Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Худіка Дмитра Володимировича*
(ПІБ)

академічної групи *121-20ск-1*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)

на тему: *Розробка програмного забезпечення розважального застосунку в середовищі Unity з дотриманням комерційного пайплайну для 3D моделей.*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*проф. Мещеряков. Л.І.*			
розділів:				
спеціальний	*проф. Мещеряков. Л.І.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2023

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

(підпис) (прізвище, ініціали)

« » 2023 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *121-20ск-1* *Худіка Д.В.*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка програмного забезпечення – розважального застосунку в середовищі Unity з дотриманням комерційного пайплайну для 3D моделей*

затверджена наказом ректора НТУ «ДП» від *18.05.2023* *№268-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2023 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки.***	***27.05.2023 р.***

Завдання видав *проф. Мещеряков. Л.І.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання *Худік Д.В.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2023 р.*
Термін подання кваліфікаційної роботи до ЕК: *13.06.2023 р.*

РЕФЕРАТ

Пояснювальна записка: 93 с., 49 рис., 4 дод., 20 джерел.

Об'єкт розробки: програмне забезпечення - розважальний застосунок в середовищі Unity з дотриманням комерційного пайплайну для 3d - моделей.

Мета кваліфікаційної роботи: створення розважального застосунку в середовищі Unity, з використанням мови програмування C#, з обов'язковим дотриманням комерційних норм та ліцензій для проекту. Створення 3d-моделей по виробничому пайплайну.

В вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточняється постановка завдання.

Перший розділ присвячено аналізу предметної галузі, визначенню актуальність завдання та призначення розробки, сформульовано постановку завдання, зазначено вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі проаналізовані наявні рішення, обрано платформи для розробки, виконано проектування і розробка програмного забезпечення, описана робота програми, алгоритм і структура її функціонування, а також виклик та завантаження програми, визначено вхідні і вихідні дані, охарактеризовано склад параметрів технічних засобів.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Практичне значення полягає у створенні комерційно стійкого та потенційно-успішного розважального додатку.

Актуальність даного програмного продукту визначається великими попитом, та має ще більш велику комерційну перспективу. Зростання глобальної індустрії геймінгу та збільшенням кількості комп'ютерів – створює унікальну можливість для підприємців та розробників отримати потенційний прибуток і розробити більш інноваційні та високоякісні розважальні додатки.

Список ключових слів: КОМП'ЮТЕР, ПРОЕКТУВАННЯ, ГРА, ДОДАТОК, ПАЙПЛАЙН, КОМЕРЦІЯ, ПРОДУКТ, ПРОГРАМУВАННЯ, МОДЕЛЮВАННЯ, 3D ГРАФІКА, UNITY, BLENDER, C#.

ABSTRACT

Explanatory note: 93 p., 49 fig., 4 appx., 20 sources.

The object of development: the software is an entertainment application in the Unity environment with compliance with the commercial pipeline for 3d models.

The purpose of the qualification work: to create an entertaining application in the Unity environment, using the C# programming language, with mandatory compliance with commercial norms and licenses for the project. Creation of 3d models of the production pipeline.

In the introduction, the analysis and current state of the problem is considered, the purpose of the qualification work and the field of its application are specified, the justification of the relevance of the topic is given, and the statement of the task is clarified.

In the first section, the subject area is analyzed, the relevance of the task and the purpose of the development is determined, the task statement is formulated, and the requirements for software implementation, technologies and software tools are specified.

In the second section, available solutions are analyzed, platforms for development are selected, software design and development is performed, the program's operation, algorithm and structure of its functioning are described, as well as program calling and loading, input and output data are determined, and the composition of technical means parameters is characterized.

In the economic section, the labor intensity of the developed information system is determined, the cost of work on creating the program is calculated, and the time for its creation is calculated.

The practical meaning is to create a commercially viable and potentially successful entertainment application.

The relevance of this software product is determined by high demand, and it has an even greater commercial perspective. The growth of the global gaming industry and the increase in the number of computers - creates a unique opportunity for entrepreneurs and developers to gain potential profits and develop more innovative and high-quality entertainment applications.

Keywords list: COMPUTER, DESIGN, GAME, APP, PIPELINE, COMMERCE, PRODUCT, PROGRAMMING, MODELING, 3D GRAPHICS, UNITY, BLENDER, C#.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

GPU – графічний процесор;

CPU – центральний процесор

ПК – персональний комп’ютер;

VR – віртуальна реальність;

NPC – неігровий персонаж;

HUD – частина графічного інтерфейсу;

HP – високополігональна модель;

LP – низькополігональна модель;

Baking - процес запікання 3D моделей;

NP – карта нормалей;

AO – модель, або карта затінення моделі.

ВСТУП

У сучасному світі інформаційних технологій, розробка розважальних застосунків є актуальною та перспективною галуззю. За останні десятиліття спостерігається швидкий розвиток індустрії відеоігор, включаючи ігрові додатки для мобільних пристроїв та персональних комп'ютерів. Застосунки, розроблені на основі сучасних технологій, стають все більш популярними серед користувачів, надаючи їм нові можливості для відпочинку та розваг.

Метою даної кваліфікаційної роботи є розробка розважального застосунку в середовищі Unity з дотриманням комерційного пайплайну для 3D-моделей.

Unity є однією з провідних інтегрованих середовищ розробки відеоігор, яке надає широкий набір інструментів для створення якісних та комерційно успішних розважальних проектів. У цій роботі буде здійснено аналіз можливостей Unity для розробки розважальних застосунків, зосереджено увагу на професійному пайплайні для створення 3D-моделей.

Актуальність теми обумовлена постійним зростанням попиту на розважальні застосунки та ігрові продукти. Користувачі все більше очікують від застосунків нових та захоплюючих геймплейних елементів, цікавого ігрового досвіду, та унікальності на ринку. Розвиток 3D-моделювання та візуалізації дозволяє створювати більш деталізовані та привабливі графічні об'єкти. Однак, розробка 3D-моделей для ігрових проектів потребує ефективного використання сучасних інструментів.

Постановка завдання цієї роботи полягає в дослідженні можливостей середовища Unity для розробки розважальних застосунків з використанням комерційного пайплайну для 3D-моделей. Для досягнення цієї мети будуть виконані такі завдання:

Ця кваліфікаційна робота наддасть практичні рекомендації для розробників, які прагнуть створювати якісні та конкурентоздатні застосунки. Розробка даного проекту є актуальною, оскільки забезпечує нові можливості та інструменти для розробників у сфері розважальних технологій.

Також однією з відмінних особливостей двигуна Unity – є використання мови програмування C#. Що дозволяє розробникам підвищувати рівень володіння мовою і відкриває перспективи у інших напрямах програмування. Це було практично використано, для написання навчальних матеріалів по темі “Розробка ігор на Unity та основи C#”.

У наступних розділах цієї роботи будуть розглянуті теоретичні та практичні аспекти розробки розважальних застосунків, детально проаналізовані можливості середовища Unity та інших програмних засобів та інструментів. Завершаючий розділ міститиме висновки з отриманих результатів та рекомендації для подальших досліджень.

Аналіз сучасного стану проблеми, актуальність теми та постановка завдання є ключовими елементами роботи, які дозволяють зрозуміти контекст та цілі проведеного дослідження.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Геймдев (Game Development, з англ. розробка відеоігор) - це галузь, що об'єднує творчість, інновації та технології. Вона стала однією з найшвидше зростаючих та найпопулярніших галузей розваг у світі.

З поширенням популярності геймінгу поширився й попит на більш якісні та цікаві ігрові продукти, що дало змогу розробникам не тільки проявити себе, а й створювати комерційно вдалі проекти.

Ігрова індустрія стверджує себе як масовий ринок, тому при якісному виконанні, та створенні може принести значний прибуток як розробнику, так і фірмі в цілому. До основних видів комерційної вигоди можна віднести продаж самого продукту, підписки, мікро транзакції, реклама, спонсорські угоди та ін.

З появою різноманітних ігрових платформ, включаючи ПК, консолі, мобільні пристрої, пристрої віртуальної реальності, розробники отримують доступ до широкої аудиторії, що відкриває можливості для залучення різних сегментів гравців та поширенням продукту на різних платформах. Створення розважальних застосунків надає розробникам можливість втілити свої творчі задуми та ідеї в життя.

Загалом розробка ігор є перспективним та вигідним напрямком.

Створені ігрові продукти класифікуються за бюджетом на декілька категорій, в залежності від розміру і доступних ресурсів для їх створення:

- Незалежні (інді) ігри – розважальні застосунки які створені невеликою командою, або навіть однією людиною, з обмеженим бюджетом. Самому розробляти повноцінну гру є надскладною задачею, тому такі розробники найчастіше долучаються до фінансування через краудфандингові платформи або розробляють за власні кошти. Бюджети цих ігор можуть бути від декількох тисяч до кількох сотень тисяч доларів.

- AAA-ігри – великомасштабні проекті зі значною кількістю розробників і значними бюджетами. Ці ігри зазвичай створюються великими студіями з високим рівнем фінансування від видавців або інвесторів. Бюджети таких ігор можуть досягати мільйонів і навість мільярдів доларів[1].
- AA-ігри – категорія яка включає в себе розважальні продукти більш-менш помірного бюджети, які орієнтовно є посередині між AAA-проектами та інді-розробниками.
- Мобільні ігри – продукти які розробляються для мобільних платформ, таких як телефони і планшети. Бюджети можуть мати дуже різні, від дуже низьких – безкоштовні ігри з вбудованими ігровими покупкам, до дуже великих які розробляються великими студіями.
- VR-ігри – категорія до якої входять продукти, що розроблені для пристроїв віртуальної реальності. Можуть мати так само різні бюджети, як і мобільні ігри.

На час створення цієї кваліфікаційно роботи, в світі створення ігор, титанами індустрії вважають такі компанії: Nintendo, Capcom, Rockstar Games. Остання названа студія є лідером, процитуючи “Rockstar Games – це кракен від світу ігрової індустрії, перемогти якого просто неможливо”[2].

При професійному підході до процесу розробки розважальних продуктів, важливо враховувати ліцензійні норми та вимоги, які встановлені з метою забезпечення законності і захисту прав власників інтелектуальної власності. Це може включати отримання ліцензій на використання марок, ліцензійних прав на використання літературних або музичних творів, а також дотримання авторських прав і правил використання третіх сторін.

Оптимізація процесу виробництва гри також відіграє важливу роль у забезпечення ефективності і успішності проекту. Це включає в себе планування ресурсів, використання інструментів і технологій, адаптацію до змінних умов розвитку ігрової індустрії, раціоналізацію процесів виконання завдань, а також дотримання пайплайну виробництва візуальних матеріалів для максимально ефективної роботи.

Одним з важливих елементів оптимізації процесу є використання ефективного пайплайну виробництва 3D моделей. За його ефективності, у розробці, можна гарантувати високу якість ігрових моделей, а дотримання послідовності та оптимізації процесу дозволить зберегти час, ресурси та забезпечити вчасне завершення модельного етапу проекту.

Комерційний пайплайн для 3D-моделей включає кілька основних етапів:

- Концепція та скетчі. Даний етап передбачає створення концепції для 3D моделі та розробку скетчів або прототипів, щоб визначити форму та зовнішній вигляд моделі. Цей етап також не виключення для кваліфікаційної роботи, тому, для проекту було також розроблені концепції та скетчі:
- Моделювання. Після визначення концепції 3D-моделі наступним кроком є етап моделювання. Моделювання може бути здійснене за допомогою спеціалізованого програмного забезпечення, такого як Blender, Maya, 3ds Max. На цьому етапі створюється геометрична форма моделі, її розміри та деталі.

