлення. Забезпечено технічну підтримку для користувачів та вирішенню проблеми, що можуть виникнути після випуску гри. Після випуску гри є можливість покращувати та оновлювати продукт.

2.6 Висновки до розділу

У цьому розділі був проведений комплексний аналіз процесів та середовища, необхідних для успішної створення ігрового додатку. Розглянуті етапи розробки включають концептуалізацію та визначення вимог, проектування, розробку, тестування, випуск та розгортання.

На етапі визначення вимог, було визначено основні принципи та напрямки розробки, зокрема аналізувалися ідеї гри та визначалися ключові вимоги до продукту. Ретельний аналіз дозволив визначити цільову аудиторію та виробити чітке бачення проекту.

Була розроблена архітектура гри, визначено геймплейні механіки та розроблено інтерфейс. Цей етап визначає основні принципи функціонування гри та створює основну концепцію продукту.

На етапі розробки використовувалися інструменти, такі як Unity 3D та JetBrains Rider. Unity 3D, завдяки своїм можливостям, став основним інструментом для створення 3D ігрового контенту, тоді як JetBrains Rider забезпечив потужний інтерфейс для розробки та оптимізації коду.

На етапі тестування проводилася перевірка правильності функціонування всіх аспектів гри. Виявлені помилки та недоліки були виправлені, що сприяло поліпшенню якості та надійності продукту.

Було завершено всі останні зміни та виправлення після етапу тестування, підготовлено остаточну версію гри для публікації. Обрані платформи для розгортання визначено відповідно до цільової аудиторії.

Загалом, використання інтегрованого підходу з використанням Unity 3D, JetBrains Rider та системи керування версіями Git дозволило забезпечити ефективну розробку, тестування та випуск ігрового додатку. Проект орієнтований на якісний інтерактивний вміст, а вибір інструментів підтримує комфортну та продуктивну роботу розробників.

3 РОЗРОБКА АЛГОРИТМУ ІГРОВОГО КОНТЕНТУ

3.1 Створення базової архітектури проекту

У даному розділі розглянуто алгоритм створення архітектури проекту, а саме: GameStateMachine, ініціалізація залежностей та стани програми.

Розробимо точку входу з якої починається виконання програми, але в Unity, через компонентний підхід, за завомчуванням такий функціонал не передбачен. Щоб мати можливість керувати станом програми був створений GameBootstrapper (рис 3.1). Це компонент який буде на ігровій сцені завжди, від початку і до самого її завершення (рис 3.2).

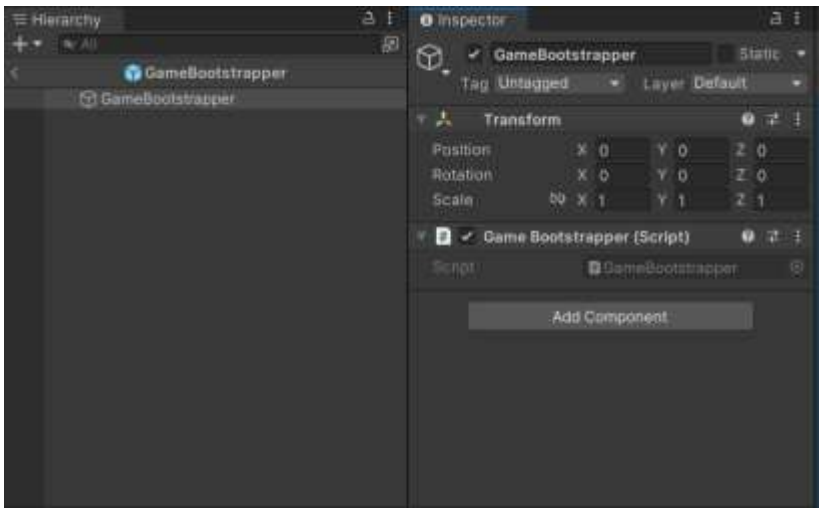


Рисунок 3.1 – Точка входу в гру

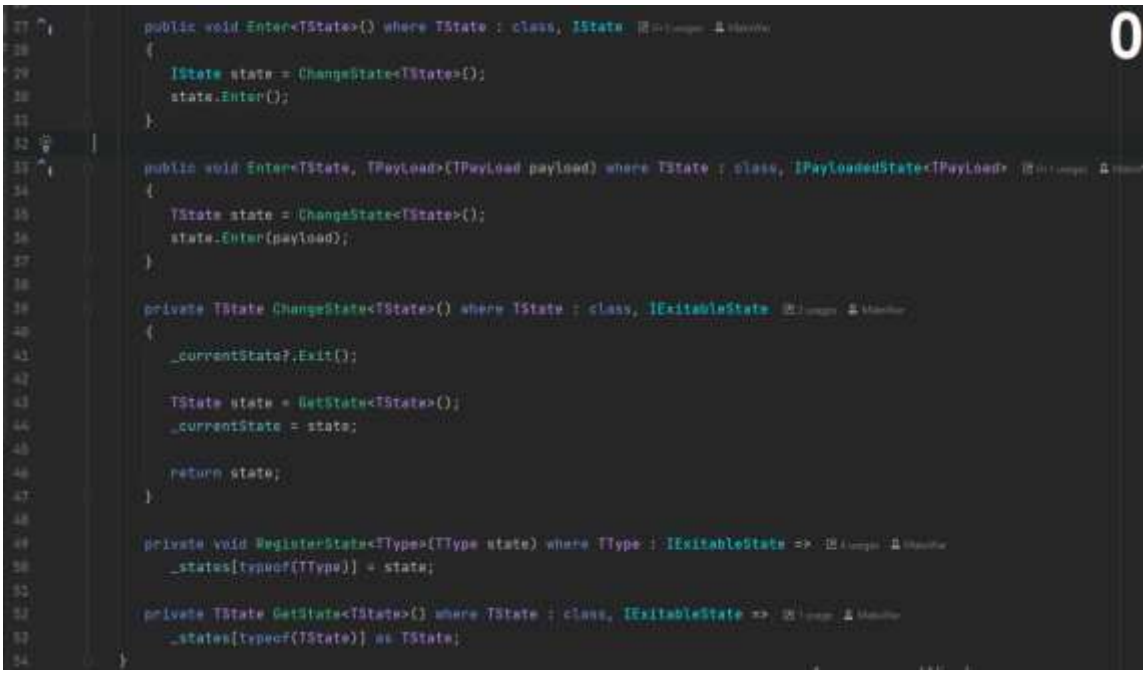
```
using Additions.Extensions;
using Infrastructure.States;
using Zenject;

namespace Infrastructure
{
    public class GameBootstrapper : MonoSingleton<GameBootstrapper>, ICoroutineRunner  3 usages
    {
        private IGameStateMachine _stateMachine;

        [Inject]
        private void Construct(IGameStateMachine stateMachine) =>
            _stateMachine = stateMachine;

        private void Start()
        {
            _stateMachine.Enter<BootstrapState>();

            DontDestroyOnLoad(this);
        }
    }
}
```

Рисунок 3.2 – Код точки входу

GameStateMachine (рис 3.3, рис. 3.4) дозволяє зручно керувати усіма станами в якому може бути наша гра, будь то стан ініціалізації, стан загрузки рівня, стан підвантаження прогресу гравця чи стан ігрового лупу.

```
namespace Infrastructure.States
{
    public interface IGameStateMachine
    {
        void Enter<TState>() where TState : class, IState;

        void Enter<TState, TPayload>(TPayload payload) where TState : class, IPayloadedState<TPayload>;
    }
}
```

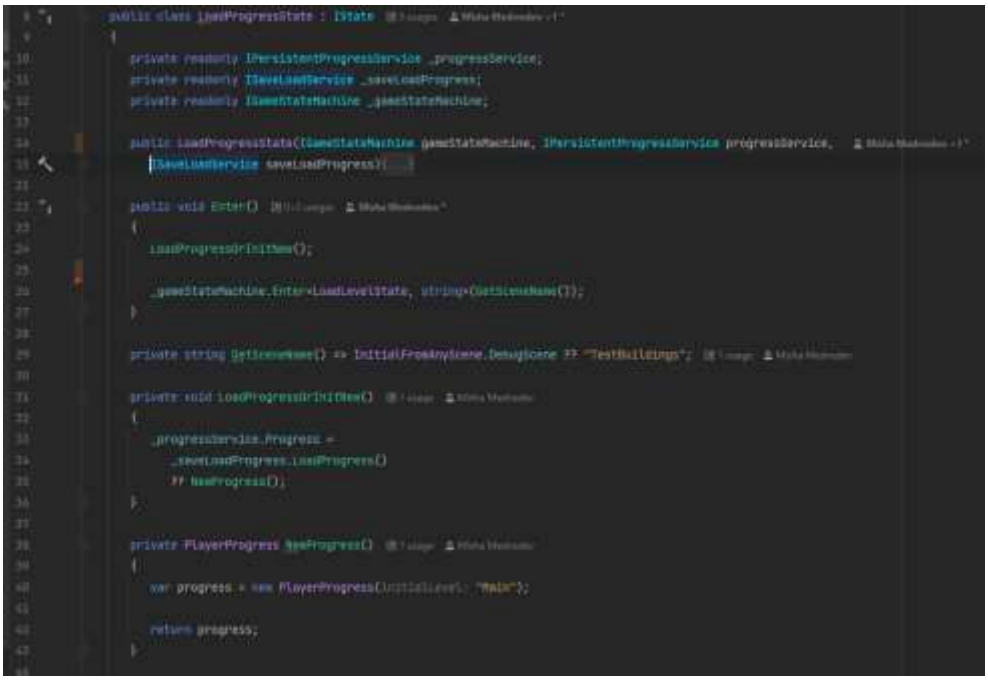
Рисунок 3.3- Інтерфейс IGameStateMachine



```
public void Enter<TState>() where TState : class, IState
{
    IState state = ChangeState<TState>();
    state.Enter();
}

public void Enter<TState, TPayLoad>(TPayLoad payload) where TState : class, IPayloadedState<TPayLoad>
{
    TState state = ChangeState<TState>();
    state.Enter(payload);
}

private TState ChangeState<TState>() where TState : class, IExitableState
{
    _currentState?.Exit();

    TState state = GetState<TState>();
    _currentState = state;

    return state;
}

private void RegisterState<TType>(TType state) where TType : IExitableState =>
    _states[typeof(TType)] = state;

private TState GetState<TState>() where TState : class, IExitableState =>
    _states[typeof(TState)] as TState;
}
```

Рисунок 3.4 – Реалізація методів інтерфейсу IGameStateMachine

LoadProgressState - це стан в якому наша гра підвантажує прогрес гравця із пам’яті, або створює новий, якщо такий не було знайдено (рис 3.5).



```
public class LoadProgressState : IState
{
    private readonly IPersistentProgressService _progressService;
    private readonly ISaveLoadService _saveLoadProgress;
    private readonly IGameStateMachine _gameStateMachine;

    public LoadProgressState(IGameStateMachine gameStateMachine, IPersistentProgressService progressService,
        ISaveLoadService saveLoadProgress)

    public void Enter()
    {
        LoadProgressOrInitNew();

        _gameStateMachine.Enter<LoadLevelState, string>(GetSceneName());
    }

    private string GetSceneName() => InitialFromAnyScene.DebugScene ?? "TestBuildings";

    private void LoadProgressOrInitNew()
    {
        _progressService.Progress =
            _saveLoadProgress.LoadProgress()
            ?? NewProgress();
    }

    private PlayerProgress NewProgress()
    {
        var progress = new PlayerProgress("Main");

        return progress;
    }
```