Рис. 1.1. Скетчі до 3D моделей

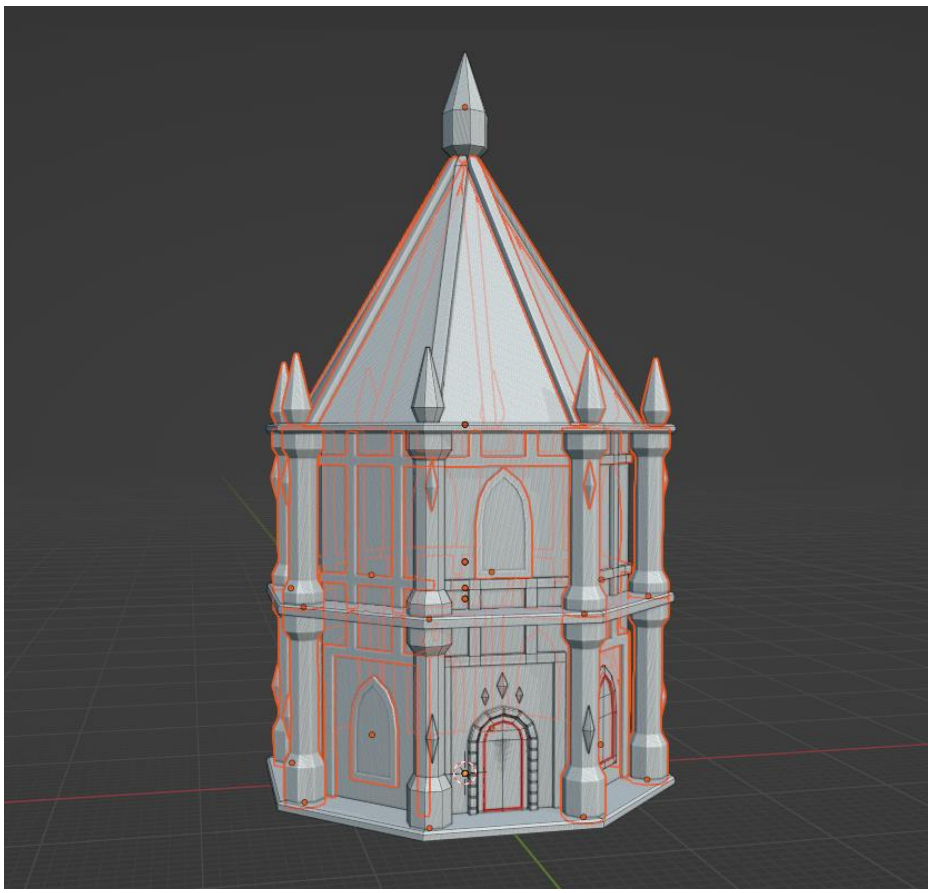

Рис. 1.2. Етап моделювання для 3D моделі проекту

- Топологія або ретопологія. Один з важливіших в моделюванні етапів. Під час ретопології моделі задається правильна організація полігонів та вершин. Гарна топологія (полігональна сітка) забезпечує оптимальну деталізацію та плавні переходи між поверхнями моделі. В залежності від конкретної ситуації, може змінюватись чергування етапів. Якщо спочатку була розроблена low poly модель (низька кількість полігонів, в середньому до 10 тисяч полігонів) то ця модель ускладняється до hight poly (висока кількість полігонів, в середньому від 100 тисяч) – в цьому випадку hight poly модель буде використовуватися в подальших етапах. Та навпаки якщо спочатку було створена hight poly, тоді відбувається етап ретопології, та оптимізації полігональної сітки тому що саме low poly модель буде задіяна у грі.

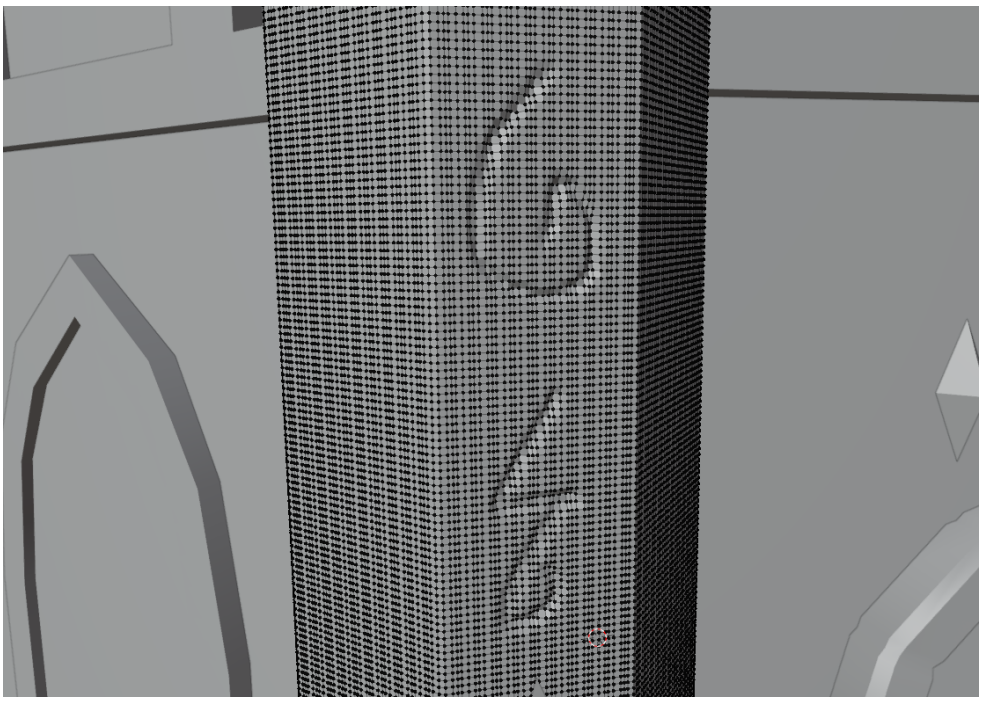

Рис. 1.3. Щільність полігональної сітки hight poly моделі для створення рельєфу колони

- Розгортання UV або UV розгортка. Наступним кроком є розгортання UV. Це процес визначення текстурних координат на моделі, щоб матеріали і текстури можна було правильно прикріпити до поверхні. Розгортання UV дозволяє розмістити текстури на моделі без спотворень або зшивань. Інакше кажучи цей процес дуже схожий на орігамі – з листа паперу виходить об’ємна фігури, тільки навпаки, треба добавити розрізи, щоб 3D модель можна було умістити на плоскій поверхні. Процес створення UV розгортки, показаний на рисунку 1.4.

- Запікання або bake. Після розгортання UV для low poly моделі, та коли вже готова hight poly модель, починається етап запікання. Цей етап дозволяє розмістити деталі високо полігональної моделі на низько полігональній шляхом запікання карти з назвою Normal Map. Це наче зображення деталей на текстурі, яка в результаті буде накладена на low poly модель. На цьому етапі важливо розмістити деталі моделей перед запіканням в одних координатах, при цьому варто їх розмістити таким чином, щоб хоча б у двох проекціях, ці деталі не накладались на інші частини моделі, аби запобігти

появи артефактів. Підготовлена модель до етапу запікання показана на рисунку 1.5.

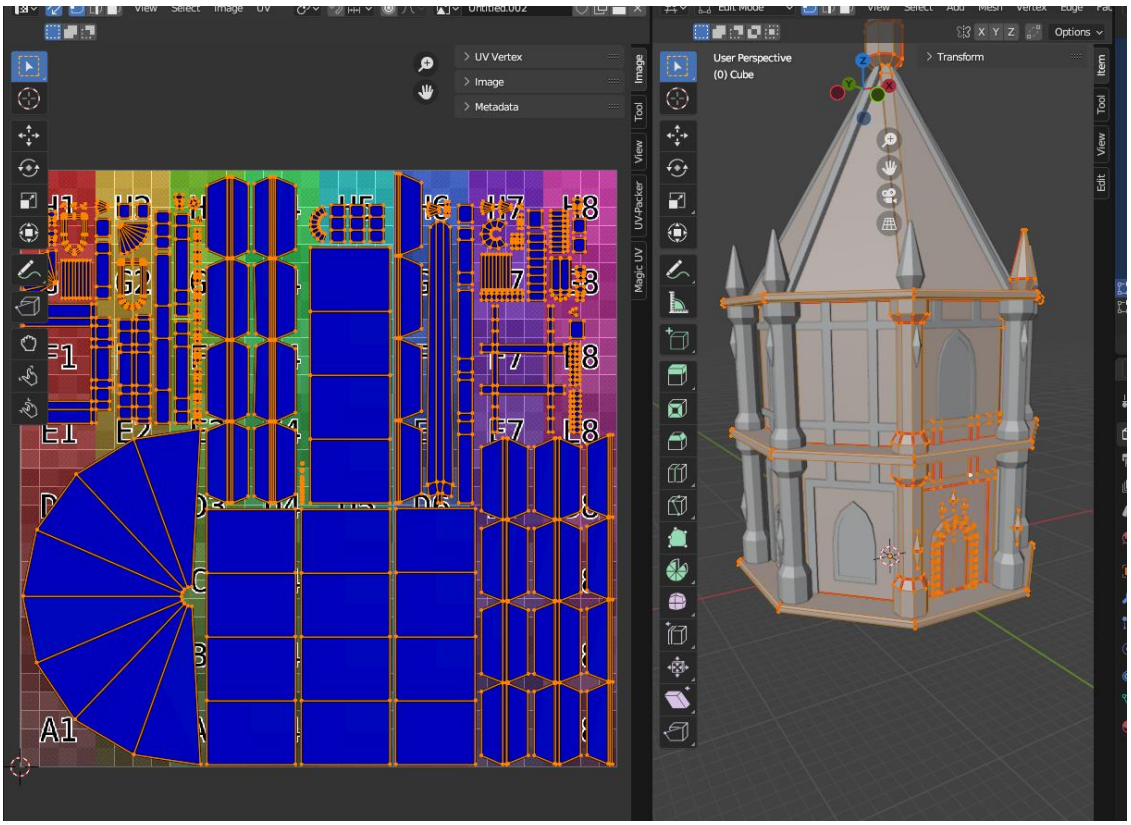

Рис. 1.4. UV-розгортка створеної low poly моделі

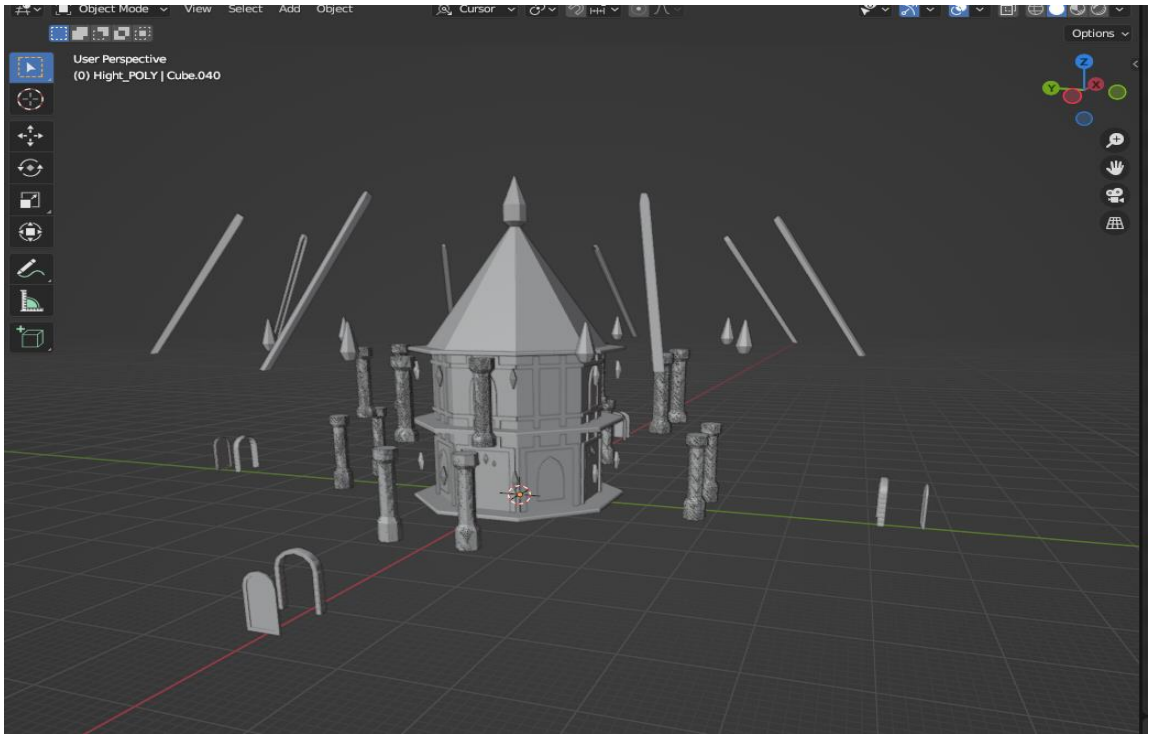

Рис. 1.5. Підготовлена модель до етапу запікання

Після правильного розміщення всіх об’єктів 3D моделей, проводиться запікання. Це здебільшого автоматичний процес який можна провести в різних програмах, результат буде майже однаковим. Запікання для цієї моделі було проведено в Substance Painter. В результаті цього процесу, з’явиться карта NormalMap.

Рис. 1.6. NormalMap для даної 3D моделі

- Текстурування. Після розгортання UV, та після запікання (якщо цей етап був проведений) відбувається створення текстур для моделі. Цей етап включає в себе створення або застосування текстур, матеріалів та фарбування моделі, щоб надати їй потрібний вигляд. Цей етап трохи відрізняється в залежності від конкретного стилю та напрямку 3D моделі. Якщо модель повинна бути реалістичною, то використовуються різноманітні фотознімки для накладання на об’єкти. Дана 3D моделі повинна буду в стилізованому стилі, згідно скетчів на першому етапі – відповідно текстури повинні бути намальовані. Для цього етапу використовувався Substance Painter.