Рисунок 3.5 – Класс LoadProgressState

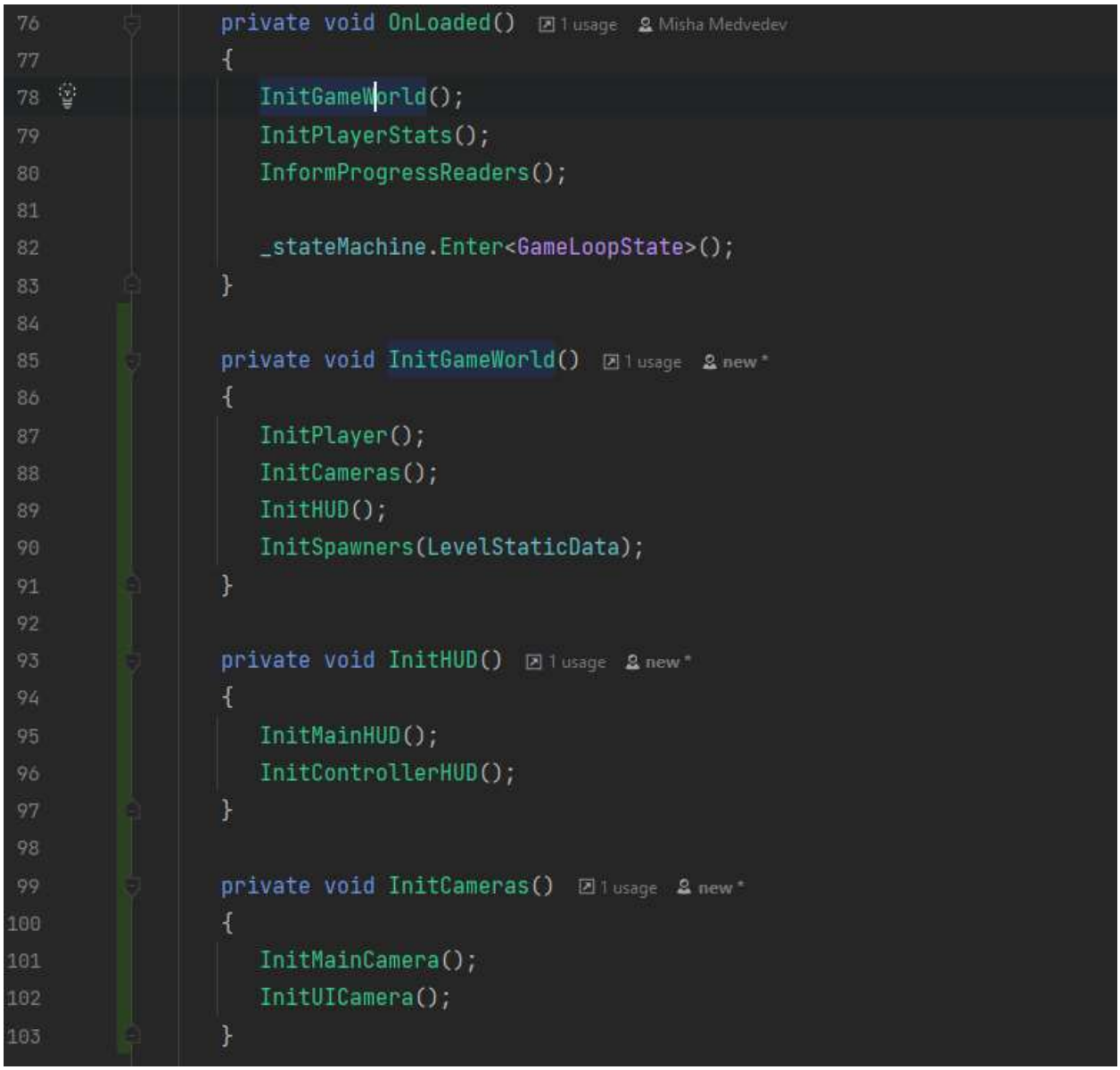
LoadLevelState – це основний стан який буде викликатись кожен раз коли треба зробити перехід на іншу сцену, це може бути сцена с налаштуваннями, інший рівень, або нова локація. Залежності класа зображені на (рис 3.6). При заході в цей стан (рис 3.7), гравець бачить екран завантаження, починається загрузка всього рівня і після того як рівень завантажився, закривається екран загрузки. Повну реалізацію в коді можна побачити на (рис 3.8). Структурна блоксхема зображена на (рис 3.9).

```
19  public class LoadLevelState : IPayloadedState<string>  5 usages  Misha Medvedev +1
20  {
21      private GameObject _player;
22
23      private readonly IPersistentProgressService _progressService;
24      private readonly IBuildingFactory _buildingFactory;
25      private readonly IFactoryProvider _factoryProvider;
26      private readonly IGameStateMachine _stateMachine;
27      private readonly IStaticDataService _staticData;
28      private readonly IResourcesFactory _resourcesFactory;
29      private readonly IUnitsFactory _unitsFactory;
30      private readonly ISceneLoader _sceneLoader;
31      private readonly IUIFactory _uiFactory;
32      private readonly CharacterConfigs _characterConfigs;
33      private readonly HarvestableConfig _harvestableConfig;
34      private readonly LoadingCurtain _loadingCurtain;
```

Рисунок 3.6 – Залежності стану LoadLevelState

```
65      public void Enter(string sceneName)  0+1 usages  Misha Medvedev +1
66      {
67          _loadingCurtain.Show();
68          _factoryProvider.Cleanup();
69
70          _sceneLoader.Load(sceneName, OnLoaded);
71      }
72
73      public void Exit() =>  0+2 usages  MaksWar
74          _loadingCurtain.Hide();
75
```

Рисунок 3.7 – Реалізація методів Enter та Exit



```
76      private void OnLoaded()  1 usage  Misha Medvedev
77      {
78          InitGameWorld();
79          InitPlayerStats();
80          InformProgressReaders();
81
82          _stateMachine.Enter<GameLoopState>();
83      }
84
85      private void InitGameWorld()  1 usage  new *
86      {
87          InitPlayer();
88          InitCameras();
89          InitHUD();
90          InitSpawners(LevelStaticData);
91      }
92
93      private void InitHUD()  1 usage  new *
94      {
95          InitMainHUD();
96          InitControllerHUD();
97      }
98
99      private void InitCameras()  1 usage  new *
100     {
101         InitMainCamera();
102         InitUICamera();
103     }
```

Рисунок 3.8 – Ініціалізація ігрового середовища

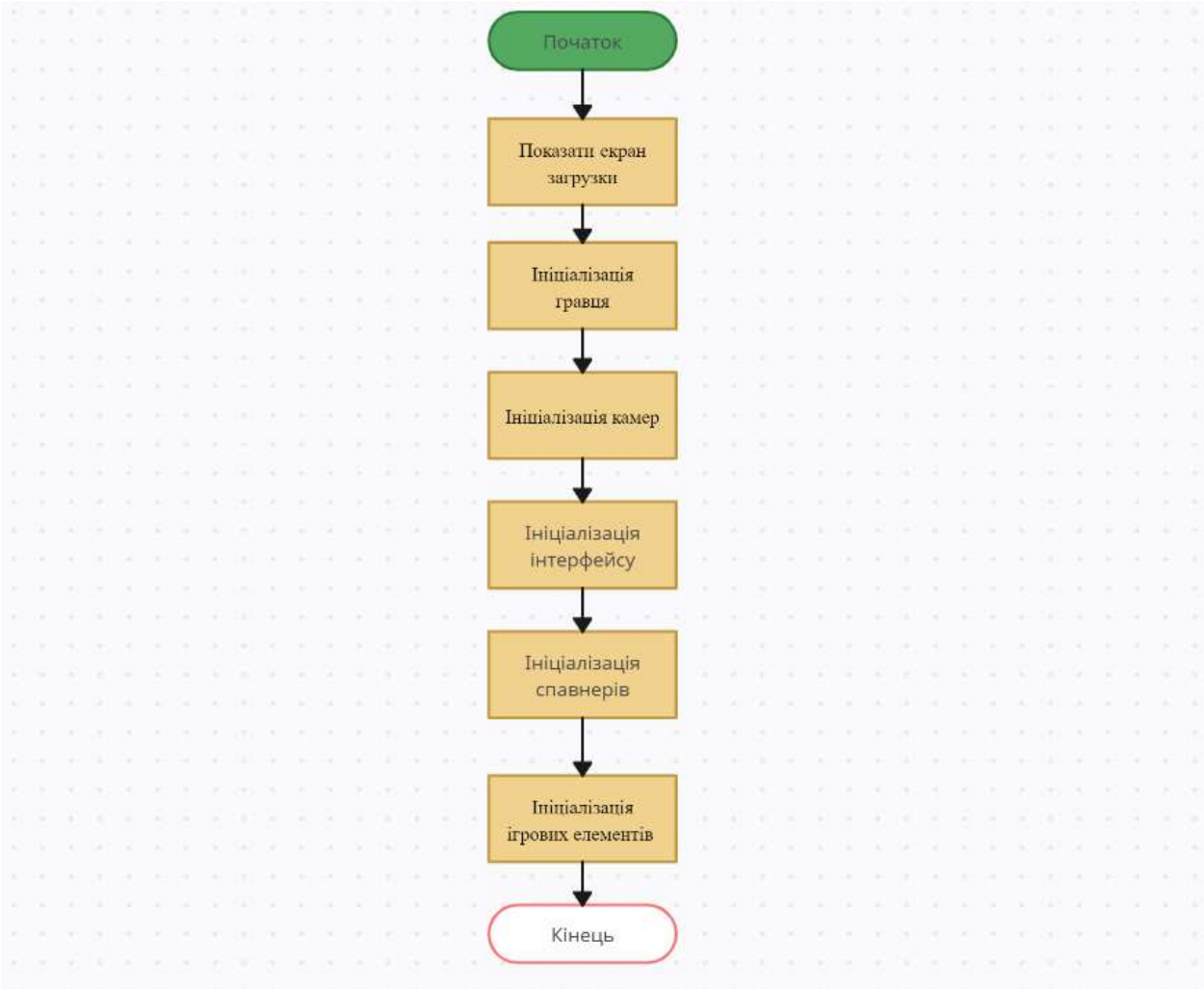
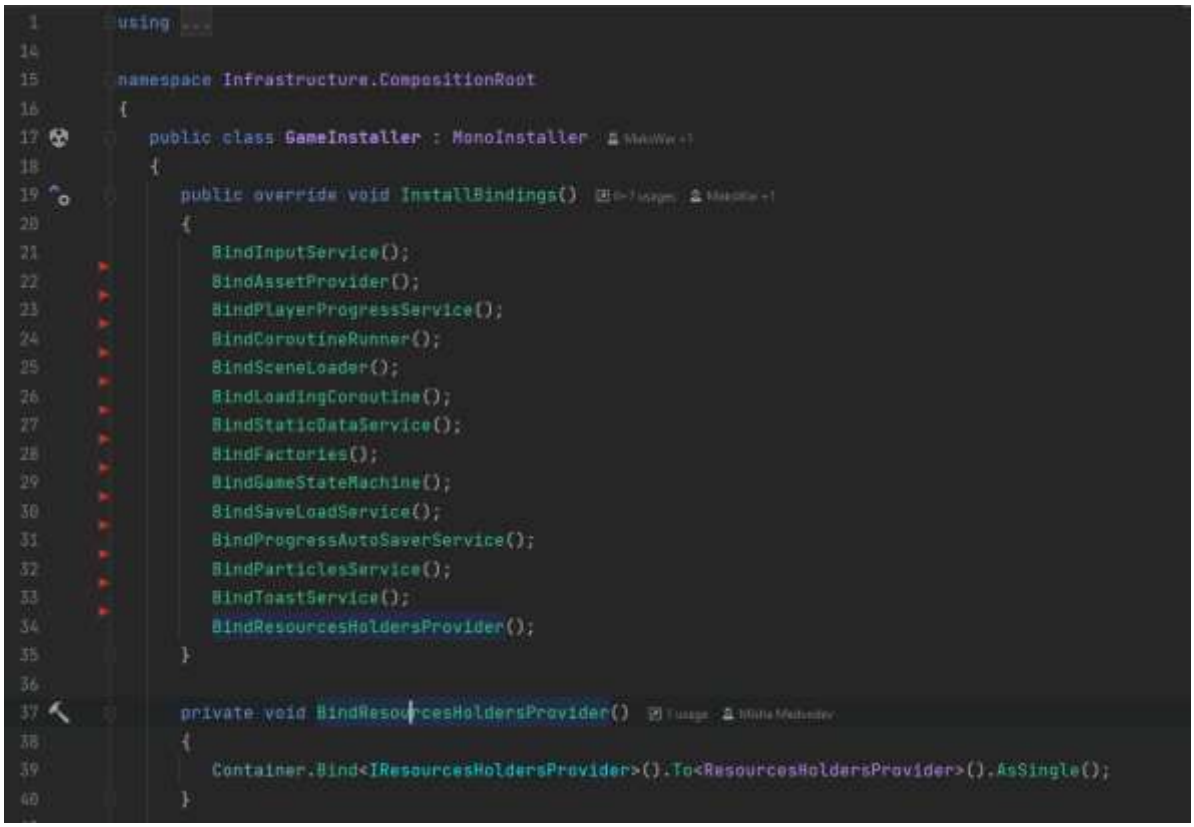


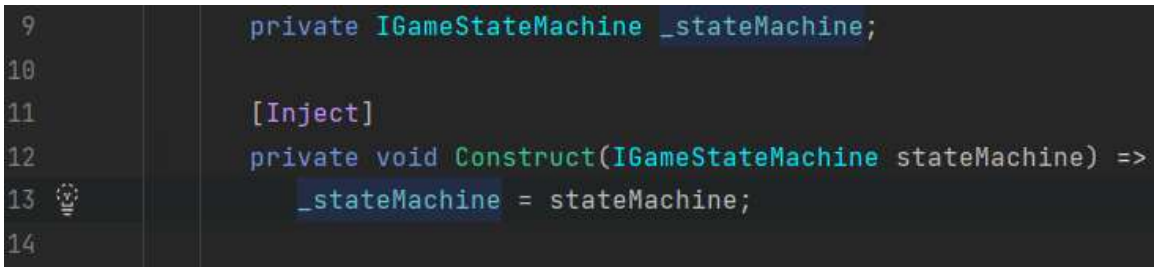
Рисунок 3.9 – Структурна блок-схема виконання LoadLevelState

Для доступу к сервісам гри використовується контейнер впровадження залежностей (рис 3.10). Процес використання контейнера впровадження залежностей робить доступ до функціоналу гри більш гнучким і легким для управління, що сприяє зручності розробки та підтримки програмного продукту. Завдяки цьому підходу розробники можуть швидко та ефективно отримувати доступ до потрібних сервісів (рис 3.11), що полегшує впровадження нових функцій та модифікацію існуючих елементів гри.



```
using ...

namespace Infrastructure.CompositionRoot
{
    public class GameInstaller : MonoInstaller
    {
        public override void InstallBindings()
        {
            BindInputService();
            BindAssetProvider();
            BindPlayerProgressService();
            BindCoroutineRunner();
            BindSceneLoader();
            BindLoadingCoroutine();
            BindStaticDataService();
            BindFactories();
            BindGameStateMachine();
            BindSaveLoadService();
            BindProgressAutoSaverService();
            BindParticlesService();
            BindToastService();
            BindResourcesHoldersProvider();
        }

        private void BindResourcesHoldersProvider()
        {
            Container.Bind<IResourcesHoldersProvider>().To<ResourcesHoldersProvider>().AsSingle();
        }
```

Рисунок 3.10 – Ініціалізація залежностей на сервіси



```
    private IGameStateMachine _stateMachine;

    [Inject]
    private void Construct(IGameStateMachine stateMachine) =>
        _stateMachine = stateMachine;
```

Рисунок 3.11 – Приклад отримання залежності через атрибут Inject

3.2 Створення персонажу

У даному розділі розглянуто алгоритм створення головного героя, компонентний підхід написання ігрової поведінки, створення компонентів та додаванні ігрових об’єктів на сцену.

Одна із кращих систем для написання гри є компонентна система. Сенс такої побудови програми полягає в тому, що функціональність гри

розбивається на окремі компоненти або модулі, де кожен модуль відповідає за конкретний аспект гри. Основними перевагами такої системи є: гнучкість, можливість повторного використання коду, легкість тестування, модульність та простота у створенні нових компонентів.

Для прикладу розглянемо компонент який зображено на (рис 3.12). Це компонент який сам по собі нічого не робить, але в ньому закладена логіка для пересування. Цей компонент може бути використан для пересування не тільки гравцем, а і ворогів зі штучним інтелектом без потрібності повторного написання цієї логіки.

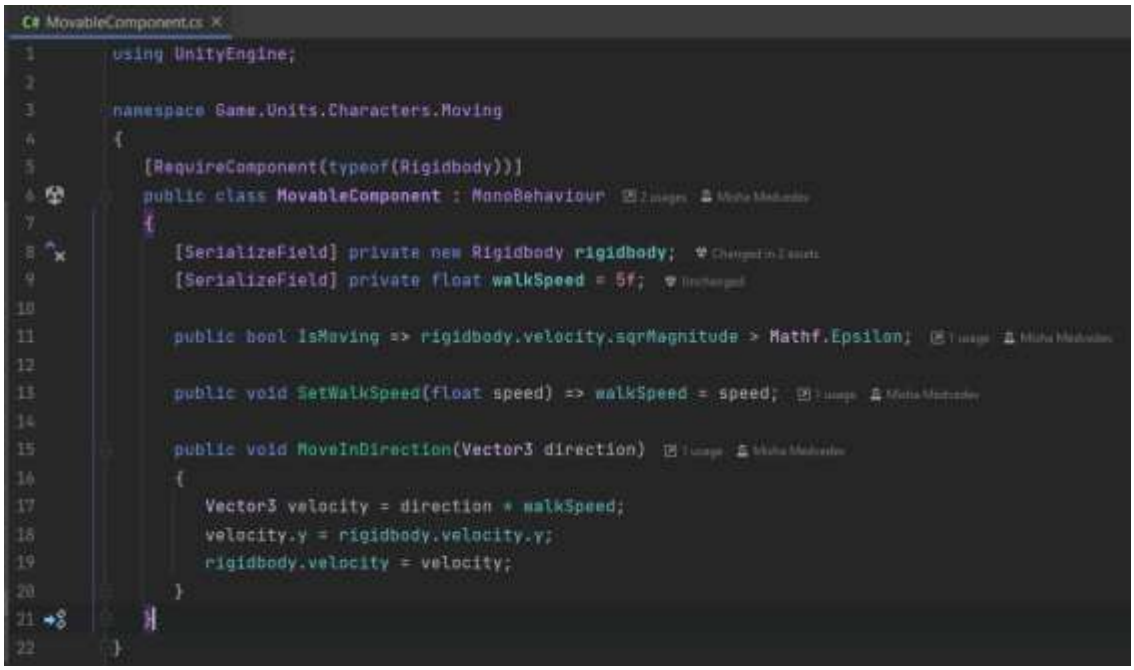


Рисунок 3.12 – Компонент який відповідає за пересування

Для пересування говловного героя було написано компонент PlayerMovement, який отримує для себе компоненти Movable та Rotateable для пересування та поворотів відповідно. Також за допомогою впровадження залежності був отриман InputService який повертає нам команди які вводить гравець, для подальшої їх обробки. В залежності від системи на якої рухається наша гра, ми ініціалізуємо різну реалізацію цього сервіса. Тобто на телефонах

сервіс буде реагувати на тачі по екрану (рис 3.15), на десктопі на ввід з клавіатури (рис 3.16), а на ігрових консолях це будуть відгуки від геймпаду.

```
public class PlayerMovement : MonoBehaviour, ISavedProgressReader
{
    [SerializeField] private MovableComponent movableComponent;
    [SerializeField] private RotatableComponent rotatableComponent;

    private IInputService _inputService;
    private Camera _camera;

    [Inject]
    private void Construct(IInputService inputService) =>
        _inputService = inputService;

    private void Start() =>
        _camera = Camera.main;

    private void Update()
    {
        Vector2 axis = _inputService.Axis;
        Vector3 transformDirection = _camera.transform.TransformDirection((Vector3)axis);
        transformDirection.y = 0;
        transformDirection.Normalize();

        movableComponent.MoveInDirection(transformDirection);

        if (axis.sqrMagnitude > Mathf.Epsilon)
        {
            rotatableComponent.RotateTo(transformDirection);
        }
    }
```

Рисунок 3.13 – Пересування гравця

```
using UnityEngine;

namespace Game.Units.Characters.Moving
{
    public class RotatableComponent : MonoBehaviour
    {
        [SerializeField] private new Rigidbody rigidbody;
        [SerializeField] private float rotationSpeed = 20f;

        public void RotateTo(Vector3 direction)
        {
            Vector3 currentRotation = transform.forward;
            Vector3 smoothedRotationDirection =
                Vector3.Lerp(currentRotation, direction, rotationSpeed * Time.deltaTime);
            rigidbody.MoveRotation(Quaternion.LookRotation(smoothedRotationDirection));
        }

        public void StopRotation()
        {
            rigidbody.angularVelocity = Vector3.zero;
        }
    }
}
```

Рисунок 3.14 – Компонент відповідаючий за повороти

```
using UnityEngine;

namespace Infrastructure.Services.Input
{
    public class MobileInputService : InputService
    {
        public override Vector2 Axis => SimpleInputAxis();
    }
}
```

Рисунок 3.15 –InputService у випадку мобільного додатку

```
using UnityEngine;

namespace Infrastructure.Services.Input
{
    public class StandaloneInputService : InputService
    {
        public override Vector2 Axis
        {
            get
            {
                var axis = SimpleInputAxis();

                if (axis == Vector2.zero)
                    axis = UnityAxis();

                return axis;
            }
        }

        private static Vector2 UnityAxis()
            => new Vector2(UnityEngine.Input.GetAxis(Horizontal), UnityEngine.Input.GetAxis(Vertical));
    }
}
```

Рисунок 3.16 –InputService у випадку десктопного додатку

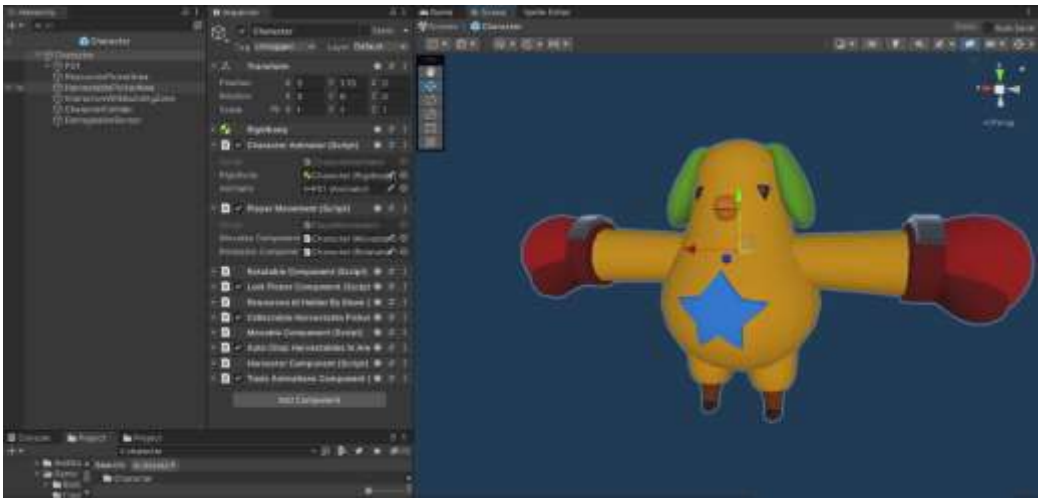


Рисунок 3.17 – Зовнішний вигляд гравця з компонентами

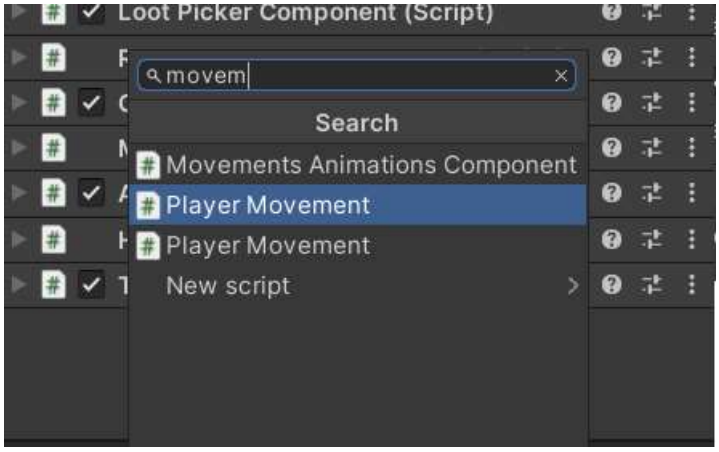


Рисунок 3.18 – Додавання нового компоненту

Рисунок 3.19 – Ігровий персонаж рухається відповідно від вводу

3.3 Збереження прогресу гравця

Майже кожна сучасна гра зберігає ігрові дані. Це може бути поточне місцезнаходження гравця, кількість видобитих ресурсів тощо.

У даному розділі розглянуто алгоритм збереження даних та їх структура.

Для зберігання даних була використана вбудована в Unity система PlayerPrefs. Це система для зберігання та отримання даних, яка дозволяє розробникам зберігати налаштування та інші дані гри між різними запусками або навіть між сценами гри. Вона дозволяє легко зберігати та отримувати значення ключ-значення, які використовуються для збереження різноманітної інформації.

Для цього був створений клас з поміткою Serializable (рис 3.20), що дозволяє забезпечити можливість серіалізації та десеріалізації об'єктів цього класу. Після визначення необхідних властивостей та методів, об'єкти цього класу можуть бути легко конвертовані в рядок, який потім може бути збережений у PlayerPrefs (рис 3.22).