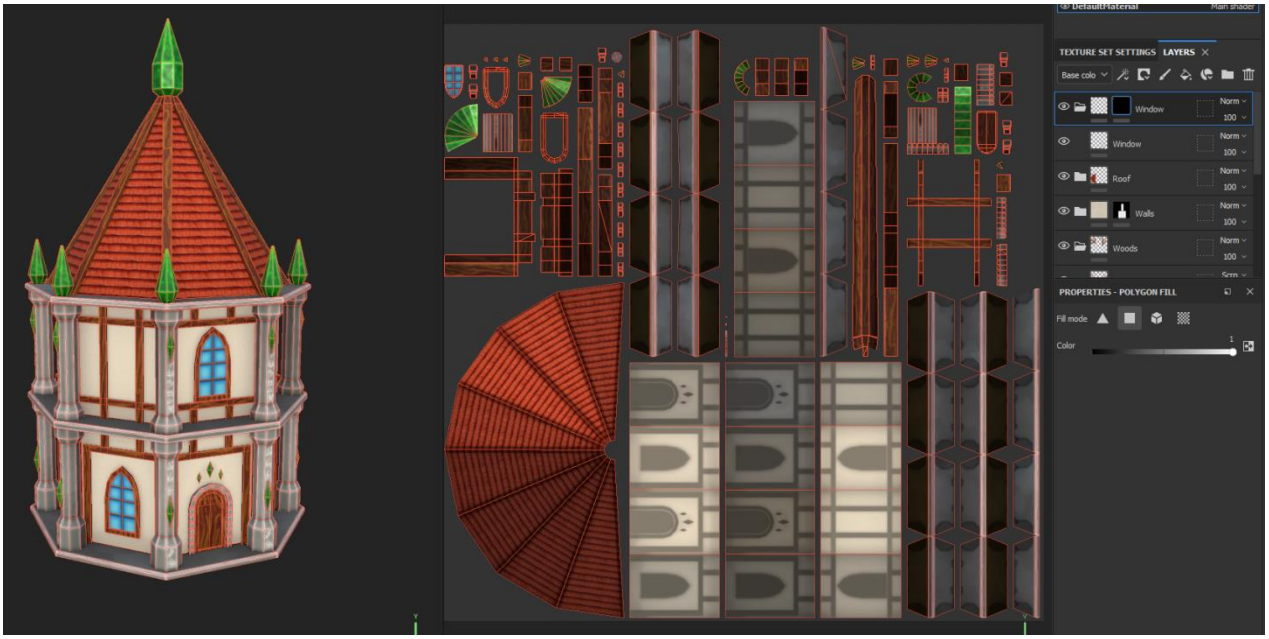

Рис. 1.7. Етап текстурування даної 3D моделі у стилізованому стилі

Також під час етапу текстурування створюється зовнішній вигляд моделі, не лише шляхом малювання певних текстур, а й шляхом запікання певних карт, які покращать зовнішній вигляд моделі. До них можна віднести такі карти:

Ambient Occlusion Map – використовується для створення ефекту затінення в ущільнених або прихованих областях моделі. Вона додає більшу глибину та реалізм освітлення моделі.

Height Map – використовується для додавання тривимірного відчуття моделі, містить інформацію про висоту кожної точки поверхні, зо дозволяє створювати візуальні ефекти рельєфу та текстур.

Specular Map – визначає ступінь блиску або відблиску на поверхні моделі, вона використовується для реалістичного відображення матеріалів, таких як метал або скло.

Та інші карті, які визначаються за потребою або розробника, або постановником завдання.

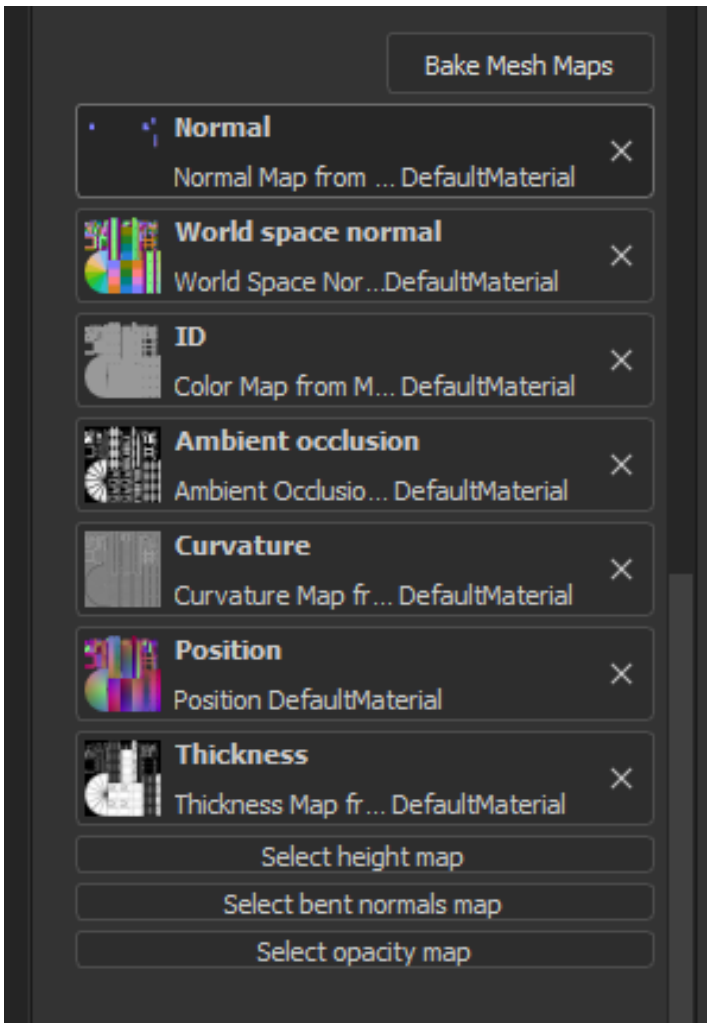

Рис. 1.8. Карти які використовуються під час роботи з 3D об’єктом

- Риггінг. (опціонально). Процес створення скелетної структури для моделі, що дозволяє анімувати її. Риггінг включає створення кісток, з’єднаних між собою за допомогою рухомих з’єднувачі, а також налаштування контролерів для керування рухом моделі. Цей етап здебільшого для моделей персонажів, техніки, механізмів які повинні весь час рухатись, тому він не використовується для моделей оточення – будівель, локацій, предметів тощо. Вигляд ригу персонажу показано на рисунку 1.9.

- Анімація. Після риггінгу можна створювати анімацію для моделі, встановлюючи рухи, виконуючи ключові кадри і створюючи переходи між різними позами. Приклад наведено на рисунку 1.10.

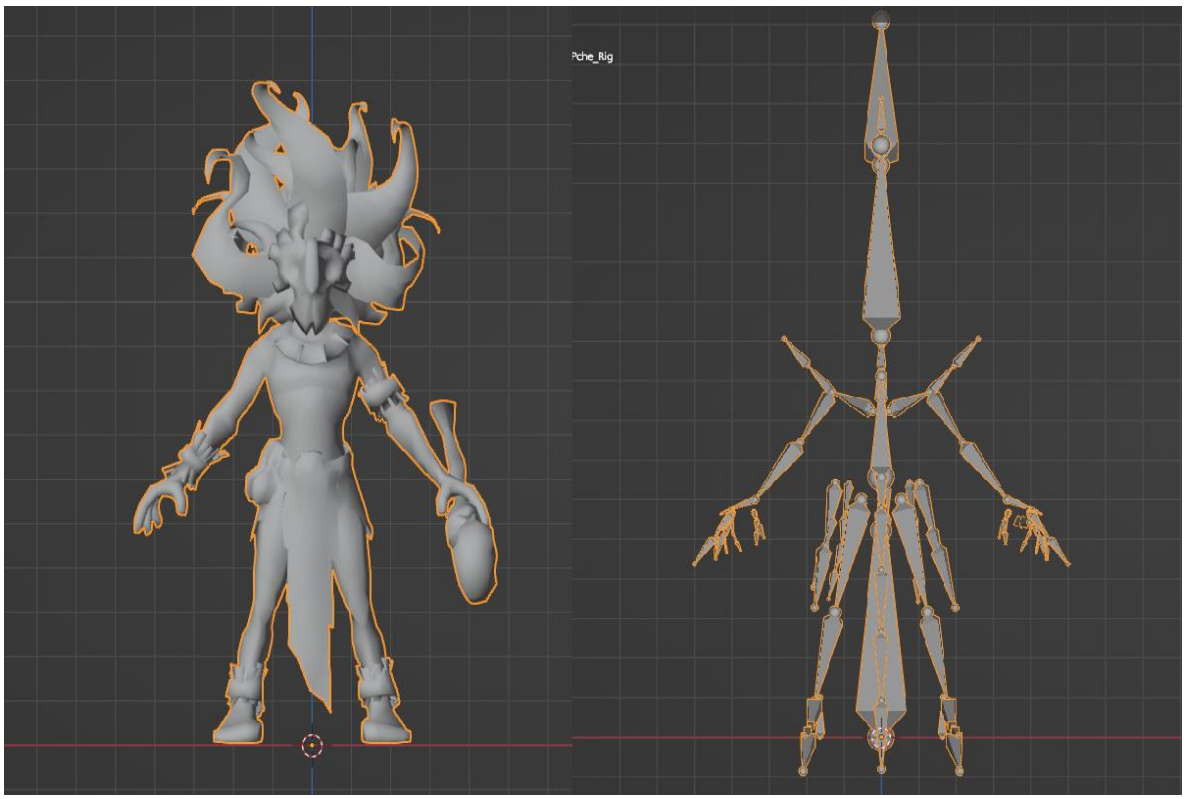

Рис. 1.9. Риггінг однієї з моделі ворога

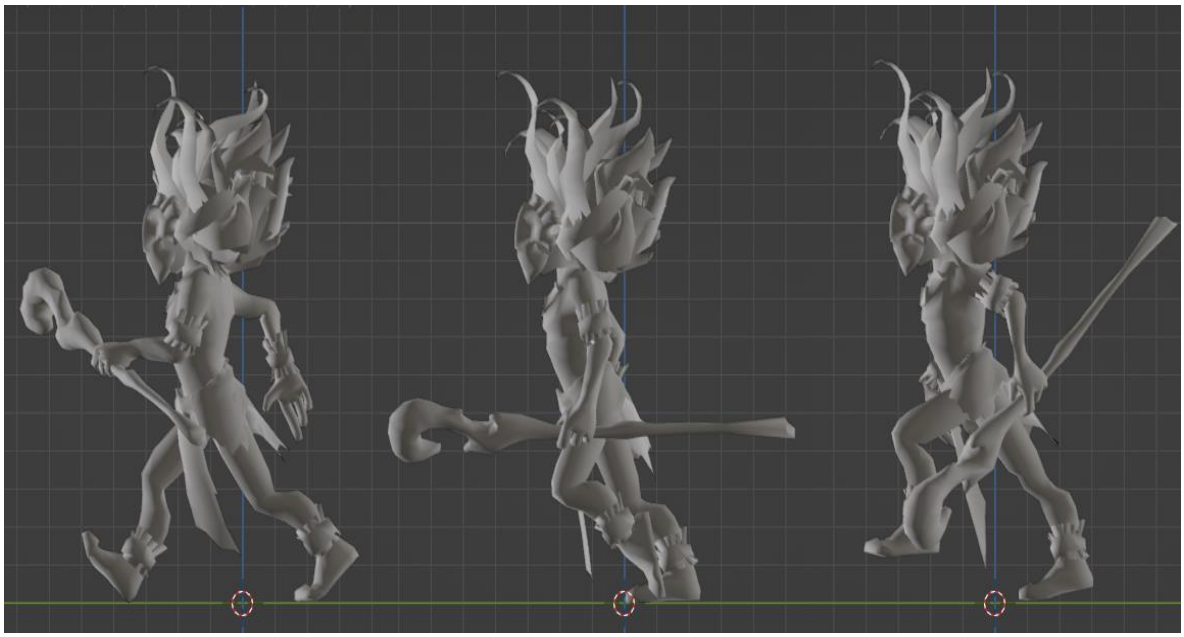

Рис. 1.10. Створення анімації героя

- Оптимізація. Цей етап включає оптимізацію моделі для забезпечення ефективності та оптимального використання рівнів деталізації (LOD) для різних відстаней, видалення непотрібних елементів та оптимізацію текстур. Використання різних рівнів деталізації дуже позитивно впливає на приріст продуктивності гри. Це дуже активно використовується особливо при створенні локації – дерев, каміння та інших об’єктів. Приклад налаштування системи LOD, на об’єкті дерева, зображено на рисунку 1.11.

- Рендеринг або Імплементація. Якщо потрібне зображення 3D моделі для певних графічних матеріалів - треба пройти процес рендерингу. В результаті вдалого процесу, ми отримаємо дуже якісне зображення потрібної 3D моделі. Після вдалих етапів пайплану – наступає фінальний етап, у якому 3D модель імпортується в гру – цей етап називають імплементацією. Потрібна модель інтегрується в гральний двигун, відбувається налаштування колізії, фізики та інших параметрів щоб модель належним чином працювала у грі. Приклад імпорту готових моделей у гру зображено на рисунку 1.12.

Кожен з цих етапів вимагає спеціалізованих навичок та використання відповідних програмних засобів для створення високоякісних та оптимізованих 3D моделей для гри. Своєчасне та якісне виконання кожного з етапу, дозволяє максимально ефективно та швидко розробляти 3D моделі, що позитивно впливає на розробку проекту в цілому.