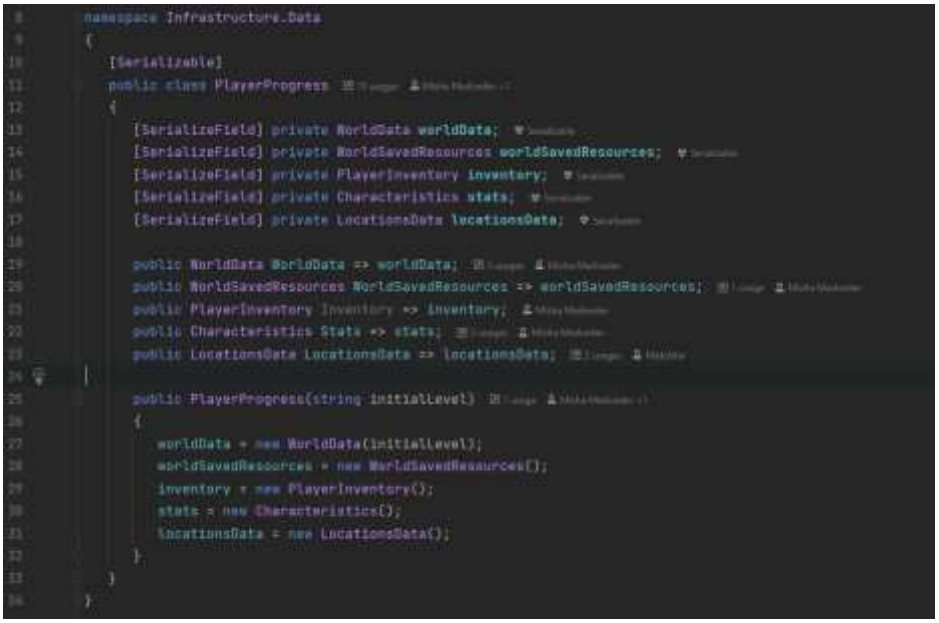
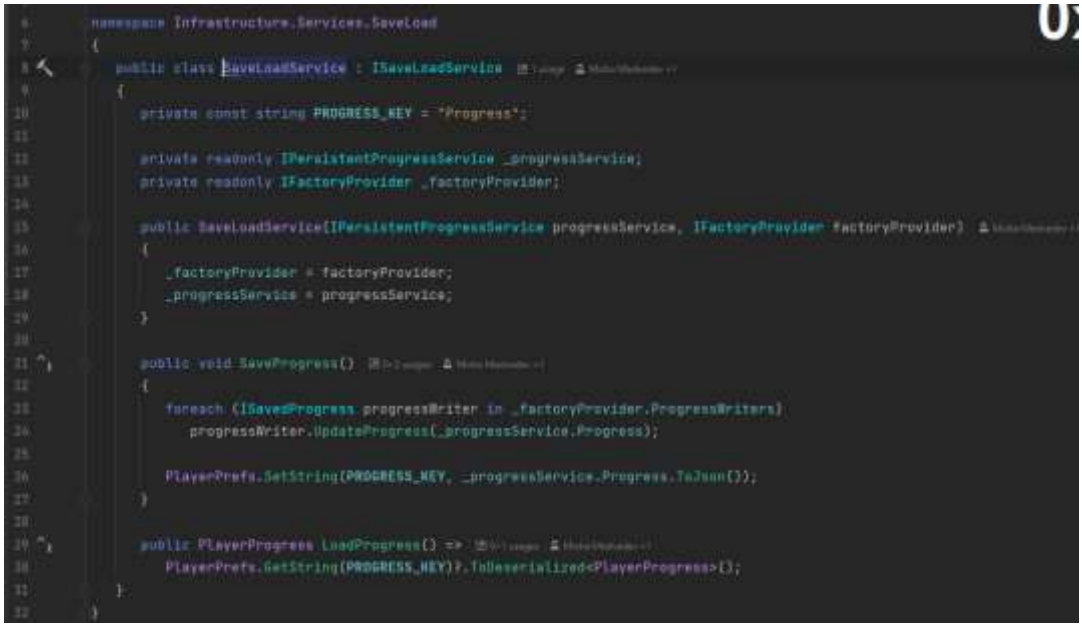


Рисунок 3.20 – Структура даних гравця

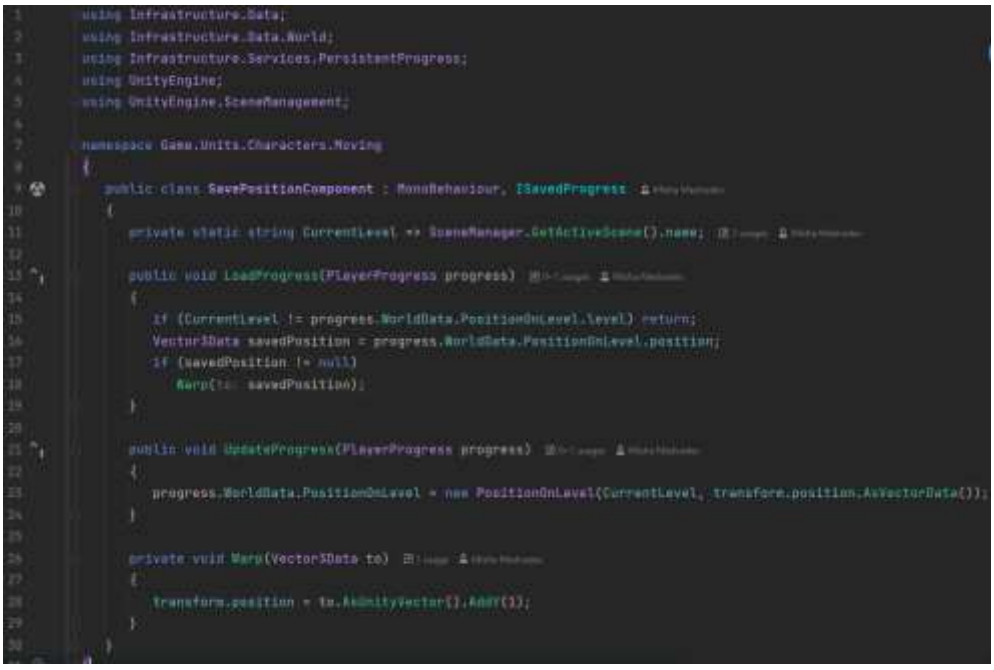
Рисунок 3.21 – Сервіс зберігаючий дані.



```
namespace Infrastructure.Services.SaveLoad
{
    public class SaveLoadService : ISaveLoadService
    {
        private const string PROGRESS_KEY = "Progress";

        private readonly IPersistentProgressService _progressService;
        private readonly IFactoryProvider _factoryProvider;

        public SaveLoadService(IPersistentProgressService progressService, IFactoryProvider factoryProvider)
        {
            _factoryProvider = factoryProvider;
            _progressService = progressService;
        }

        public void SaveProgress()
        {
            foreach (ISavedProgress progressWriter in _factoryProvider.ProgressWriters)
                progressWriter.UpdateProgress(_progressService.Progress);

            PlayerPrefs.SetString(PROGRESS_KEY, _progressService.Progress.ToJson());
        }

        public PlayerProgress LoadProgress() =>
            PlayerPrefs.GetString(PROGRESS_KEY)?.ToDeserialized<PlayerProgress>();
    }
}
```

Рисунок 3.22 – Реалізація інтерфейсу ISaveLoadService.

Для збереження і завантаження позиції гравця був написаний компонент з назвою яка відповідає його функціоналу, SavePositionComponent (рис 2.23). Як і усі інші компоненти, його можна використовувати не тільки для головного героя, але і для елементів оточення



```
using Infrastructure.Data;
using Infrastructure.Data.World;
using Infrastructure.Services.PersistentProgress;
using UnityEngine;
using UnityEngine.SceneManagement;

namespace Game.Units.Characters.Moving
{
    public class SavePositionComponent : MonoBehaviour, ISavedProgress
    {
        private static string CurrentLevel => SceneManager.GetActiveScene().name;

        public void LoadProgress(PlayerProgress progress)
        {
            if (CurrentLevel != progress.WorldData.PositionOnLevel.Level) return;
            Vector3Data savedPosition = progress.WorldData.PositionOnLevel.position;
            if (savedPosition != null)
                Warp(to: savedPosition);
        }

        public void UpdateProgress(PlayerProgress progress)
        {
            progress.WorldData.PositionOnLevel = new PositionOnLevel(CurrentLevel, transform.position.AsVectorData());
        }

        private void Warp(Vector3Data to)
        {
            transform.position = to.AsUnityVector().AddY(1);
        }
    }
}
```

Рисунок 3.23- Компонент зберігаючий позицію об’єкта на рівні

```
namespace Infrastructure.Data.World
{
    [Serializable]
    public class PositionOnLevel
    {
        public string level;
        public Vector3Data position;

        public PositionOnLevel(string level, Vector3Data position)
        {
            this.level = level;
            this.position = position;
        }

        public PositionOnLevel(string initialLevel)
        {
            level = initialLevel;
        }
    }
}
```

Рисунок 3.24 – Формат в якому зберігається позиція об’єкта

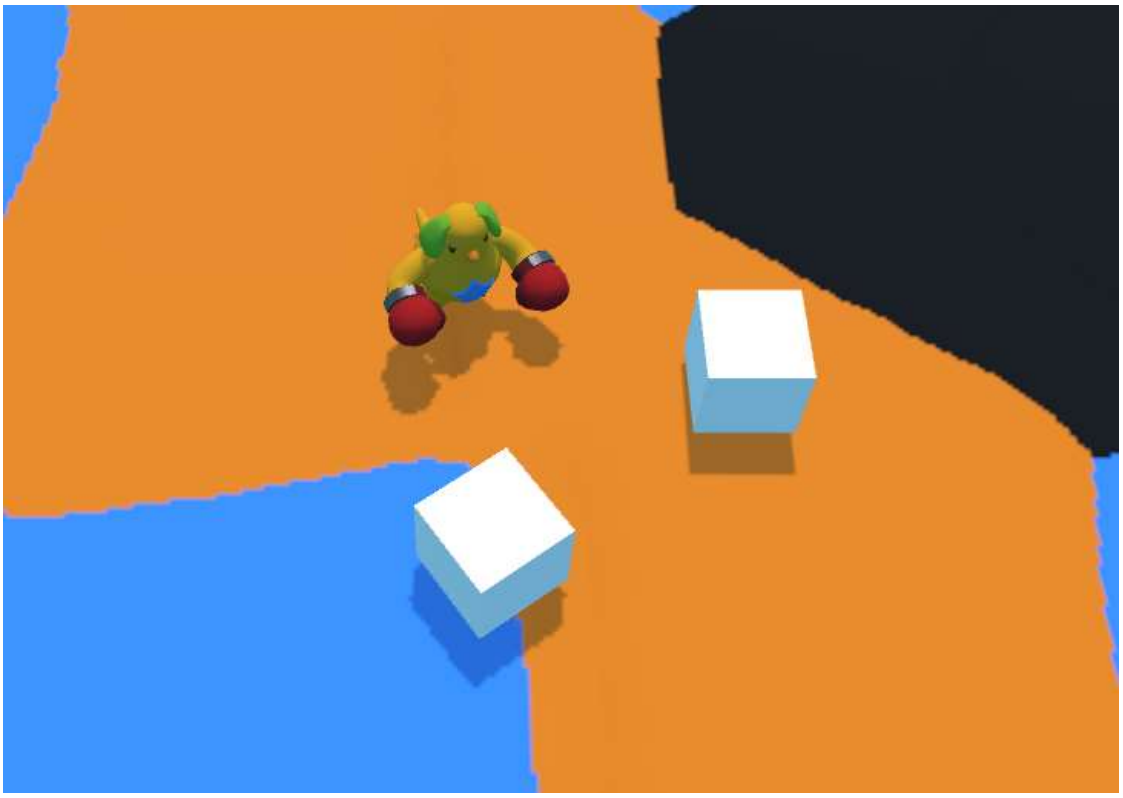


Рисунок 3.25 – Позиція зберігається навіть після перезаходу в гру.

3.4 Збереження налаштувань на сервері

Кожен розробник хоче мати можливість віддалено регулювати характеристики гравця та його оточення, для швидкого регулювання балансу без оновлення всієї гри, в разі необхідности.

У даному розділі розглянуто конфігі, таблиці з характеристиками та доступ до них.

Для збереження даних в едіторі Unity було використано ScriptableObject. Це клас в Unity, який дозволяє створювати об'єкти, які можуть бути збережені між сценами, геймплейними сеансами та навіть в редакторі. Використання ScriptableObject (рис 3.26) є ефективним способом управління конфігураційними даними, ресурсами та іншою інформацією в грі.

Для збереження даних на віддаленому сервері було використано сервіс Google Sheets (рис 3.27). Цей підхід надає можливість динамічно змінювати та оновлювати дані гри, зберігаючи їх у віддаленому електронному документі. Використання Google Sheets для збереження даних має декілька переваг: оновлення в реальному часі; можливість роботи в команді; гнучкість та зручність управління даними; зменшення необхідності в оновленнях програмного коду; можливість використання в інших проектах;.

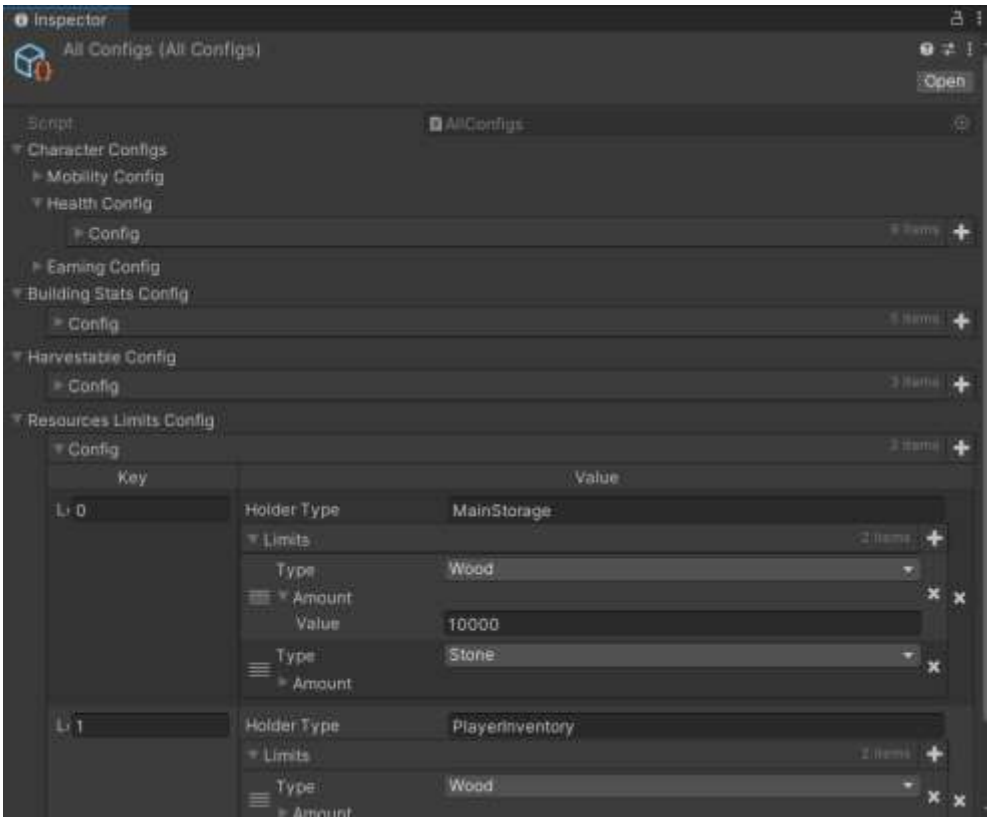


Рисунок 3.26 – Вигляд конфігу в едіторі Unity.

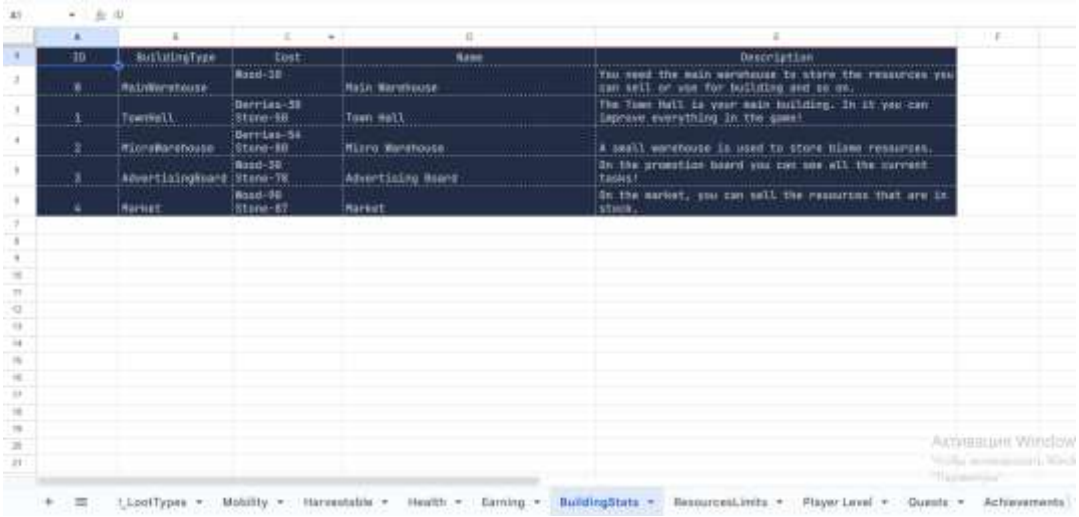


Рис 3.27 – Збереження конфігурації в Google Sheets

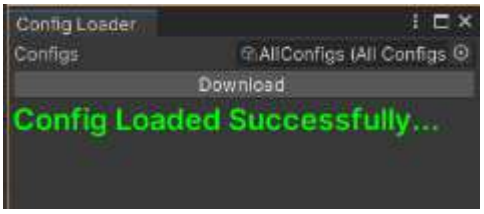
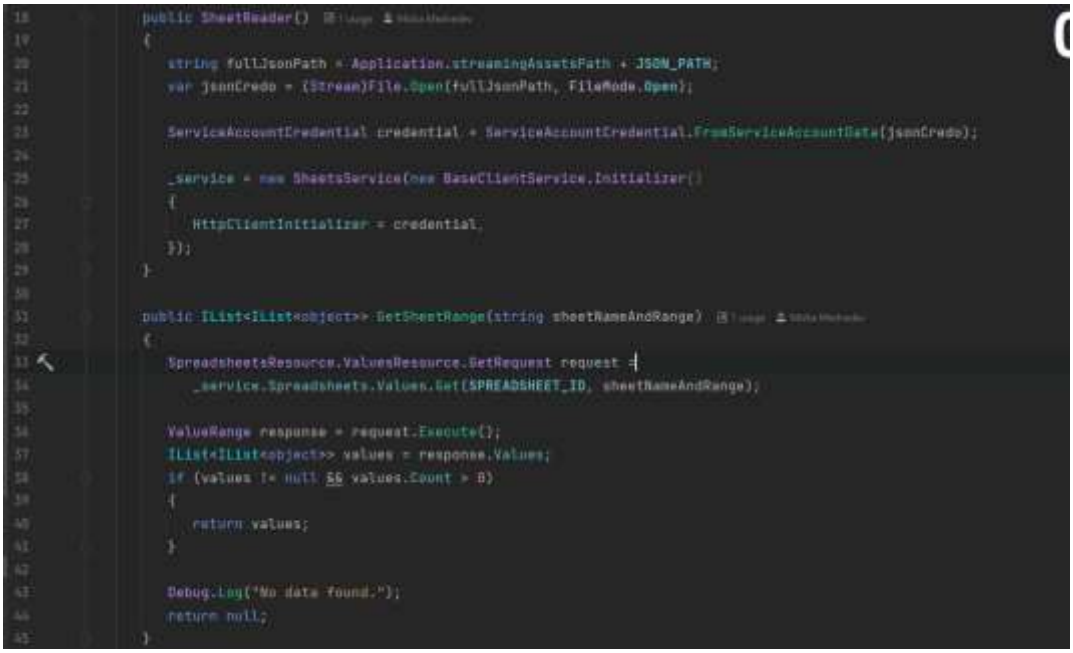
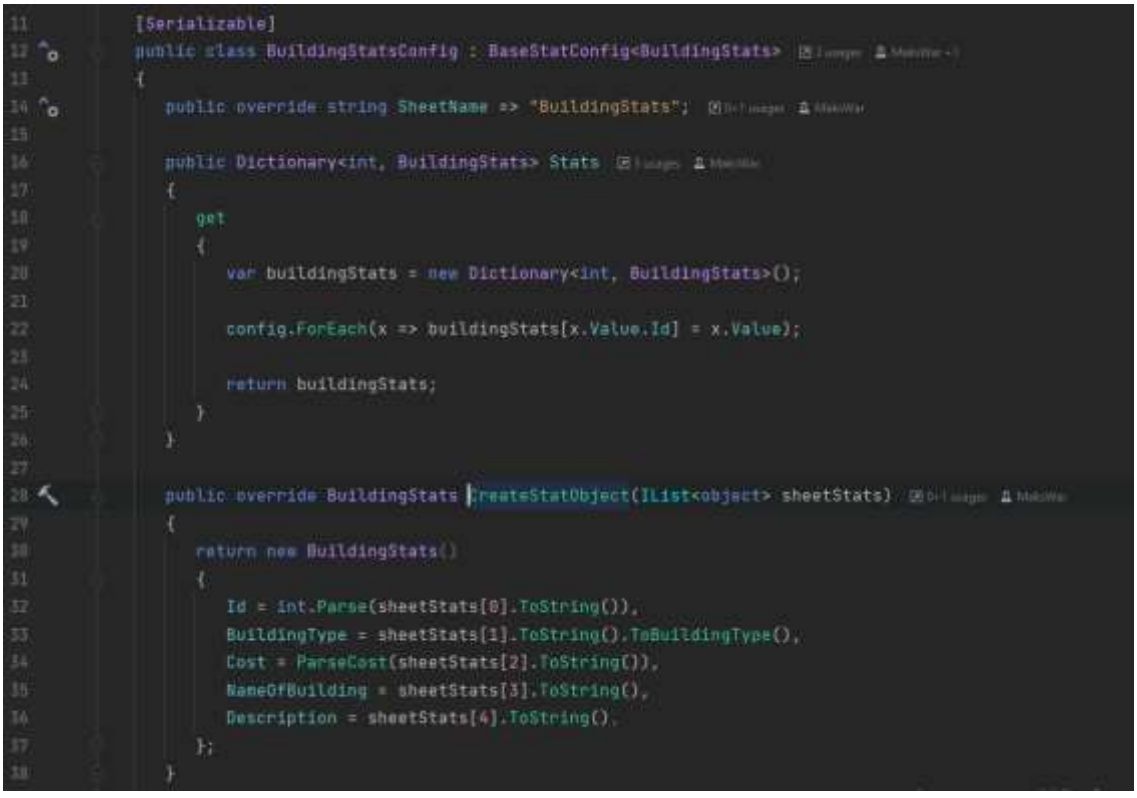


Рисунок 3.28 – Завантаження конфігурації з таблиці



```
public SheetReader()
{
    string fullJsonPath = Application.streamingAssetsPath + JSON_PATH;
    var jsonCredo = (Stream)File.Open(fullJsonPath, FileMode.Open);

    ServiceAccountCredential credential = ServiceAccountCredential.FromServiceAccountData(jsonCredo);

    _service = new SheetsService(new BaseClientService.Initializer()
    {
        HttpClientInitializer = credential,
    });
}

public IList<IList<object>> GetSheetRange(string sheetNameAndRange)
{
    SpreadsheetsResource.ValuesResource.GetRequest request =
        _service.Spreadsheets.Values.Get(SPREADSHEET_ID, sheetNameAndRange);

    ValueRange response = request.Execute();
    IList<IList<object>> values = response.Values;
    if (values != null && values.Count > 0)
    {
        return values;
    }

    Debug.Log("No data found.");
    return null;
}
```

Рисунок 3.29 – Код для зчитування даних з таблиці.



```
[Serializable]
public class BuildingStatsConfig : BaseStatConfig<BuildingStats>
{
    public override string SheetName => "BuildingStats";

    public Dictionary<int, BuildingStats> Stats
    {
        get
        {
            var buildingStats = new Dictionary<int, BuildingStats>();

            config.ForEach(x => buildingStats[x.Value.Id] = x.Value);

            return buildingStats;
        }
    }

    public override BuildingStats CreateStatObject(IList<object> sheetStats)
    {
        return new BuildingStats()
        {
            Id = int.Parse(sheetStats[0].ToString()),
            BuildingType = sheetStats[1].ToString().ToBuildingType(),
            Cost = ParseCost(sheetStats[2].ToString()),
            NameOfBuilding = sheetStats[3].ToString(),
            Description = sheetStats[4].ToString(),
        };
    }
```

Рисунок 3.30 – Приклад зчитування конфігурації для будівлі.

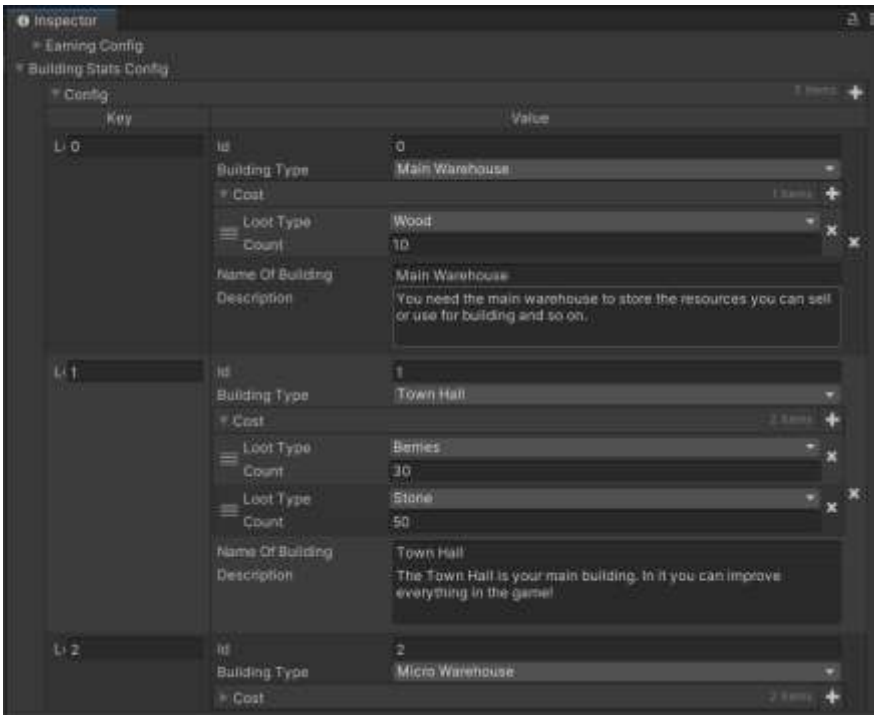


Рисунок 3.31 – Вигляд завантаженої конфігурації

3.5 Створення елементів оточення

У даному розділі розглянуто елементи оточення, та керуючі компоненти цих елементів.