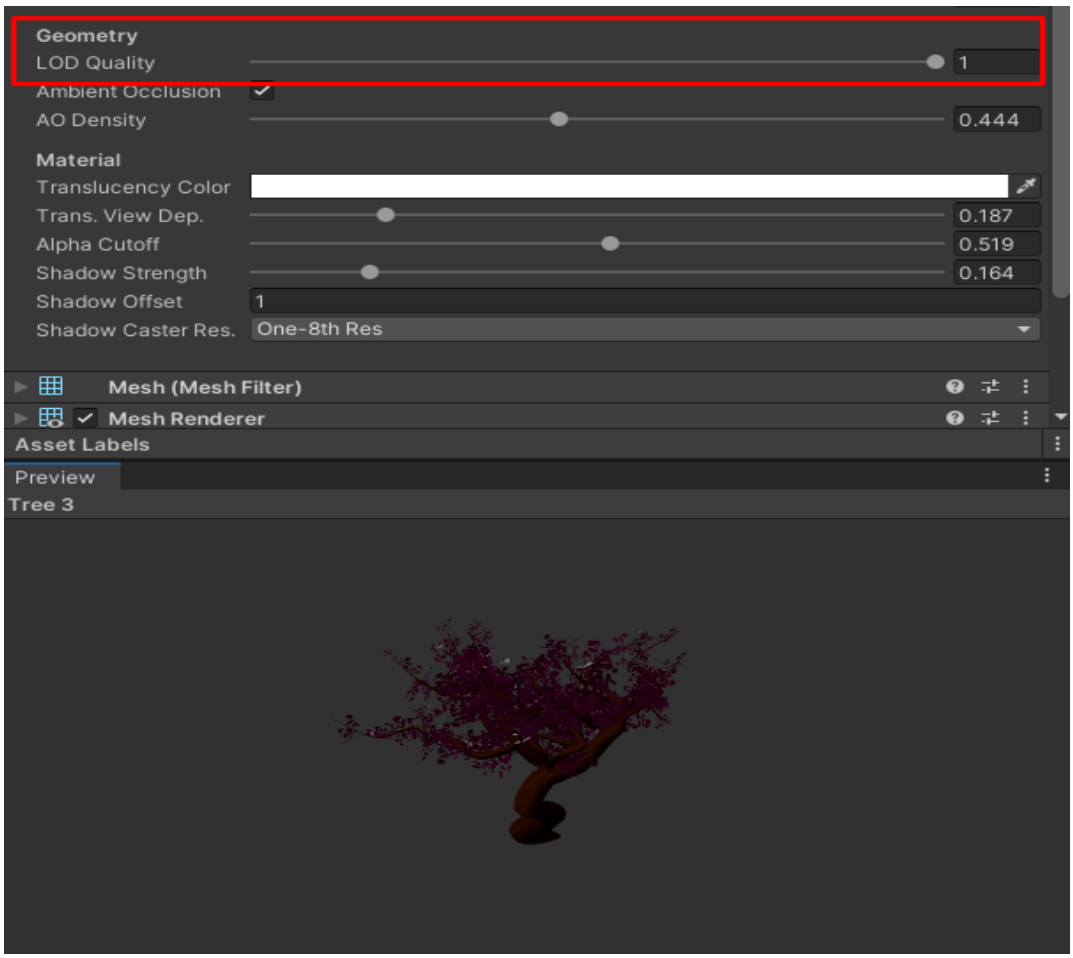

Рис. 1.11. Налаштування рівнів деталізації

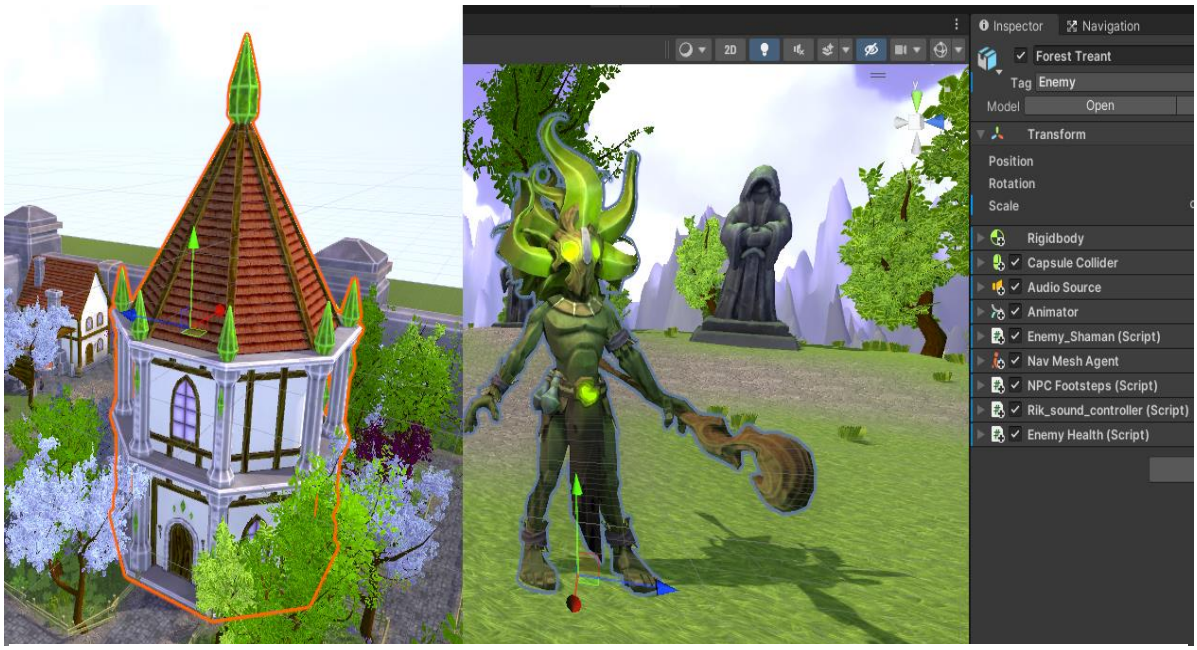

Рис. 1.12. Імпортовані та налаштовані моделі в двигуні

1.2. Призначення розробки та галузь застосування

3D моделювання використовується в багатьох галузях, оскільки це дуже потужний інструмент для створення візуальних об’єктів і сцен. 3D моделювання використовується в таких галузях:

- Комп'ютерні ігри: 3D моделі використовуються для створення персонажів, об'єктів, локацій і анімації у відеоіграх.
- Кіноіндустрія: 3D моделювання використовується для створення спеціальних ефектів, візуалізації сцен, моделювання персонажів та створення віртуальних світів.
- Архітектура та дизайн інтер'єрів: 3D моделі допомагають архітекторам і дизайнерам візуалізувати будівлі, квартири, меблі та інші об'єкти перед їх реалізацією.
- Промисловий дизайн: 3D моделювання використовується для створення прототипів виробів, інженерних деталей, машин і обладнання.
- Медицина: 3D моделі використовуються для виготовлення протезів, моделювання органів та тканин для медичних досліджень, планування хірургічних операцій та навчання майбутніх лікарів.
- Реклама і маркетинг: 3D моделювання дозволяє створювати реалістичні візуалізації продуктів для рекламних кампаній і презентацій.
- Освіта: 3D моделювання використовується у навчанні для створення інтерактивних віртуальних середовищ, анатомічних моделей, симуляцій та навчальних матеріалів.
- Анімація і мультфільми: 3D моделювання є основою для створення комп'ютерної анімації і мультфільмів.

Це лише кілька прикладів галузей, де використовується 3D моделювання. Завдяки своїй гнучкості і можливостям, 3D моделювання знаходить застосування в багатьох інших сферах творчості і промисловості.

1.3. Підстава для розробки

Відповідно до освітньої програми, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу.

Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується наказом ректора.

Отже, підставами для розробки (виконання кваліфікаційної роботи) є:

− освітня програма 121 «Інженерія програмного забезпечення»;

− навчальний план та графік навчального процесу;

− наказ ректора Національного технічного університету «Дніпровська політехніка» № 268-с від 18.05.2023 р;

− завдання на кваліфікаційну роботу на тему “ Розробка програмного забезпечення - розважального застосунку в середовищі Unity з дотриманням комерційного пайплайну для 3d моделей ”.

1.4. Постановка завдання

Завдання даної роботи є розробка розважального застосунку в середовищі Unity з дотриманням комерційного пайплайну для 3d моделей.

В результаті необхідно спроектувати та розробити гру для платформи Windows, дотримуючись правил комерційного пайплану для 3d моделей, використовуючи ігровий двигун Unity, з дотриманням всіх ліцензійних угод на використані графічні та аудіо матеріали.

Поставлена задача буде досягнена при виконанні таких умов:

- вивчення предметної галузі;
- написання коду програми;
- моделювання, та інтеграція в ігровий двигун Unity;
- використання комерційного пайплайну для 3d моделей;
- дотримання всіх ліцензійних угод на використані матеріали.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Кінцевий продукт повинен дотримуватися наступних функціональних вимог:

- Заготовлений зрозумілий, та лаконічний графічний інтерфейс;
- Можливість керування персонажем за допомогою клавіатури та миші.
- Використаний штучний інтелект для ворогів.
- Коректна робота моделей та оточення.
- Наявність декількох ігрових локацій.

1.5.2. Вимоги до інформаційної безпеки

Для забезпечення інформаційної безпеки користувача повинні бути забезпечені такі умови:

- Своєчасне встановлення оновлень;
- Відсутність необхідності реєструватися та створювати особистий акаунт.
- Використання ігрового додатку без потреби мережевого з’єднання.
- Закритий вихідний код продукту.

1.5.3. Вимоги до складу та параметрів технічних засобів

Відповідні мінімальні характеристики необхідні для роботи додатку на Windows:

- OS: Windows 7 (SP1+) або вище;
- CPU: AMD Ryzen 3 1200 3.2 GHz або аналогічні;
- GPU: Nvidia GeForce 1050 Ti або аналогічні;
- DirectX: Версії 10;
- Вільне місце на жорсткому дискі: 2ГБ;
- Оперативна пам’ять: 8 Гб.

Відповідні рекомендовані характеристики для найкращої роботи додатку на Windows:

- OS: Windows 10 (22H2) або Windows 11;
- CPU: AMD Ryzen 5 3600 3.6 GHz або вище;
- GPU: Nvidia GeForce GTX 1660 Super або вище;
- DirectX: Версії 12;
- SSD накопичувач з вільним місцем: 2ГБ;
- Оперативна пам’ять: 16 Гб або вище.

1.5.4. Вимоги до інформаційної та програмної сумісності

Додаток створений на основі ігрового двигуна Unity та призначений для використання на операційній системі Windows, а мовою програмування був обраний C#.

Додаток потребує встановлення в систему за допомогою інсталлера. Оновлення відбувається шляхом перевстановлення додатку.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Результатом даної кваліфікаційної роботи має бути ігровий додаток жанру Action RPG, розроблений в двигуні Unity, у стилізованому візуальному стилі, створений за допомогою 3D-графіки.

Проект повинен продемонструвати можливості створення комерційно-стійкого розважального додатку, з дотриманням всіх ліцензій на візуальні та аудіо матеріали. Продемонструвати реалізацію відкритого світу, та реалізації використання штучного інтелекту для ворогів.

2.2. Опис застосованих математичних методів

Під час створення 3D-моделей для проекту використовувався математичний метод тріангуляції.

Тріангуляція 3D моделей - це процес перетворення поверхні моделі з полігонів складених з чотирикутників (або інших форм) у полігональну мережу, де кожен полігон має лише три вершини (трикутник). Цей процес необхідний для багатьох графічних програм і ігор, оскільки багато алгоритмів і обчислень, пов'язаних з візуалізацією та обробкою 3D моделей, працюють саме з трикутниками.

Трикутники є простими та ефективними геометричними об'єктами для обробки в 3D графіці. Вони легко опрацьовуються графічним апаратом комп'ютера, а також дозволяють використовувати різні алгоритми рендерингу, освітлення та взаємодії з об'єктами.

Тріангуляція 3D моделей може бути реалізована за допомогою різних математичних методів. Один з поширених методів тріангуляції, який використовується в Blender, - це алгоритм Делоне. Алгоритм Делоне забезпечує тріангуляцію таким чином, щоб кожний трикутник утворював опуклу оболонку

вершин моделі, тобто жодна вершина не знаходилася в середині кола, описаного навколо будь-якого трикутника[3] (рис. 2.1).