Реалізовано геймплейний елемент - збираємий гриб (рис 3.32). Для його реалізації використовується компонент ініціалізації (рис 3.33), який дозволяє налаштовувати параметри об'єкта та визначати його властивості.

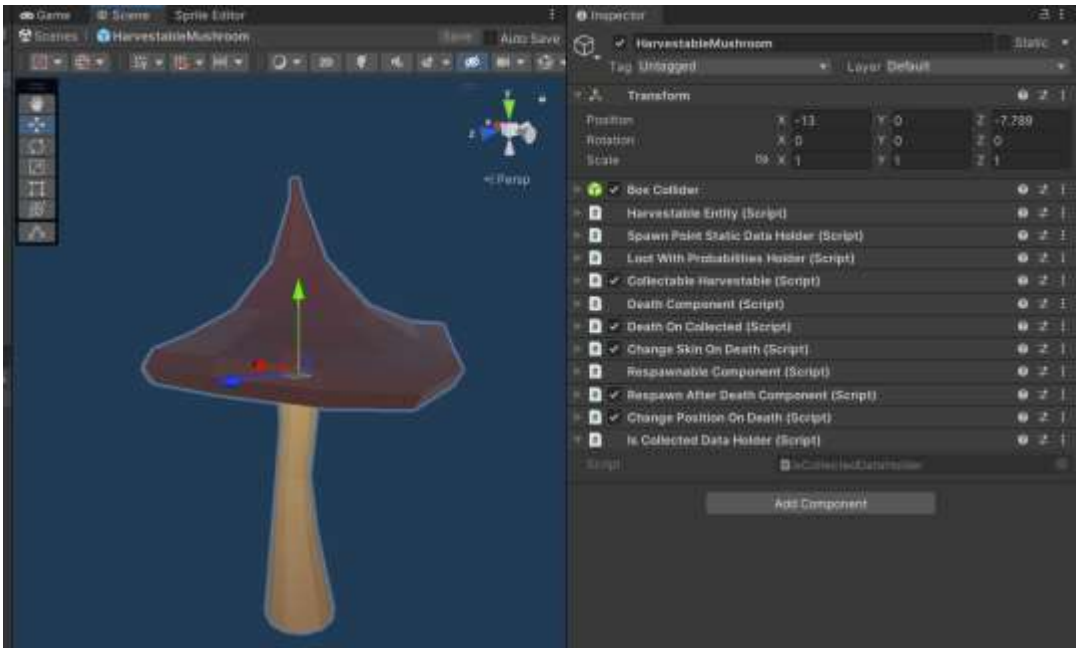


Рисунок 3.32 – Збираємий гриб

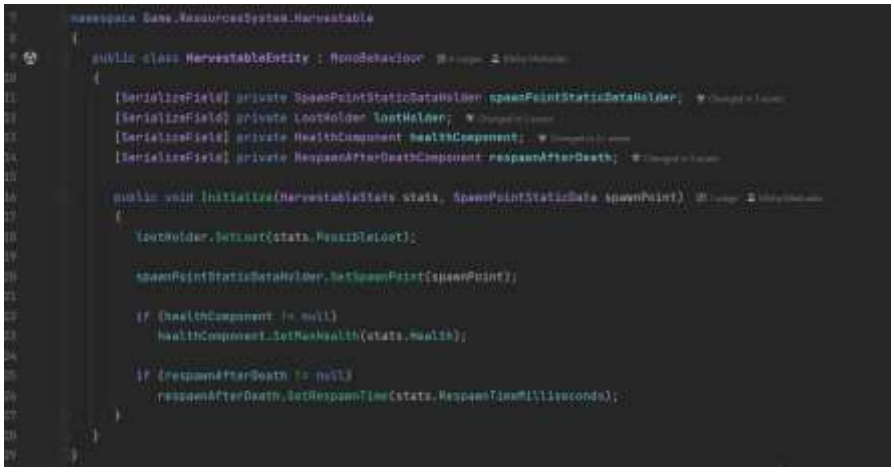


Рисунок 3.33 – Компонент ініціалізації для збираємих об’єктів

Щоб мати можливість знову взаємодіяти з об'єктом, коли він був зібраний або використаний, використовується компонент респавну. Цей компонент (рис 3.343) дозволяє об'єкту знову з'явитися в грі після певного часу чи події.

Рисунок 3.34 – Компонент для респавну

Для інформування системи про те, що об'єкт був використаний чи зібраний, використовується компонент звітуючий про смерть об'єкта. Цей компонент (рис 3.35) генерує подію або повідомлення при смерті об'єкта.

Рисунок 3.35 – Компонент звітуючий про смерть об’єкта.

Деякі збираємі об'єкти можуть мати свої механіки смерті. Компонент викликаючий смерть (рис 3.36) відповідає за ініціацію процесу "смерті" для цих об'єктів.

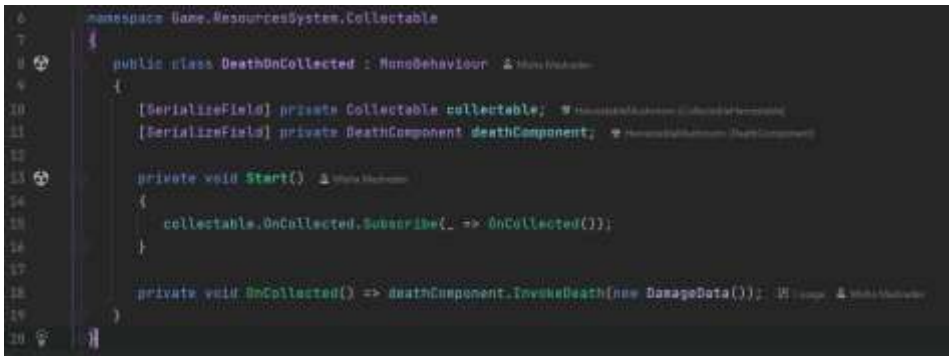


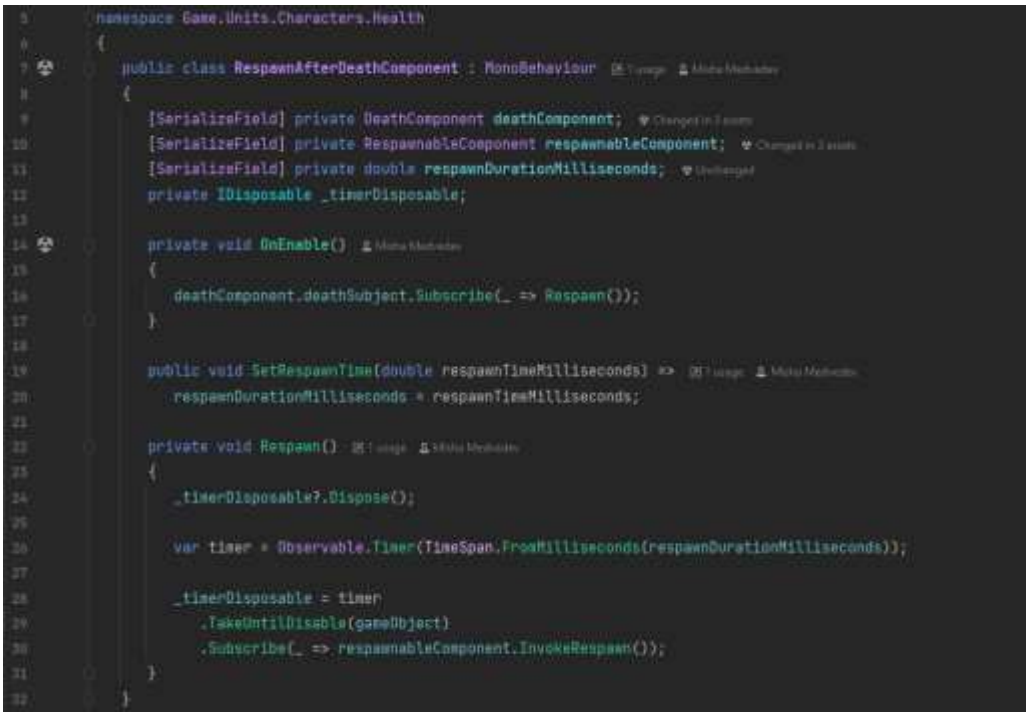
Рисунок 3.36 – Компонент викликаючий смерть для збираємих об’єктів

Деякі об'єкти можуть зазнавати змін у своєму зовнішньому вигляді після "смерті". Компонент змінюючий зовнішній вигляд при смерті (рис 3.37) відповідає за анімацію чи зміни візуального стану об'єкта.

```
namespace Game.Units.Characters.Health
{
    public class ChangeSkinOnDeath : MonoBehaviour
    {
        [SerializeField] private DeathComponent deathComponent;
        [SerializeField] private RespawnableComponent respawnableComponent;
        [SerializeField] private List<GameObject> aliveSkins;
        [SerializeField] private List<GameObject> deathSkins;

        private void Start() => Init();

        private void Init()
        {
            if (deathComponent != null)
                deathComponent.deathSubject.Subscribe(_ => RefreshSkins(true));
            if (respawnableComponent != null)
                respawnableComponent.respawnSubject.Subscribe(_ => RefreshSkins(false));
        }

        private void RefreshSkins(bool isDead)
        {
            aliveSkins.ForEach(x => x.SetActive(!isDead));
            deathSkins.ForEach(x => x.SetActive(isDead));
        }
    }
}
```

Рисунок 3.37 – Компонент змінюючий зовнішний вигляд при смерті

Якщо об'єкт має можливість відродження після смерті, використовується компонент для відродження (рис 3.38). Цей компонент визначає, як і коли об'єкт повинен знову з'явитися в грі.



```
namespace Game.Units.Characters.Health
{
    public class RespawnAfterDeathComponent : MonoBehaviour
    {
        [SerializeField] private DeathComponent deathComponent;
        [SerializeField] private RespawnableComponent respawnableComponent;
        [SerializeField] private double respawnDurationMilliseconds;
        private IDisposable _timerDisposable;

        private void OnEnable()
        {
            deathComponent.deathSubject.Subscribe(_ => Respawn());
        }

        public void SetRespawnTime(double respawnTimeMilliseconds) =>
            respawnDurationMilliseconds = respawnTimeMilliseconds;

        private void Respawn()
        {
            _timerDisposable?.Dispose();

            var timer = Observable.Timer(TimeSpan.FromMilliseconds(respawnDurationMilliseconds));

            _timerDisposable = timer
                .TakeUntilDisable(gameObject)
                .Subscribe(_ => respawnableComponent.InvokeRespawn());
        }
    }
```

Рисунок 3.38 – Компонент для відродження після смерті

Окремий тип об'єкта - ресурс деревини (рис 3.9). Цей ресурс може використовуватися гравцем для будівництва чи інших цілей в грі.

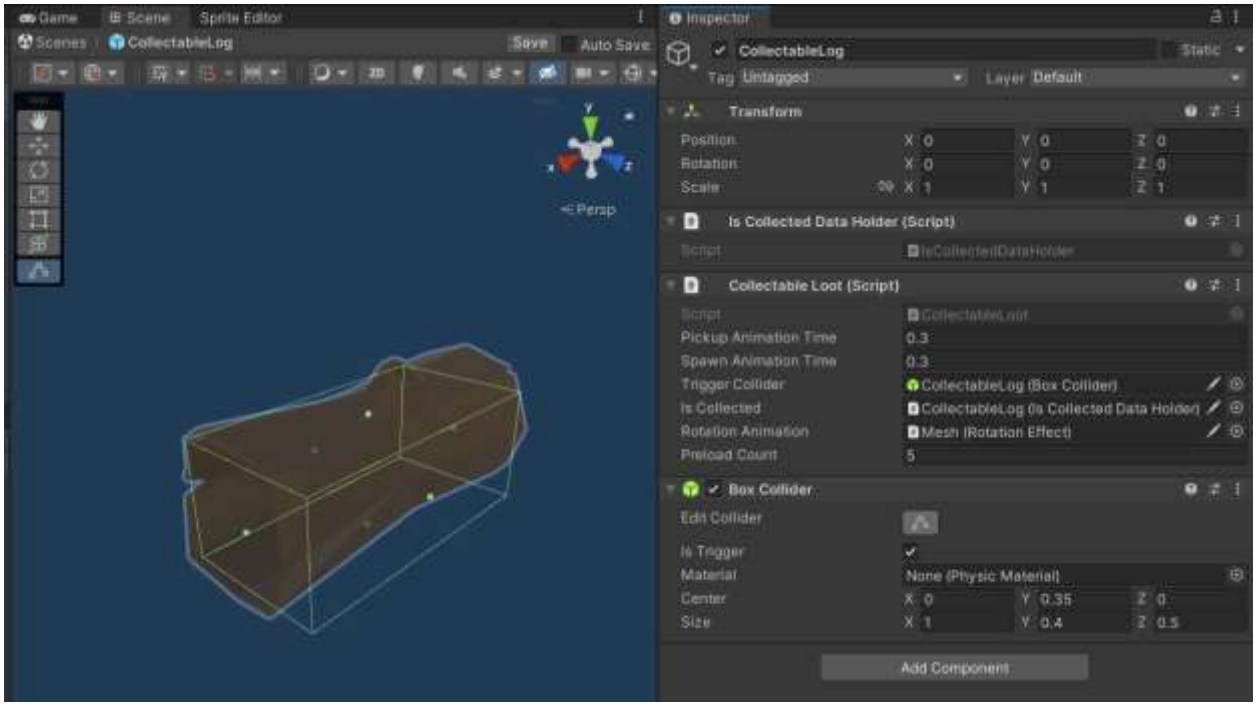


Рисунок 3.39 – Ресурс деревини

На (рис 3.40) демонструється, як гравець може взаємодіяти з ресурсом, збирати його та використовувати для своїх потреб.