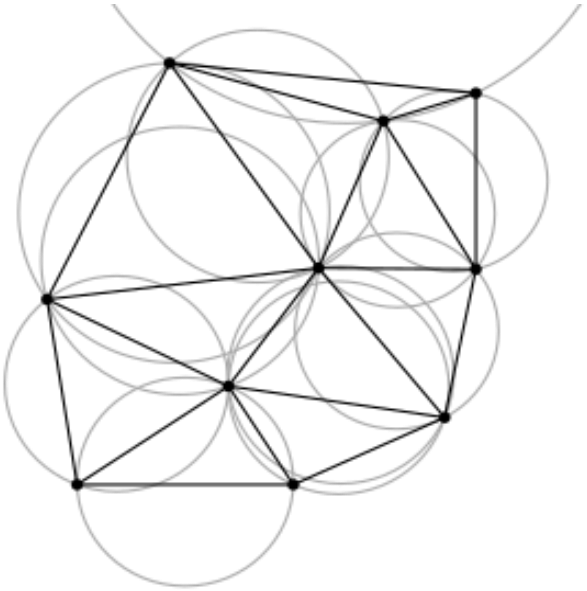

Рис. 2.1. Тріангуляція Делоне і описані навколо трикутників кола

Багато алгоритмів для обчислення тріангуляції Делоне спираються на швидкі операції для визначення того, чи точка знаходиться всередині описаного навколо трикутника кола, і ефективних структур даних для зберігання трикутників та ребер. В двовимірному випадку, якщо D знаходиться всередині кола описаного навколо A, B, C треба перевірити чи визначник більше 0, як показано на рисунку *.

$$\begin{vmatrix} A_x & A_y, & A_x^2 + A_y^2, & 1 \\ B_x & B_y, & B_x^2 + B_y^2, & 1 \\ C_x & C_y, & C_x^2 + C_y^2, & 1 \\ D_x & D_y, & D_x^2 + D_y^2, & 1 \end{vmatrix} = \begin{vmatrix} A_x - D_x, & A_y - D_y, & (A_x^2 - D_x^2) + (A_y^2 - D_y^2) \\ B_x - D_x, & B_y - D_y, & (B_x^2 - D_x^2) + (B_y^2 - D_y^2) \\ C_x - D_x, & C_y - D_y, & (C_x^2 - D_x^2) + (C_y^2 - D_y^2) \end{vmatrix} > 0$$

Рис. 2.2. Перевірка визначника

Для множини P точок d-вимірного Евклідового простору, тріангуляція Делоне це тріангуляція DT(P) така що жодна з точок з P не лежить всередині гіперсфери описаної навколо будь-якого симплекса з DT(P). Відомо що існує

єдина тріангуляція Делоне для P, якщо P множина точок в загальній позиції; така що не існує підпростору розмірності k що містить k+2 точок ні k-сфери що містить k+3 точок, для $1 \leq k \leq -1$ (тобто, наприклад для множини точок в $\mathbb{R}^3$ не існує трьох точок що лежать на одній прямій, не існує чотирьох що лежать в одній площині, і жодні п'ять точок не лежать на одній сфері). Задача знаходження тріангуляції Делоне для множини точок в d-вимірному просторі може бути зведена до задачі знаходження опуклої оболонки множини точок в (d+1)-вимірному просторі, додаючи кожній точці p додаткову координату яка дорівнює $|p|^2$, беручи нижню частину опуклої оболонки, і відображаючи її назад в d-вимірний простір видаленням останньої координати. Так як опукла оболонка єдина, то тріангуляція теж, вважаючи, що всі сегменти опуклої оболонки — симплекси. Не симплексні сегменти з'являються лише коли d+2 різних точок лежить на одній d-гіперсфері, тобто точки не знаходяться в загальній позиції.

Процес тріангуляції може бути автоматичним або здійснюватися за допомогою ручних налаштувань. У Blender можна використовувати автоматичну тріангуляцію для створення трикутникової мережі з полігонів, але також є можливість редагувати тріангуляцію вручну, перетворюючи полігони в трикутники за допомогою різних інструментів.

Ручне редагування тріангуляції може бути корисним, коли важливо забезпечити правильність геометрії моделі, зберігаючи оптимальну топологію. Неправильна тріангуляція може призводити до артефактів рендерингу, проблем з текстуруванням, а також ускладнювати процес анімації та моделювання.

Узагалі, тріангуляція є важливою складовою обробки 3D моделей в графічних програмах та іграх. Вона дозволяє ефективно працювати з геометрією моделей і забезпечує правильне відображення об'єктів у візуалізації 3D-сцени.

2.3. Опис використаних технологій та мов програмування

Blender — об'єктно-орієнтована програма для створення тривимірної комп'ютерної графіки. Це не тільки моделювання, але і анімація, рендеринг, створення ігор, обробка відеоматеріалів.

Даний пакет інструментів поширюється по безкоштовній ліцензії, тому він встає у нагоді багатьом інді-розробникам і навіть фірмам. Нова версія вже чекає вас в каталозі.

Blender 3D систематично оновлюється групою розробників і постійно розвивається. Цьому він і зобов'язаний своєю популярністю. Крім того, вага дистрибутива важить всього вісім мегабайт![4]

Unity - це популярний ігровий двигун, що використовується для розробки ігор, симуляцій, візуалізацій та інтерактивних додатків. Він надає потужні інструменти і середовище розробки, які дозволяють розробникам створювати як 2D, так і 3D проекти для різних платформ, включаючи комп'ютери, консолі, мобільні пристрої та віртуальну реальність.

Основні риси і можливості Unity:

- Мультиплатформеність: Unity підтримує розробку ігор для різних платформ, таких як Windows, macOS, iOS, Android, Xbox, PlayStation і багатьох інших. Це дозволяє розробникам зробити свої ігри доступними для широкої аудиторії.
- Графічний двигун: Unity має потужний графічний двигун, який надає велику свободу в створенні стилізованих 2D і 3D візуальних ефектів. Він підтримує рендеринг у реальному часі, освітлення, тіні, частинки, анімацію та багато інших можливостей для створення вражаючого візуального досвіду.
- Інтегроване середовище розробки: Unity надає зручне інтегроване середовище розробки (IDE), що спрощує процес розробки ігор. Воно має візуальний редактор, систему управління активами, інструменти для редагування сцен та анімацій, систему фізики, скриптування за допомогою мови C# та інші зручні функції.

- Компонентна архітектура: Unity базується на компонентній архітектурі, де функціональність об'єктів визначається компонентами, що можуть бути додані та налаштовані розробником. Це дозволяє легко розширювати функціональність об'єктів і взаємодіяти з ними.

- Магазин активів: Unity має великий магазин активів, де розробники можуть придбати готові моделі, текстури, анімації, звуки та інші ресурси для використання у своїх проектах. Це спрощує розробку і дозволяє економити час і зусилля.

Unity широко використовується як незалежними розробниками, так і великими студіями для створення різноманітних ігор і додатків. Він надає потужні інструменти та ресурси, що допомагають зробити процес розробки більш ефективним та доступним[5].

Substance Painter - це потужний програмний інструмент для текстурування і малювання 3D моделей. Він розроблений компанією Allegorithmic (придбаною Adobe у 2019 році) і використовується в галузі відеоігор, візуалізації, анімації та інших областях 3D-графіки.

Substance Painter надає широкі можливості для створення високоякісних текстур і матеріалів, які можуть бути застосовані на 3D моделях[6].

ZBrush – програма для 3D-моделювання, створена компанією «Pixologic». Відмінною особливістю цього ПЗ є імітація процесу «ліплення» тривимірної скульптури, посиленого двигуном тривимірного рендерингу в реальному часі, що спрощує процедуру створення необхідного тривимірного об’єкта. Кожна точка (звана піксель) містить інформацію не тільки про свої координати XY та значення кольору, але також і глибину Z, орієнтацію та матеріал. Це означає, що можна не тільки «ліпити» тривимірний об’єкт, але і «розфарбувати» його, малюючи штрихами з глибиною. Реалістичне відображення тіней та відблисків, є автоматизованим в ZBrush. Також швидко працює зі стандартними 3D-об’єктами, використовуючи пензлі для модифікації геометрії матеріалів та текстур. Дозволяє досягти інтерактивності при великій кількості полігонів. Використовуючи спеціальні методи, можна підняти деталізацію до десятків

мільйонів полігонів. Також є безліч модулів, що підключаються (робота з текстурами, геометрією, безліч нових пензлів, швидка інтеграція з професійними пакетами 2D-графіки і багато іншого)[7].

Мова програмування C# (C-Sharp) є потужною і популярною мовою програмування, розробленою компанією Microsoft. Вона використовується для створення різноманітних додатків, включаючи веб-додатки, настільні програми, серверні додатки та ігри. C# є частиною технологічної платформи .NET, яка надає широкі можливості для розробки програмного забезпечення.

У контексті Unity, C# є основною мовою програмування для розробки ігор. Unity надає розробникам можливість створювати скрипти та логіку гри за допомогою C#. Це означає, що можна написати програмний код, який контролюватиме поведінку об'єктів, реалізовуватиме фізику, оброблятиме взаємодію гравця та багато іншого.

2.4. Опис структури програми та алгоритмів її функціонування

Для опису сценаріїв використання користувачем програмного продукту та взаємозв’язків у самому додатку було створену наступні UML діаграми.

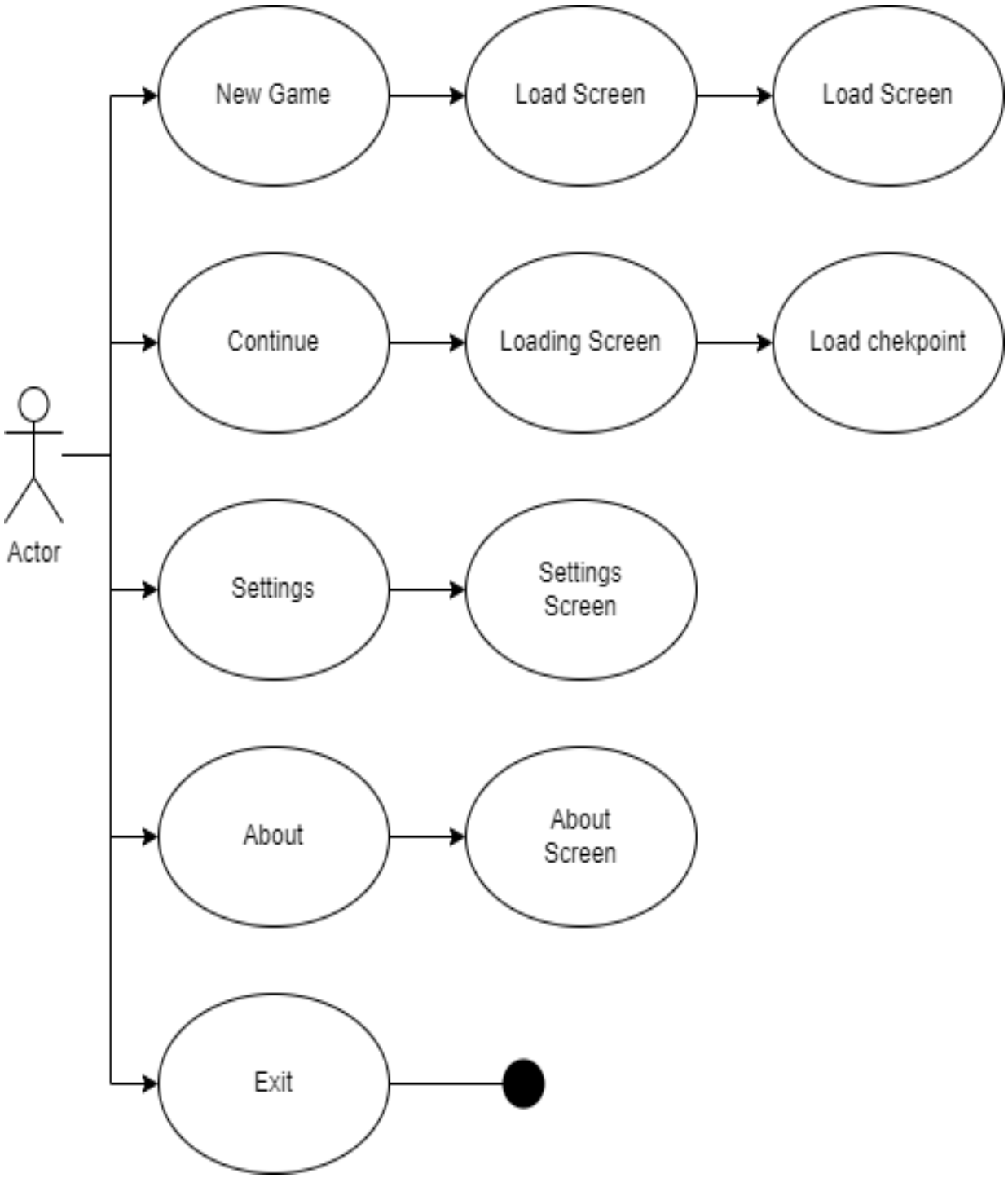

Рис. 2.3. UML діаграма роботи головного меню гри

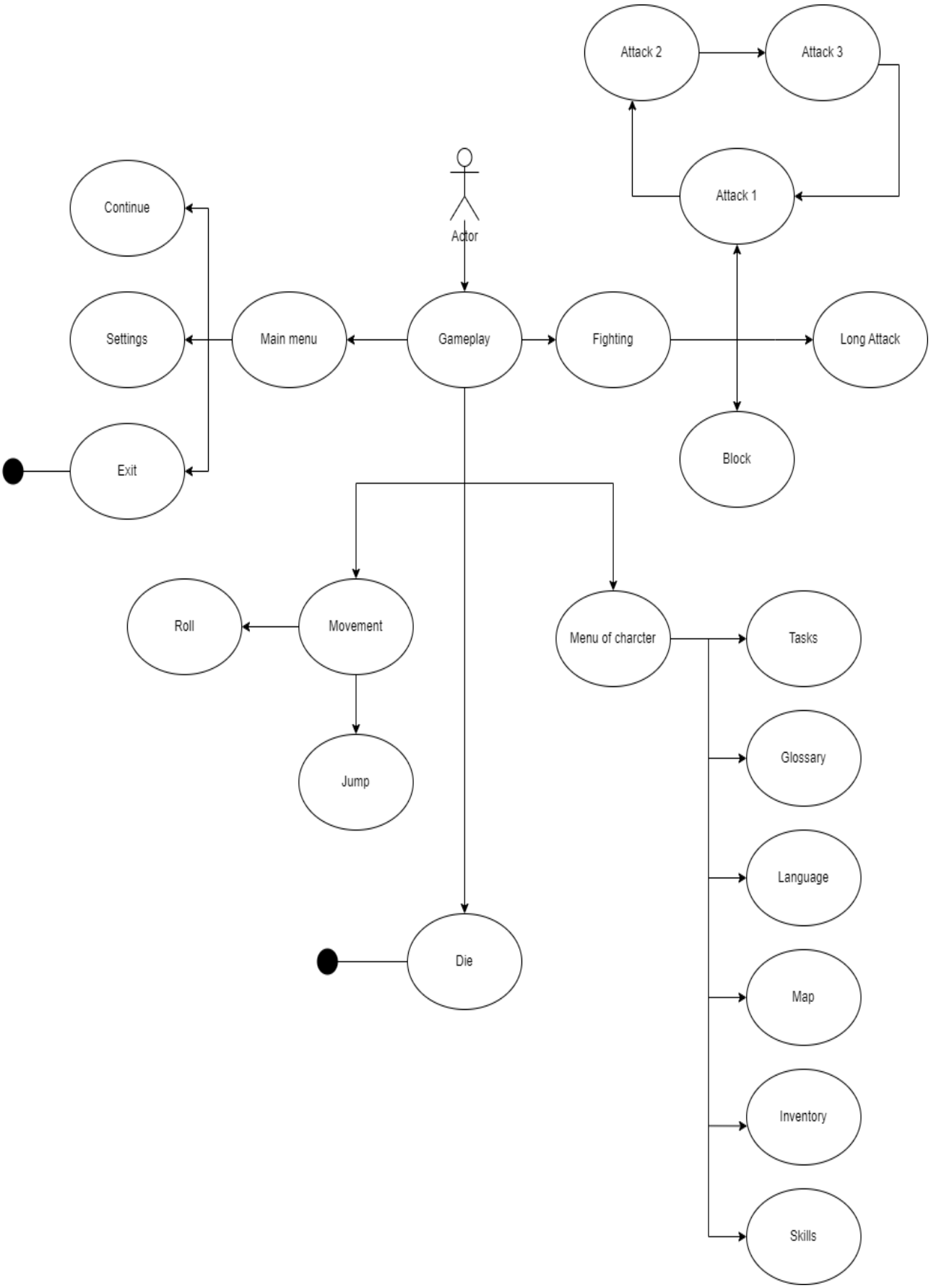

Рис. 2.4. UML діаграма основного ігрового процесу

Дуже цікавою особливість проекту, є взаємодія гравця з ворогами. Вороги рухаються за допомогою вбудованої функції Unity, яка дозволяє налаштувати штучний інтелект. Та розрахувати карту “вільного місця” по якому може рухатися ворог, в режимі патрулювання.

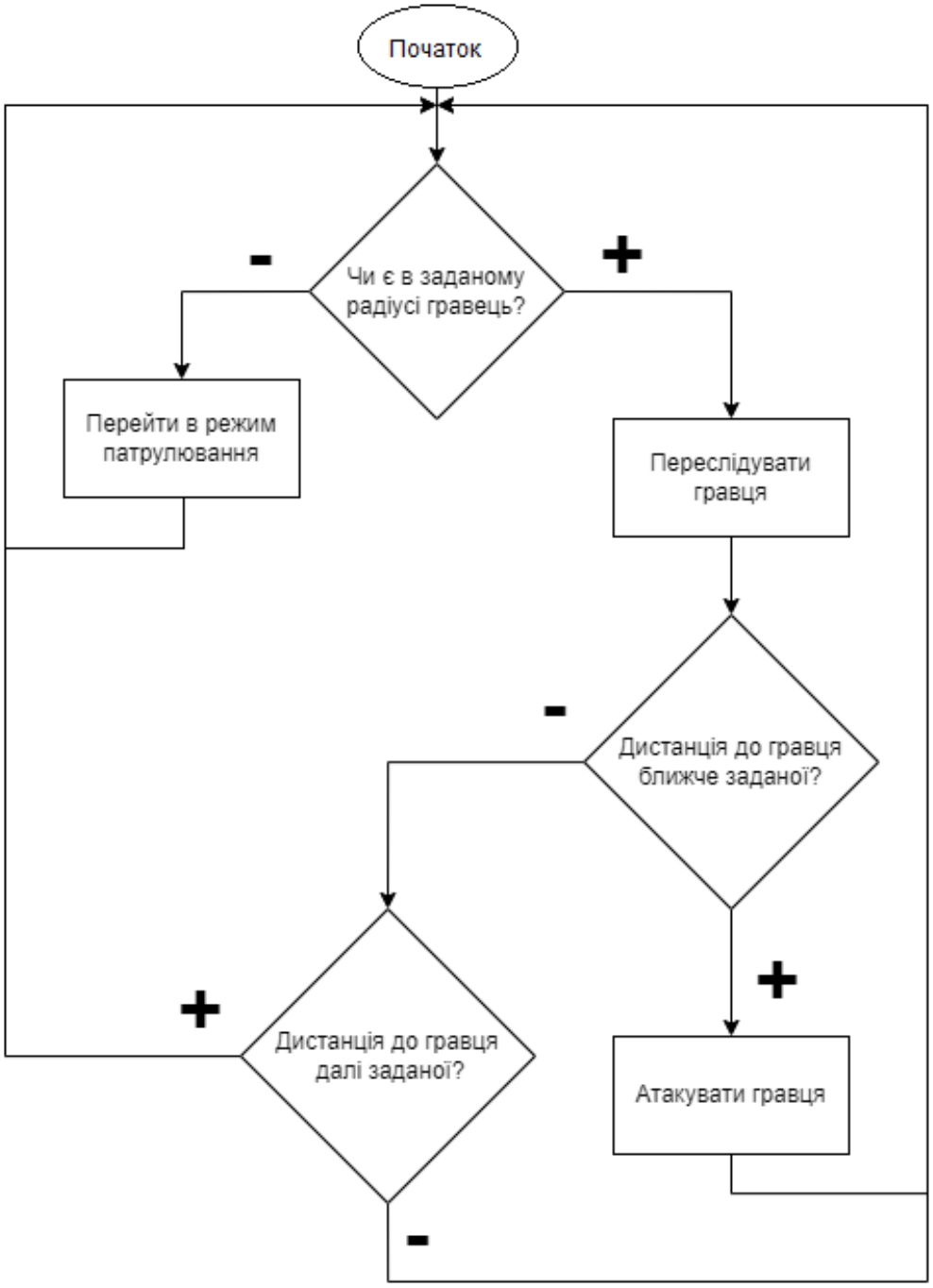

Рис. 2.5. Блок схема поведінки ворога

На рисунку 2.6, зображено вікно налаштування запікання карти вільного простору для руху ворога. Де, 1 – ворог, 2 – місця не виділені блакитним кольором – місця які повинен оминати ворог, при патрулюванні або переслідуванні гравця, 3 – вікно налаштування геометричної форми ворога та об’єктів які він повинен оминати, 4 – блакитний колір, карта поверхні по якій може вільно пересуватися ворог.

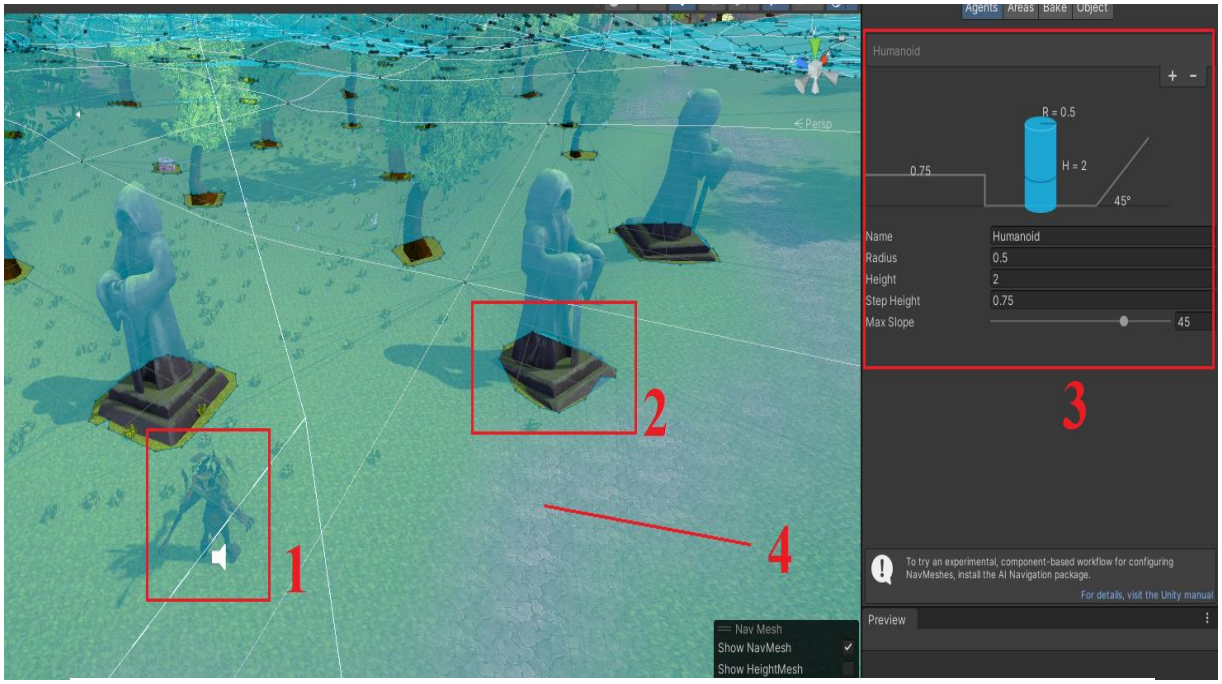

Рис. 2.6. Вікно налаштування вільного простору

Слід зазначити, що ворог рухається випадково до любої з заданої точки, враховуючи свою карту для пересування. Рух, ворога не є лінійним і реалізований за допомогою штучного інтелекту вбудованого в спеціальні функції Unity. Це дозволяє зробити ворога більш “розумним” та уникнути проблем при переслідуванні гравця, або застряганні в об’єктах.

За схожим принципом працюють і NPC, але без режимів переслідування та атаки.

В грі також реалізовані скрипти-контролери, які дозволяють зробити працюючий HUD гравця. Ці скрипти взаємодіють з ігровим простором, гравцем та часом і дозволяють виводити на екран користувача інформацію у вигляді барів. На рисунку 2.7, зазначений принцип роботи цих скриптів.

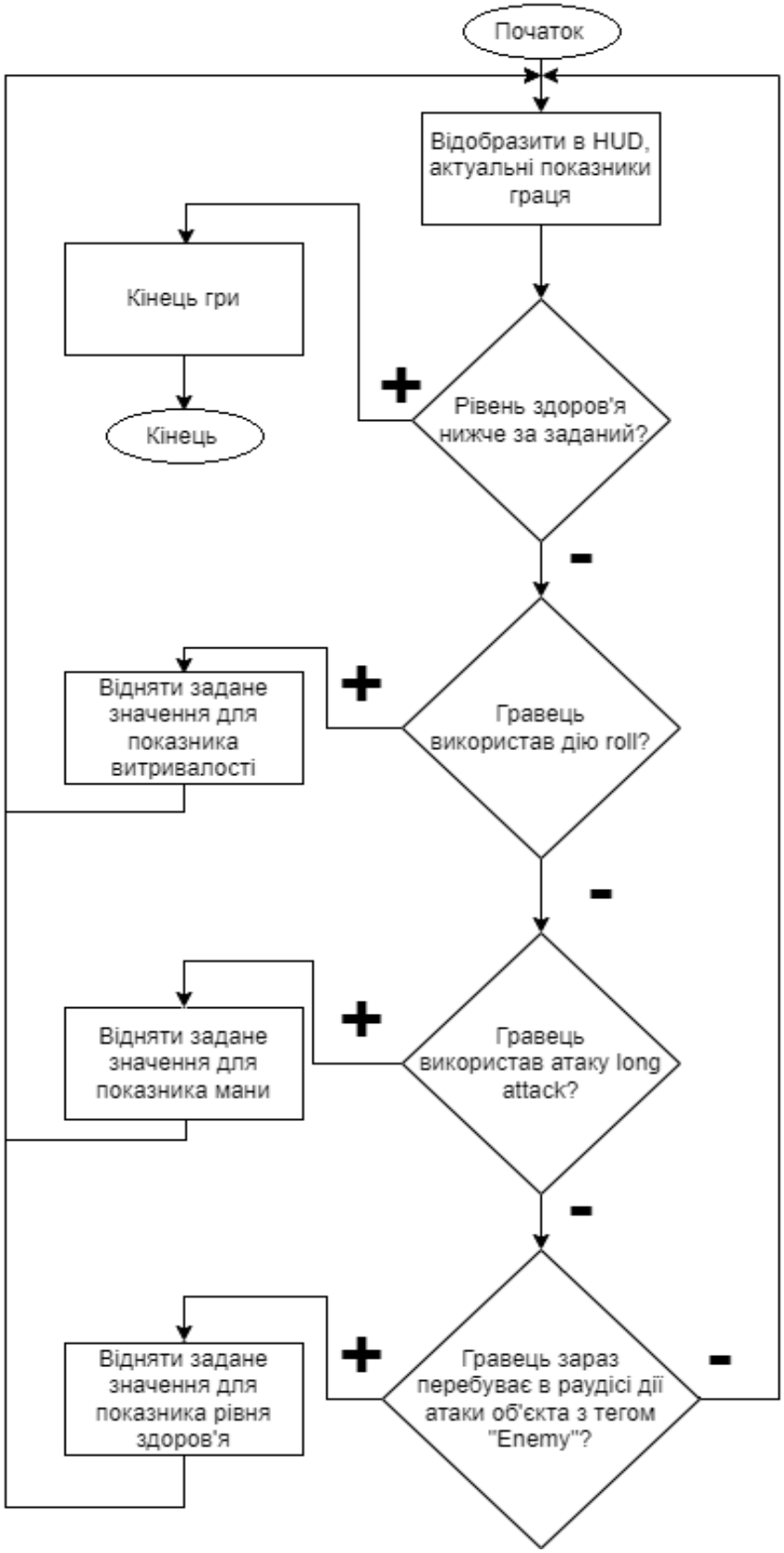

Рис. 2.7. Блок-схема роботи HUD гравця

Рис. 2.8. Зовнішній вигляд HUD гравця

Також в цьому застосунку реалізовані інші скрипти:

- контролер рівня здоров’я ворога – який керує здоров’ям ворога та відображенням його в інтерфейс користувача;
- контролер керування руху, анімацій персонажа;
- скрипти які відповідають за взаємодію з ігровим меню гравця;
- скрипт який за допомогою системи координат відображає поточне перебування гравця на мінікарті;
- скрипт який реалізовує атаку ворога. (Створює окремі об’єкти в сцені та зазначає радіус дії атаки);
- скрипт який керує асинхронною системою завантаження рівня для екранів завантаження;
- скрипти які реалізовують переходи між локаціями;

Коди роботи цих скриптів зазначено в лістингу.