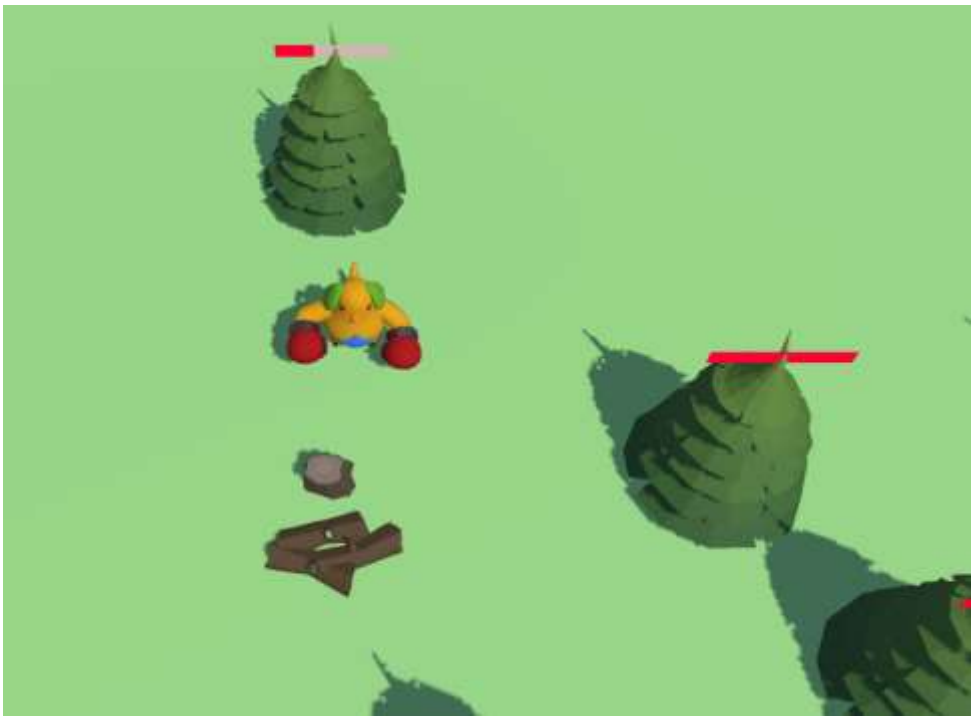


Рисунок 3.40 – Демонстрація механіки видобування ресурсів

В грі присутня механіка будівництва. (рис 3.41) демонстрація процесу, коли гравець будує будівлю, використовуючи ресурси.

Рисунок 3.41 - Процес побудови будівлі

Завершальний скріншот (рис 3.42) показує персонажа гравця біля вже побудованих споруд, що підкреслює геймплейну інтеракцію.

Рисунок 3.42 – Персонаж біля побудованих споруд

Представлено прототипування елементів оточення (рис 3.42) , зокрема трави. Прототипування дозволяє визначити вигляд і структуру оточення перед його реалізацією в грі.

Рисунок 3.43 - Прототипування трави і оточення

Щоб зробити оточення більш реалістичним та динамічним, використовується система часток (рис 3.43). Ця система додає ефекти рухомих часток, таких як трав'яний пухирець або іскри, що поліпшує візуальний ефект гри.

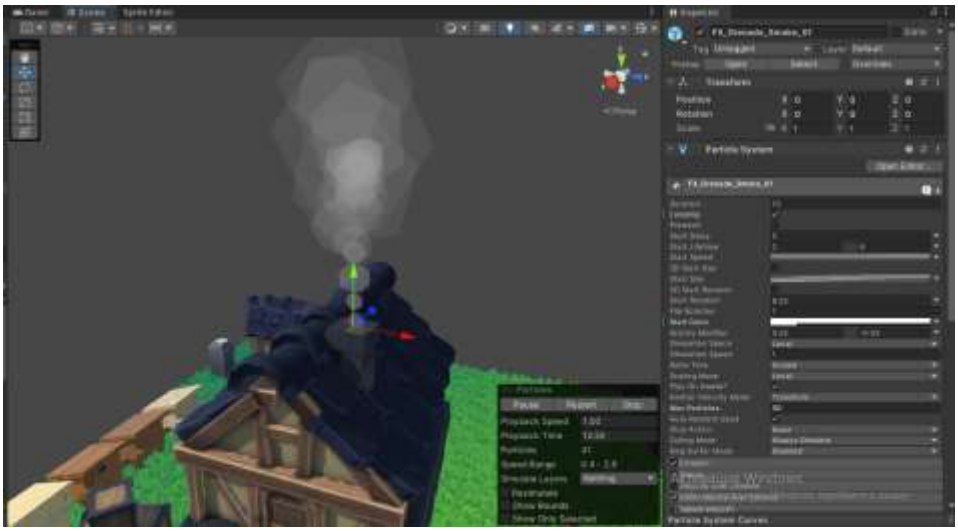


Рисунок 3.44 – Додавання системи часток

3.6 Висновки до розділу

У цьому розділі був проведений огляд ключових аспектів розробки ігрового контенту. Визначено важливі кроки та завдання для успішної реалізації гри. Була обговорена створення базової архітектури проекту, де акцент робився на гнучкості та легкості розширення. Наступним кроком було розглядання процесу створення персонажу, де важливо враховувати унікальні характеристики та взаємодію з оточенням.

Окрема увага була приділена збереженню прогресу гравця та налаштувань на сервері. Ці аспекти є ключовими для забезпечення стабільності та послідовності гри. Зокрема, збереження прогресу гравця

впливає на взаємодію з грою, тоді як збереження налаштувань на сервері дозволяє зручно керувати ігровими даними.

Останнім етапом було розглядання створення елементів оточення. Тут було важливо врахувати деталізацію та взаємодію цих елементів з гравцем, щоб створити навколишній світ, який захоплює та викликає інтерес.

Процес розробки ігрового контенту вимагає виваженого підходу та уваги до деталей. Кожен етап відіграє важливу роль у формуванні цілісного та захоплюючого геймплею.

ВИСНОВКИ

У процесі розробки ігрового додатку були вирішені усі поставлені задачі. Було проведено аналіз існуючих рішень та на основі цього зроблено удосконалення. Було розроблено клієнтську частину з можливістю редагувати ігрові данні віддалено.

Розробка ігрових додатків буде завжди актуальною, тому що людям подобається грати в комп'ютерні ігри, а в сучасності смартфон є майже у кожного. Була зроблена програмна архітектура яку легко розширити, тому в майбутньому цей дипломний проект підлягає для вдосконалення.

Аналітичний огляд різних ігрових рушіїв, показав: які з них можуть бути використані для створення ігрового контенту. Кожен з рушіїв володіє унікальними характеристиками та можливостями для розробки віртуальних світів. Від індустріальних стандартів, таких як Unreal Engine, до популярних та доступних рішень, як Godot, ці програми пропонують різноманітні інструменти, придатні для різних сфер творчості та професійної діяльності.

Аналізуючи існуючі засоби для створення 3D об'єктів, в даній роботі будуть використовуватись дві програми: Unity 3D та JetBrains Rider.

Unity 3D є визнаним лідером у галузі розробки відмінних ігрових додатків. Його потужний інструментарій дозволяє створювати якісні та креативні 3D ігри для різних платформ, включаючи мобільні, настільні, та веб-платформи. Unity 3D володіє широким спектром можливостей у сфері анімації, фізики, штучного інтелекту та інших аспектів геймдеву. Безкоштовний доступ та велика активна спільнота додають йому конкурентну перевагу.

JetBrains Rider є інтегрованою середою розробки, спеціально призначеною для підтримки роботи з .NET-проектами, включаючи розробку ігор в Unity. Його продвинуті можливості редагування коду, рефакторінг та

підтримка роботи з Unity API роблять його відмінним інструментом для програмістів, що працюють над проектами у галузі геймдеву.

Unity 3D та JetBrains Rider взаємодіють між собою ефективно, надаючи команді розробників потужні та гнучкі засоби для творчого процесу. Їхні спільні можливості дозволяють створювати якісні та інноваційні 3D ігри, а також ефективно керувати та вдосконалювати кодову базу проекту.

Додатково, для ефективного керування версіями коду та спільної роботи над проектом використовуватиметься система керування версіями Git. Git надає зручні засоби для відстеження змін, створення гілок розробки, а також об'єднання різних гілок. Використання Git у поєднанні з Unity 3D та JetBrains Rider дозволяє зберігати та контролювати історію змін в коді, полегшує роботу з командою розробників та сприяє стабільності та надійності проекту. Такий підхід дозволяє зберігати інтегровану та організовану систему розробки, а також спрощує виявлення та вирішення конфліктів у випадку одночасної роботи над кодом кількох розробників.

Проведено комплексний аналіз процесів та середовища, необхідних для успішної створення ігрового додатку. Розглянуті етапи розробки включають концептуалізацію та визначення вимог, проектування, розробку, тестування, випуск та розгортання.

На етапі визначення вимог, було визначено основні принципи та напрямки розробки, зокрема аналізувалися ідеї гри та визначалися ключові вимоги до продукту. Ретельний аналіз дозволив визначити цільову аудиторію та виробити чітке бачення проекту.

Була розроблена архітектура гри, визначено геймплейні механіки та розроблено інтерфейс. Цей етап визначає основні принципи функціонування гри та створює основну концепцію продукту.

На етапі розробки використовувалися інструменти, такі як Unity 3D та JetBrains Rider. Unity 3D, завдяки своїм можливостям, став основним інструментом для створення 3D ігрового контенту, тоді як JetBrains Rider забезпечив потужний інтерфейс для розробки та оптимізації коду.

На етапі тестування проводилася перевірка правильності функціонування всіх аспектів гри. Виявлені помилки та недоліки були виправлені, що сприяло поліпшенню якості та надійності продукту.

Було завершено всі останні зміни та виправлення після етапу тестування, підготовлено остаточну версію гри для публікації. Обрані платформи для розгортання визначено відповідно до цільової аудиторії.

Загалом, використання інтегрованого підходу з використанням Unity 3D, JetBrains Rider та системи керування версіями Git дозволило забезпечити ефективну розробку, тестування та випуск ігрового додатку. Проект орієнтований на якісний інтерактивний вміст, а вибір інструментів підтримує комфортну та продуктивну роботу розробників.

Проведено огляд ключових аспектів розробки ігрового контенту. Визначено важливі кроки та завдання для успішної реалізації гри. Була обговорена створення базової архітектури проекту, де акцент робився на гнучкості та легкості розширення. Наступним кроком було розглядання процесу створення персонажу, де важливо враховувати унікальні характеристики та взаємодію з оточенням.

Окрема увага була приділена збереженню прогресу гравця та налаштувань на сервері. Ці аспекти є ключовими для забезпечення стабільності та послідовності гри. Зокрема, збереження прогресу гравця впливає на взаємодію з грою, тоді як збереження налаштувань на сервері дозволяє зручно керувати ігровими даними.

Останнім етапом було розглядання створення елементів оточення. Тут було важливо врахувати деталізацію та взаємодію цих елементів з гравцем, щоб створити навколишній світ, який захоплює та викликає інтерес.