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Даний програмний додаток приймає ввід користувача для управління головним героєм та елементами інтерфейсу. В якості вихідних даних програма повертає файл з ігровим прогресом та особистими налаштуваннями користувача.

2.6 Опис розробленого програмного продукту

2.6.1 Використані технічні засоби

Для розробки була використана електронно-обчислювальна машина (комп’ютер) з нижченаведеними характеристиками:

- OS: Windows 10
- CPU: AMD Ryzen 3 1200 (2.4 ГГц)
- GPU: Nvidia GeForce GTX 1050Ti
- Memory: SSS 512 Gb
- RAM: 16 Gb;
- Screen: 1920x1080.

А також периферійні засоби, які використовувалися на проміжних етапах проектування: у якості графічного планшету ChromeBook Lenovo 300e.

2.6.2. Використані програмні засоби

Blender - це безкоштовне і відкрите програмне забезпечення для 3D-моделювання, анімації, візуалізації і комп'ютерної графіки. Воно широко використовується в ігровій індустрії завдяки своїм багатим функціональним можливостям та гнучкості.

Одна з ключових переваг Blender - це його безкоштовна доступність. В порівнянні з комерційними програмами, які вимагають значних витрат на ліцензії, Blender є безкоштовним для використання всіма. Це робить його привабливим вибором для незалежних розробників, стартапів та невеликих команд, що працюють над ігровими проектами з обмеженим бюджетом.

Blender також володіє потужними інструментами для 3D-моделювання. Він має широкі можливості створення складних геометричних форм, текстур і матеріалів, а також підтримує багато форматів файлів для імпорту і експорту. Це дозволяє розробникам створювати реалістичні ігрові об'єкти та світи.

У Blender також вбудовані інструменти для анімації. Розробники можуть створювати складні рухи персонажів, анімовані ефекти і взаємодіючі анімаційні

сцени. Завдяки системі скелетного анімування і вагової покладки, Blender дозволяє створювати реалістичні персонажі зі зручним керуванням рухами[8].

Інтерфейс програми наведено на рисунку 2.9.

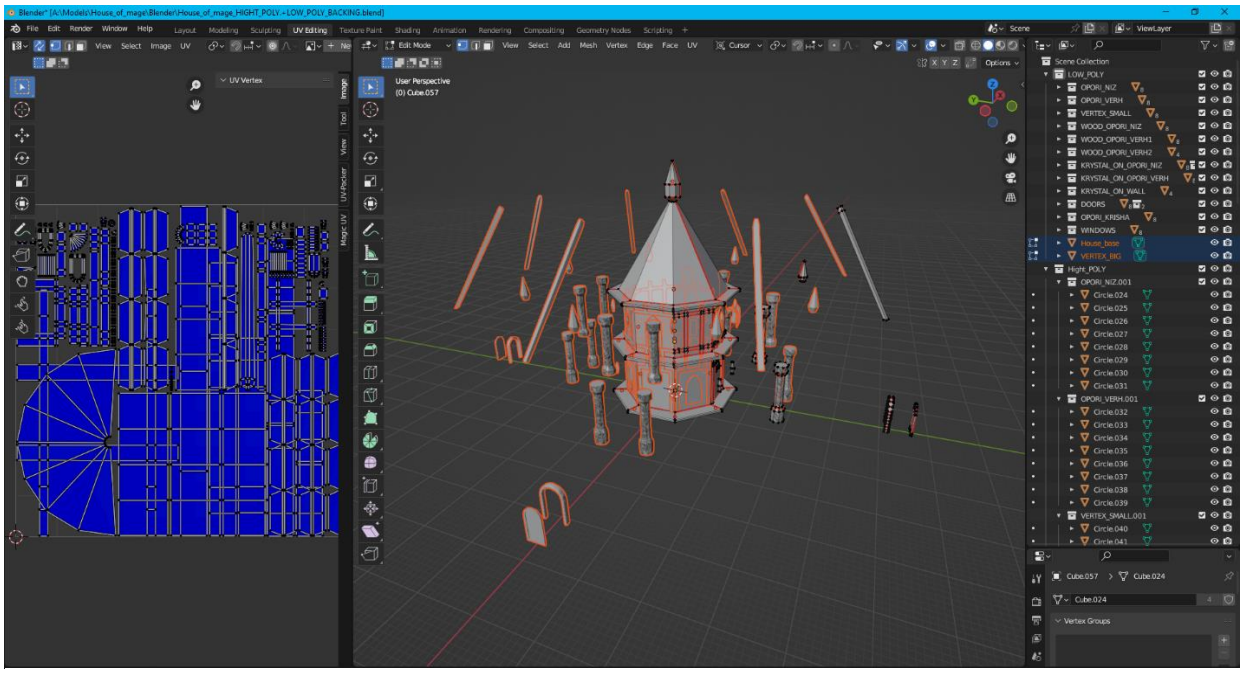

Рис. 2.9. Інтерфейс програми Blender, версії 3.4

Unity - це популярне інтегроване середовище розробки програмного забезпечення, яке використовується для створення ігор, віртуальної реальності, доповненої реальності та інших інтерактивних додатків. Воно володіє численними перевагами, які зробили його популярним у галузі геймдеву.

Unity має простий інтерфейс та потужні інструменти для розробки ігор, що робить його доступним для розробників з будь-яким рівнем досвіду. Він підтримує багато платформ, включаючи ПК, консолі, мобільні пристрої та віртуальну реальність, що дозволяє створювати гру один раз і розгортати її на різних пристроях.

Unity має велику спільноту розробників, що надає значний обсяг документації, уроків і підтримки. Розробники можуть знайти відповіді на свої питання, вирішити проблеми і отримати поради від інших членів спільноти, що допомагає прискорити процес розробки і вдосконалення гри.

Unity має гнучку систему компонентів, яка дозволяє розробникам легко додавати, налаштовувати та управляти функціональністю об'єктів в грі. Це спрощує процес створення складних ігрових об'єктів і систем.

Unity також підтримує широкий спектр мов програмування, таких як C#, JavaScript та Boo. Це дає розробникам можливість вибирати мову, з якою вони найбільше знайомі і комфортні для розробки своїх ігрових проектів.

Ще однією з переваг Unity є його інтеграція з багатьма зовнішніми програмами і сервісами, такими як фізичні двигуни, аналітика, соціальні мережі та інші. Це дає розробникам можливість легко інтегрувати додатковий функціонал у свої ігри та отримувати доступ до різних сервісів, що поліпшує користувацький досвід.

Загалом, Unity є потужним інструментом для розробки ігор, який забезпечує широкі можливості, гнучкість і підтримку спільноти, що робить його популярним серед розробників у галузі геймдеву.

Інтерфейс програми наведено на рисунку 2.10.

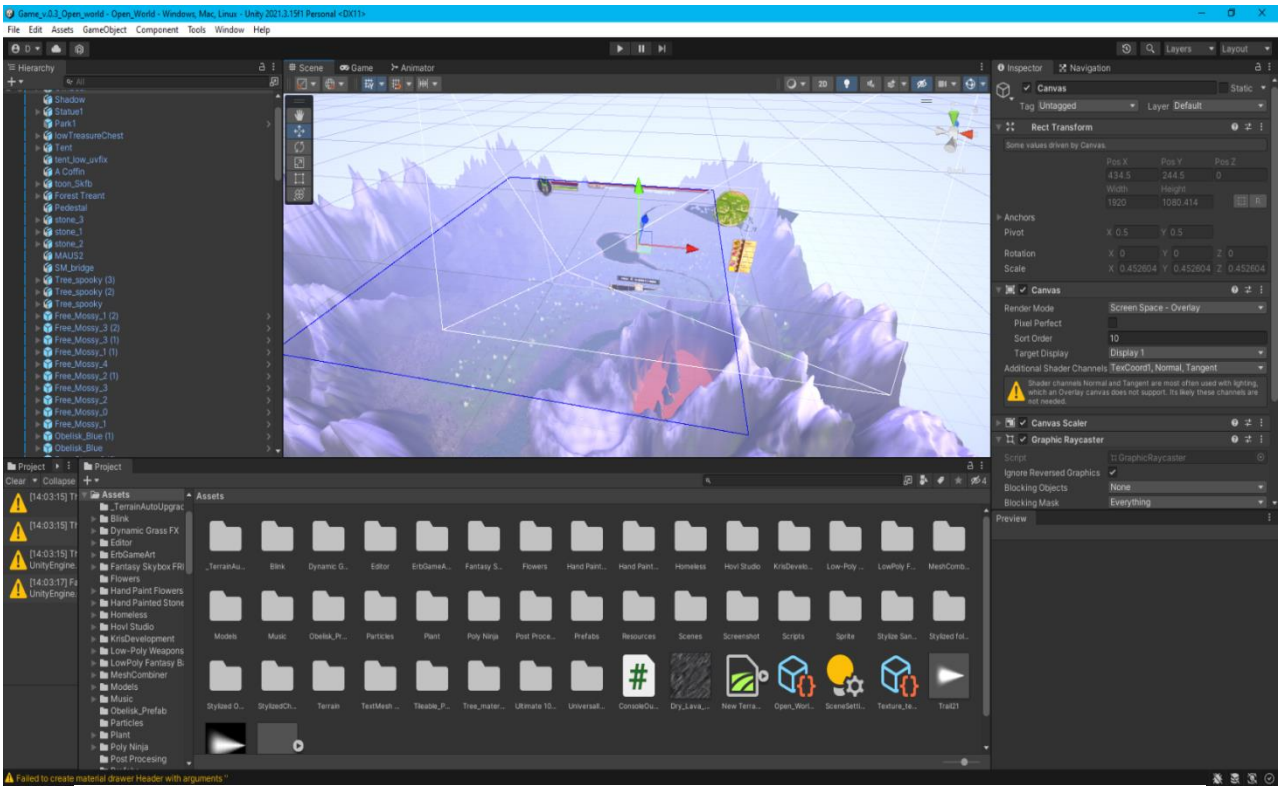

Рис. 2.10. Інтерфейс програми Unity, версія 2021.3.15f1

Substance Painter є програмним забезпеченням для текстурування і малювання 3D-моделей, спеціально розробленим для ігрової індустрії. Вона володіє численними перевагами, які роблять її незамінним інструментом під час розробки 3D-моделей для ігор.

Одна з основних переваг Substance Painter полягає у її потужних можливостях текстурування. Вона надає широкий набір інструментів і бібліотек матеріалів, що дозволяють розробникам створювати реалістичні текстури для об'єктів у грі. Завдяки розширеній системі шарів і масок, розробники можуть легко створювати складні ефекти і деталізацію.

Substance Painter також володіє потужними інструментами для редакції матеріалів і текстур. Вона підтримує процедурне текстурування, що дозволяє створювати динамічні матеріали з параметрами, які можна налаштовувати. Це дозволяє швидко змінювати вигляд і властивості матеріалів без необхідності редагування самої текстури.

Іншою перевагою Substance Painter є його висока швидкість роботи та оптимізація. Вона ефективно використовує ресурси комп'ютера і дозволяє розробникам швидко і легко малювати, редагувати та наносити текстури на 3D-моделі. Це особливо важливо при роботі з великими проектами, де продуктивність є ключовим фактором.

Substance Painter також підтримує різні формати текстур і має вбудовану підтримку важливих двигунів гри, таких як Unity та Unreal Engine. Це спрощує інтеграцію з іншими інструментами розробки та полегшує обмін текстурами та матеріалами між різними програмами.

Крім того, Substance Painter забезпечує велику кількість готових матеріалів і текстурних ресурсів, які можна використовувати для прискорення процесу розробки. Вона також має активну спільноту користувачів, яка ділиться ресурсами, уроками та порадами, що полегшує вивчення програми та покращує творчий обмін ідеями.