Процес розробки ігрового контенту вимагає виваженого підходу та уваги до деталей. Кожен етап відіграє важливу роль у формуванні цілісного та захоплюючого геймплею.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Офіційний сайт ігрового рушія Unreal Engine [Електронний ресурс] : https://www.unrealengine.com/en-US (дата звернення: 23.11.2023).
2. Офіційний сайт ігрового рушія Godot Engine [Електронний ресурс]: https://godotengine.org/ (дата звернення: 23.11.2023).
3. Офіційний сайт ігрового рушія Unity 3D [Електронний ресурс]: https://unity.com/ (дата звернення: 23.11.2023).
4. Офіційний сайт ігрового рушія Game Maker [Електронний ресурс]: https://gamemaker.io/ (дата звернення: 23.11.2023).
5. Офіційний сайт AppGameKit [Електронний ресурс]: https://www.appgamekit.com/ (дата звернення: 23.11.2023).
6. Офіційний сайт ігрового рушія Cry Engine [Електронний ресурс]: https://www.cryengine.com/ (дата звернення: 23.11.2023).
7. Офіційний сайт Amazon Lumberyard [Електронний ресурс]: https://aws.amazon.com/lumberyard/ (дата звернення: 23.11.2023).
8. Офіційний сайт ігрового рушія RPG Maker [Електронний ресурс]: https://www.rpgmakerweb.com/ (дата звернення: 23.11.2023).
9. Офіційний сайт LibGDX [Електронний ресурс]: https://libgdx.com/ (дата звернення: 23.11.2023).
10. Application of Fast Frequency Shift Measurement Method for INS in Navigation of Drones / D. Avalos-Gonzalez, D.H. Balbuena, V. Tyrsa, V.M. Kartashov, M. Kolendovska, S. Sheiko, O. Sergiyenko, V. Melnyk, F.N. Murrieta-Rico // IECON 2018 – 44th Annual Conference of the IEEE Industrial Electronics Society. – P. 3159–3164.
11. Avalos-Gonzalez, D., Sergiyenko, O., Hernandez-Balbuena, D., Tyrsa,V., Kartashov V.M., V.,Rivas-Lopes, M., Murrieta-Rico, F.N. Constraints definition and application optimization based on geometric analysis of the frequency measurement method by pulse coincidence// Measurement: Journal

of the International Measurement Confederation (USA). 2018, V.126. P. 184-193.

12. Book “Control and Signal Processing Applications for Mobile and Aerial Robotic Systems”, Hardback - Advances in Computational Intelligence and Robotics English. Edited by Oleg Sergiyenko, Moises Rivas-Lopez, Wendy Flores-Fuentes, Julio Cesar Rodríguez-Quiñonez, Lars Lindner. Editorial IGI Global, Hershey, United States, January 2020, 340 páginas. ISBN10 152259924X, ISBN13 9781522599241
13. Cesar Sepulveda-Valdez ; Oleg Sergiyenko ; Vera Tyrsa ; Wendy Flores-Fuentes ; Julio César Rodríguez-Quiñonez ; Fabian Natanael Murrienta-Rico ; Jesús Elías Miranda-Vega ; Paolo Mercorelli ; Marina Kolendovska. "Geometric analysis of a laser scanner functioning based on dynamic triangulation," 2020 IEEE 29th International Symposium on Industrial Electronics (ISIE), Delft, Netherlands, 17-19 of June 2020, pp. 1398-1403, doi: 10.1109/ISIE45063.2020.9152268. https://ieeexplore.ieee.org/abstract/document/9152268
14. Cuauhtémoc Mariscal-García; Wendy Flores-Fuentes; Daniel Hernández-Balbuena; Julio C. Rodríguez-Quiñonez ; Oleg Sergiyenko. "Classification of Vehicle Images through Deep Neural Networks for Camera View Position Selection," 2020 IEEE 29th International Symposium on Industrial Electronics (ISIE), Delft, Netherlands, 17-19 of June 2020, pp. 1376-1380, doi: 10.1109/ISIE45063.2020.9152440. https://ieeexplore.ieee.org/abstract/document/9152440
15. Developing and Applying Optoelectronics in Machine Vision/ O. Sergiyenko, J.C. Rodriguez-Quiñonez, IGI Global, 2016; 341p.
16. Experimental estimation of direction finding to unmanned air vehicles algorithms efficiency by their acoustic emission, /Oleynikov, V., Zubkov, O., Kartashov, V., ...Sheiko, S., Babkin, S.//2019 IEEE International Scientific-Practical Conference: Problems of Infocommunications Science

and Technology, PIC S and T 2019 - Proceedings, 2019, стр. 175-178, 9061337

17. Features of acoustic noise of small unmanned aerial vehicles / Semenets, V.V., Kartashov, V.M., Leonidov, V.I. //Telecommunications and Radio Engineering (English translation of Elektrosvyaz and Radiotekhnika), 2020, 79(11), стр. 985-995
18. Geometric Analysis Of A Laser Scanner Functioning Based On Dynamic Triangulation /Sepulveda-Valdez, C., Sergiyenko, O., Tyrsa, V, Mercorelli, P., Kolendovska, M.// IEEE International Symposium on Industrial Electronics, 29th IEEE International Symposium on Industrial Electronics, ISIE 2020; Delft; Netherlands; 17 June 2020 до 19 June 2020; Volume 2020-June, June 2020, № 9152268, Pages 1398-1403

https://ieeexplore.ieee.org/abstract/document/9152255

https://ieeexplore.ieee.org/document/9161870

19. I. Y. A. Corpus, L.Lindner, O.Sergiyenko. "Transimpedance Amplifier for Laser Scanning System Range Extension," 2020 IEEE 29th International Symposium on Industrial Electronics (ISIE), Delft, Netherlands, 17-19 of June 2020, pp. 1421-1426, doi: 10.1109/ISIE45063.2020.9152487. https://ieeexplore.ieee.org/abstract/document/9152487
20. Ivanov, M., Sergiyenko, O., Mercorelli, P., Hernandez, W.c, Rodriguez Quinonez, J.C.d, Katashov V., Kolendovska, M., Iryna, T. Effective informational entropy reduction in multi-robot systems based on real-time TVS. IEEE International Symposium on Industrial Electronics, 2019-June,8781209, c. 1162-1167.
21. Jonathan J. Sanchez-Castro ; Julio C. Rodríguez-Quiñonez ; Luis R. Ramírez-Hernández ; Guillermo Galaviz ; Daniel Hernández-Balbuena ; Gabriel Trujillo-Hernández ; Wendy Flores-Fuentes ; Paolo Mercorelli ; Wilmar Hernández-Perdomo ; Oleg Sergiyenko ; Félix Fernando González-Navarro. "A Lean Convolutional Neural Network for Vehicle Classification," 2020 IEEE 29th International Symposium on Industrial Electronics (ISIE), Delft,

Netherlands, 17-19 of June 2020, pp. 1365-1369, doi: 10.1109/ISIE45063.2020.9152274.
https://ieeexplore.ieee.org/abstract/document/9152274

22. Lindner, L., Sergiyenko, O., Rivas-López, M., (...), Gurko, A., Kartashov, V.M. Machine vision system for UAV navigation; IEEE, 2016 International Conference on Electrical Systems for Aircraft, Railway, Ship Propulsion and Road Vehicles and International Transportation Electrification Conference, ESARS-ITEC, 2016; pp.1–6. DOI: 10.1109/ESARS-ITEC.2016.7841356.
23. M. Ivanov, O. Sergiyenko, V. Tyrsa, P. Mercorelli, V. Kartashov, W. Hernandez, S. Sheiko, M. Kolendovska. Individual scans fusion in virtual knowledge base for navigation of mobile robotic group with 3D TVS // Proceedings of 44th Annual Conference of IEEE Industrial Electronics Society (IECON).. -2018. – Washington DC, USA. -S. 3187-3192. . ISBN 978-1-5090-6683-4/18/.
24. Murrieta-Rico, F.N., Petranovskii, V., Galvan, D.H., Sergiyenko, O., Yocupicio-Gaxiola, R.I., De Dios Sanchez-Lopez, J. Phase effect in frequency measurements of a quartz crystal using the pulse coincidence principle. 2020 IEEE 29th International Symposium on Industrial Electronics (ISIE), Delft, Netherlands, 17-19 of June 2020, pp. 185-190, 9152255, DOI: 10.1109/ISIE45063.2020.9152255
25. Oleksandr Sotnikov, Vladimir Kartashov, Oleksandr Tymochko, Oleg Sergiyenko, Vera Tyrsa, Paolo Mercorelli, Wendy Flores-Fuentes. Methods for Ensuring the Accuracy of Radiometric and Optoelectronic Navigation Systems of Flying Robots in a Developed Infrastructure. Chapter 16// Machine Vision and Navigation; Springer, Cham. pp.537–578. Editors: Sergiyenko, Oleg, Flores-Fuentes, Wendy, Mercorelli, Paolo. DOI: 10.1007/978-3-030-22587-2_16.
26. Optical detection of unmanned air vehicles on a video stream in a real-time/Kartashov, V., Oleynikov, V., Zubkov, O., Sheiko, S.// 2019 International Conference on Information and Telecommunication

Technologies and Radio Electronics, UkrMiCo 2019 - Proceedings, 2019, 9165362/

27. Principles Of Construction And Assessment Of Technical Characteristics Of Multi-Frequency Atmospheric Sodar In The Humidity Measurement Mode / Kartashov, V.M., Sidorov, G.I., Sheiko, S.A., Kolendovskaya, M.M., Sergienko, O.Yu. // Telecommunications And Radio Engineering (English Translation Of Elektrosvyaz And Radiotekhnika), 2020, ISSN Print: 0040-2508, ISSN Online: 1943-6009, DOI: 10.1615/TelecomRadEng.v79.i4.50, p. 323-333/
28. Research Of The Uncertainty Of Measurement Frequencies And Definitions Of The Frequency Signal In The Waveguide With Respect To Power / Semenets, V.Zakharov, I. Serhiienko, M., Kartashov, V.M, , Kolendovska, M., Hernandez, W., Hipolito, J.I.N., , Tyrsa, V.// 45th Annual Conference of the IEEE Industrial Electronics Society, IECON 2019; Lisbon Congress CenterLisbon; Portugal; 14 October 2019 до 17 October 2019; CFP19IEC-ART; Код 155980, Volume 2019-October, October 2019, № 8927203, Pages 4674-4679
29. Spatial-Temporal Processing Of Acoustic Signals Of Unmanned Aerial Vehicles /Kartashov V.M., Oleinikov V.N., Zubkov O.V., Sheiko S.A., Kolendovska M.M.// Telecommunications And Radio Engineering (English Translation Of Elektrosvyaz And Radiotekhnika), 2020, ISSN Print: 0040-2508, ISSN Online: 1943-6009, DOI: 10.1615/Telecomradeng.v79.i9.40, p. 769-780
30. Stereoscopic Vision Systems In Machine Vision, Models, And Applications (Book Chapter)/ Ramírez-Hernández, L.R., Rodríguez-Quiñonez, J.C., Castro-Toscano, M.J., Kolendovska, M., Murrieta-Rico, F.N.// Machine Vision And Navigation, 2019 Machine Vision and Navigation30 September 2019, Pages 241-265
31. StrelkovaT., KartashovV., Lytyuga A., Strelkov A. Theoretical Methods of Images Processing in Optoelectronic Systems. Chapter 16. // Biometrics:

Concepts, Methodologies, Tools, and Applications; Oleg Sergiyenko and Julio C. Rodriguez-Quiñonez. (341p.), IGI Global, 2017; pp. 361-381. DOI: 10.4018/978-1-5225-0983-7.ch016.

32. StrelkovaT., KartashovV., Lytyuga A.,StrelkovA. Theoretical Methods of Images Processing in Optoelectronic Systems. Chapter 6// Developing and Applying Optoelectronics in Machine Vision; Oleg Sergiyenko and Julio C. Rodriguez-Quiñonez. (341p.) – USA, Herhey, IGI Global, 2016; pp.180-205.
33. Sytnik O., KartashovV. Methods and Algorithms for Technical Visionin Radar Introscopy. Chapter 13// Optoelectronics in Machine Vision-Based Theories and Applications. IGI Global, 2019; pp. 373-391.
34. The Use of Factorization and Multimode Parametric Spectra in Estimating Frequency and Spectral Parameters of Signal/Semenets, V., Kartashov, V., Sergiyenko, O., ...Rodriguez-Quinonez, J.C., Flores-Fuentes, W.//IEEE International Symposium on Industrial Electronics, 2020, 2020-June, p. 215-219
35. Unda, O.F., Hernandez, W., Vargas, O., Mendez, A., Sergiyenko, O., Tyrsa, V. Construction of a robotic platform of differential type for first-year students of electronic engineering, 2020 International Symposium on Power Electronics, Electrical Drives, Automation and Motion, SPEEDAM 2020, 24-26 de junio de 2020, Sorrento, Italia, pp. 538-543, 9161870, DOI: 10.1109/SPEEDAM48782.2020.9161870
36. Use of Acoustic Signature for Detection, Recognition and Direction Finding of Small Unmanned Aerial Vehicles/Kartashov, V., Oleynikov, V., Koryttsev, I., ...Babkin, S., Selieznov, I.//Proceedings - 15th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering, TCSET 2020, 2020, p. 377-380/
37. V. Semenets; Vladimir Kartashov ; Oleg Sergiyenko; Vyacheslav Tikhonov ; Paolo Mercorelli ; Sergiy Sheiko ; Nataliya Chmelarova. "The Use of Factorization and Multimode Parametric Spectra in Estimating Frequency and Spectral Parameters of Signal," 2020 IEEE 29th International Symposium on

Industrial Electronics (ISIE), Delft, Netherlands, 17-19 of June 2020, pp. 215-219, doi: 10.1109/ISIE45063.2020.9152238.

https://ieeexplore.ieee.org/abstract/document/9152238