Загалом, Substance Painter є потужним інструментом для текстурування 3D-моделей у галузі геймдеву. Вона надає розробникам широкі можливості для

створення реалістичних текстур, швидкого редагування та оптимізації робочого процесу, що допомагає прискорити процес розробки і підвищити якість гри[9]. Інтерфейс програми наведено на рисунку 2.11.

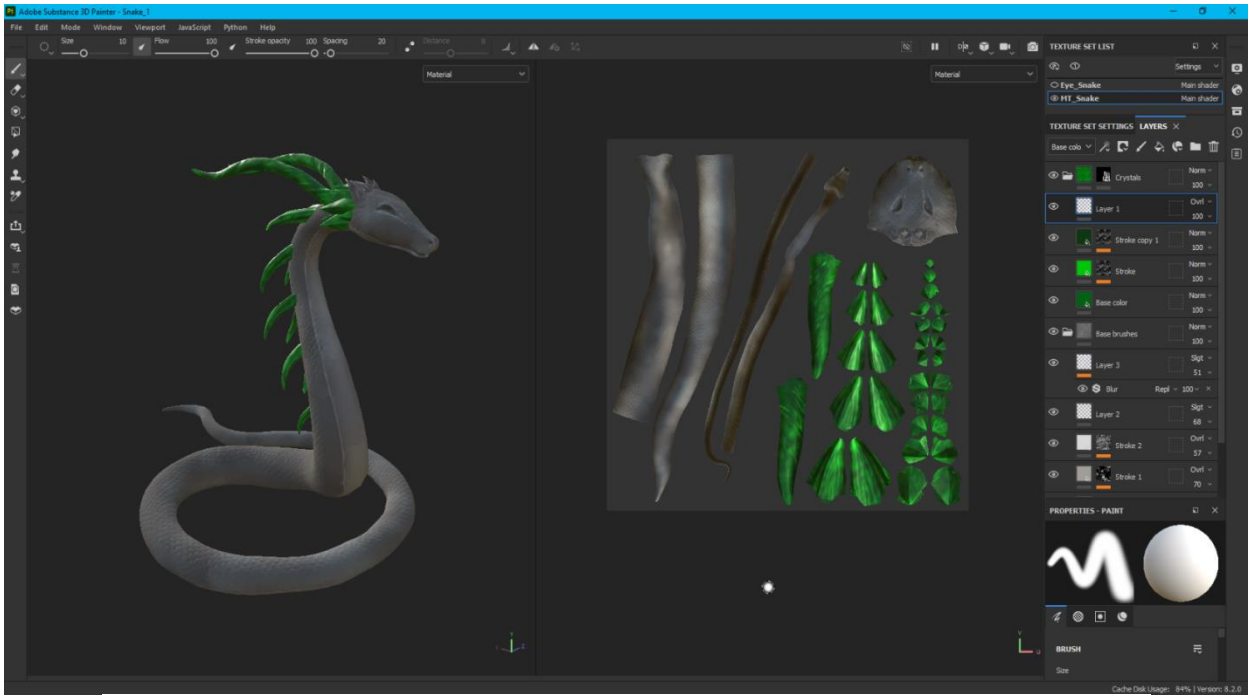

Рис. 2.11. Інтерфейс програми Substance 3D Painter версії 8.3.1

ZBrush - це програмне забезпечення для цифрового моделювання та скульптури 3D-об'єктів. Його використовують розробники ігор з різних причин, оскільки воно має декілька важливих переваг у галузі геймдеву.

Одна з ключових переваг ZBrush - це його потужність і реалістичність в моделюванні. Воно надає розробникам можливість створювати деталізовані, високоякісні 3D-моделі з великою кількістю полігонів. Завдяки передовим алгоритмам скульптури, ZBrush дозволяє деталізувати моделі до мікрорівня, що робить його ідеальним для створення складних ігрових персонажів та об'єктів.

Ще одна перевага ZBrush полягає у його інтуїтивному інтерфейсі та потужних інструментах. Воно має широкі можливості для формування, текстурування, розмальовування та деталізації моделей. Розробники можуть використовувати пензлі, кисті, маски, різні типи матеріалів і багато інших інструментів для досягнення бажаного результату.

ZBrush також володіє потужними функціями для ретопології моделей, що дозволяє оптимізувати їх для ігрових двигунів. Розробники можуть створювати ефективнішу геометрію, зменшуючи кількість полігонів і підготовлюючи моделі для оптимальної продуктивності у грі.

ZBrush також підтримує високу деталізацію текстур, включаючи нормальні карти, дисплейсмент-карти, амбієнтні окулюзії та інші. Це дозволяє розробникам створювати реалістичні, живі та деталізовані текстури для своїх ігрових об'єктів.

Крім того, ZBrush має активну спільноту користувачів, яка надає велику кількість ресурсів, уроків та підтримку. Розробники можуть отримати поради, навчитися новим технікам та ділитися своїми досягненнями з іншими фахівцями в галузі.

Усе це робить ZBrush незамінним інструментом для розробників ігор, які потребують високоякісних, деталізованих ігрових моделей та текстур. Він сприяє творчому процесу, полегшує роботу з 3D-об'єктами та забезпечує високу якість готових продуктів[10]. Інтерфейс програми наведений на рисунку 2.12.

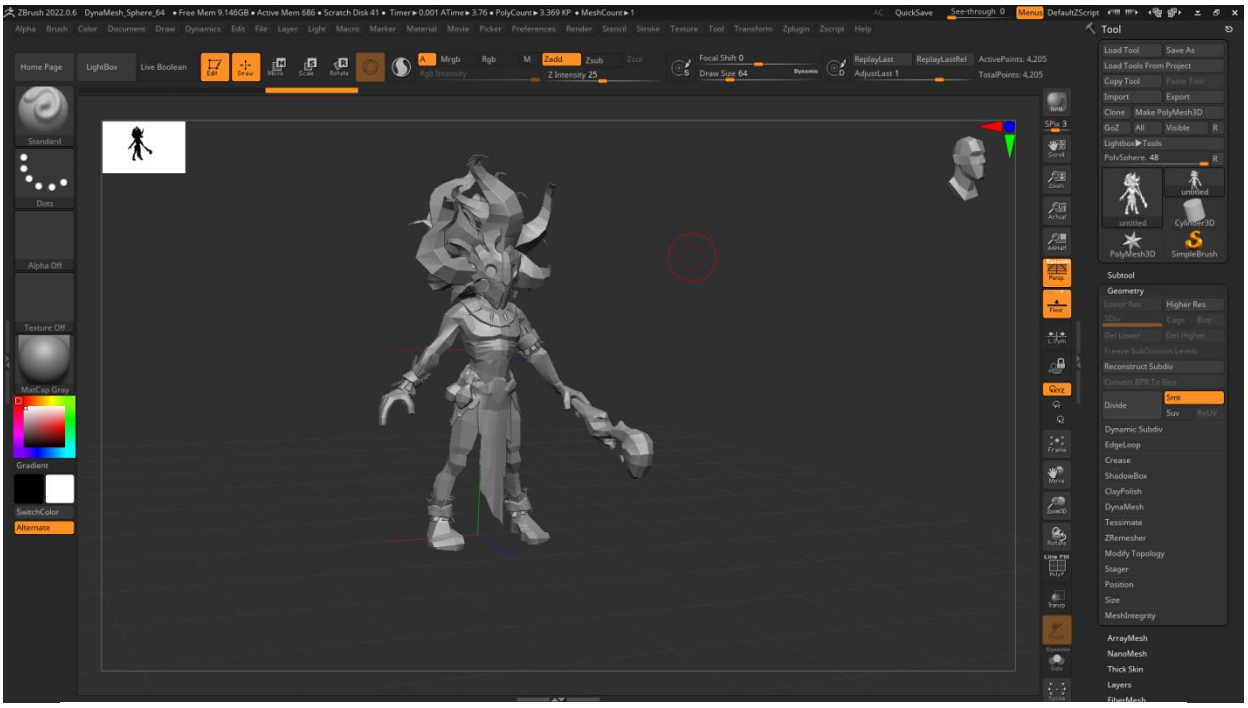

Рис. 2.12. Інтерфейс програми ZBrush, версії 2022.0.6

Photoshop є програмним забезпеченням для редакції та обробки графічних зображень. Він широко використовується в індустрії геймдеву через свої багатофункціональність та набір інструментів, що надають численні переваги для розробки ігор.

Одна з основних переваг Photoshop - це його потужність у роботі з растровою графікою. Він дозволяє розробникам створювати та редагувати текстури, спрайти, іконки та інші графічні елементи, необхідні для ігор. Завдяки багатому набору інструментів для малювання, виправлення, ретуші та фільтрації зображень, Photoshop дозволяє створювати високоякісну графіку з великою деталізацією.

Іншою перевагою Photoshop є його інтеграція з іншими програмами та інструментами. Photoshop також має можливість роботи з шаровими файлами, що спрощує обробку окремих елементів та створення комплексних комбінацій зображень.

Photoshop також володіє потужними інструментами для створення спеціальних ефектів і анімації. Це дозволяє покращити візуальний досвід гравців та створити захоплюючі ефекти в грі[11]. Інтерфейс програми наведено на рисунку 2.13.

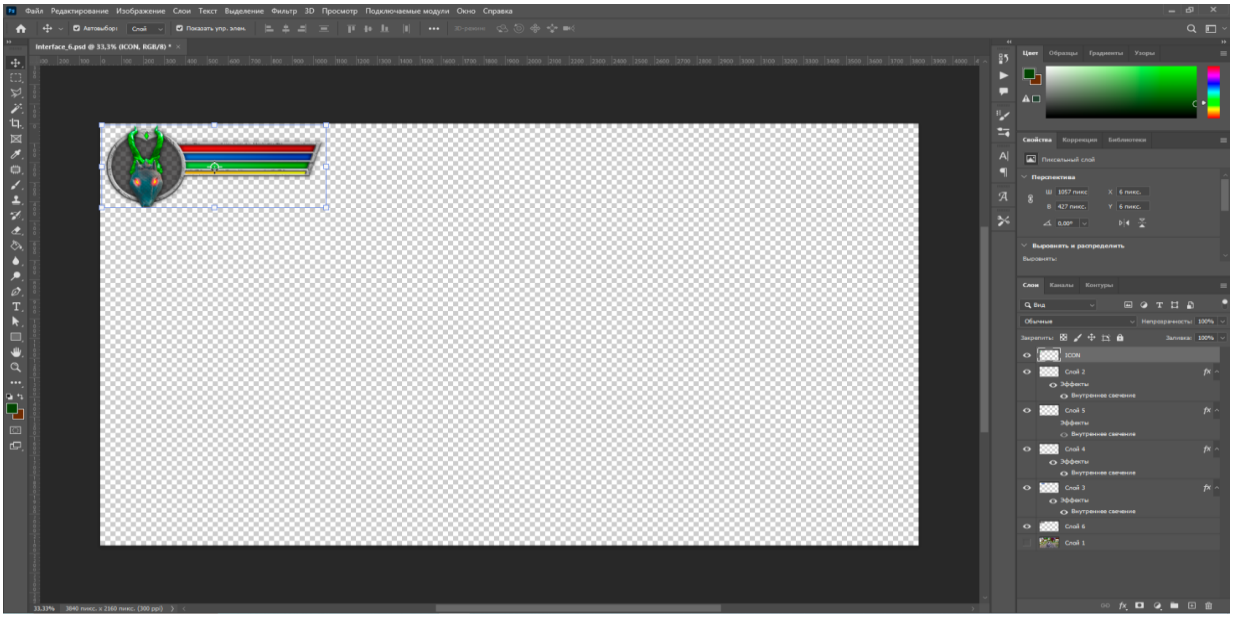

Рис. 2.13. Інтерфейс програми Photoshop версія 2022.

Редактор коду Atom - це безкоштовний, відкритий текстовий редактор, розроблений компанією GitHub. Він має широкі можливості налаштування, розширення та підтримує багато мов програмування, включаючи C#, яку використовує Unity для написання скриптів.

Одна з основних переваг Atom - це його розширюваність. Він має велику кількість плагінів та розширень, що дозволяють розробникам налаштувати редактор під свої потреби. У випадку роботи з Unity, розробники можуть встановити плагіни, які надають підсвічування синтаксису, автодоповнення, відлагодження та інші корисні функції спеціально для роботи з Unity-скриптами.

Іншою перевагою Atom є його легкість використання та інтуїтивний інтерфейс. Він надає зручну і зрозумілу робочу область, яка полегшує написання, редагування та огляд коду

Крім того, Atom має вбудовану систему контролю версій, що дозволяє розробникам взаємодіяти з репозиторіями Git безпосередньо з редактора. Це спрощує роботу зі змінами, комітами, гілками та іншими аспектами керування версіями проекту.

У порівнянні з Visual Studio, Atom може бути більш легким та швидшим редактором, особливо при роботі зі скриптами Unity, якщо використовуються лише основні функціональні можливості. Він також має менше вимоги до ресурсів комп'ютера, що може бути важливим при роботі з обсягом коду Unity-проектів[12]. Суб’єктивно для мене редактор коду ATOM став більш ліпшим рішенням, саме завдяки його швидкодії, коли постійно треба закривати-відкривати скрипти – це займає менше часу, в порівнянні з тим же Visual Studio. Інтерфейс програми зображений на рисунку 2.14.

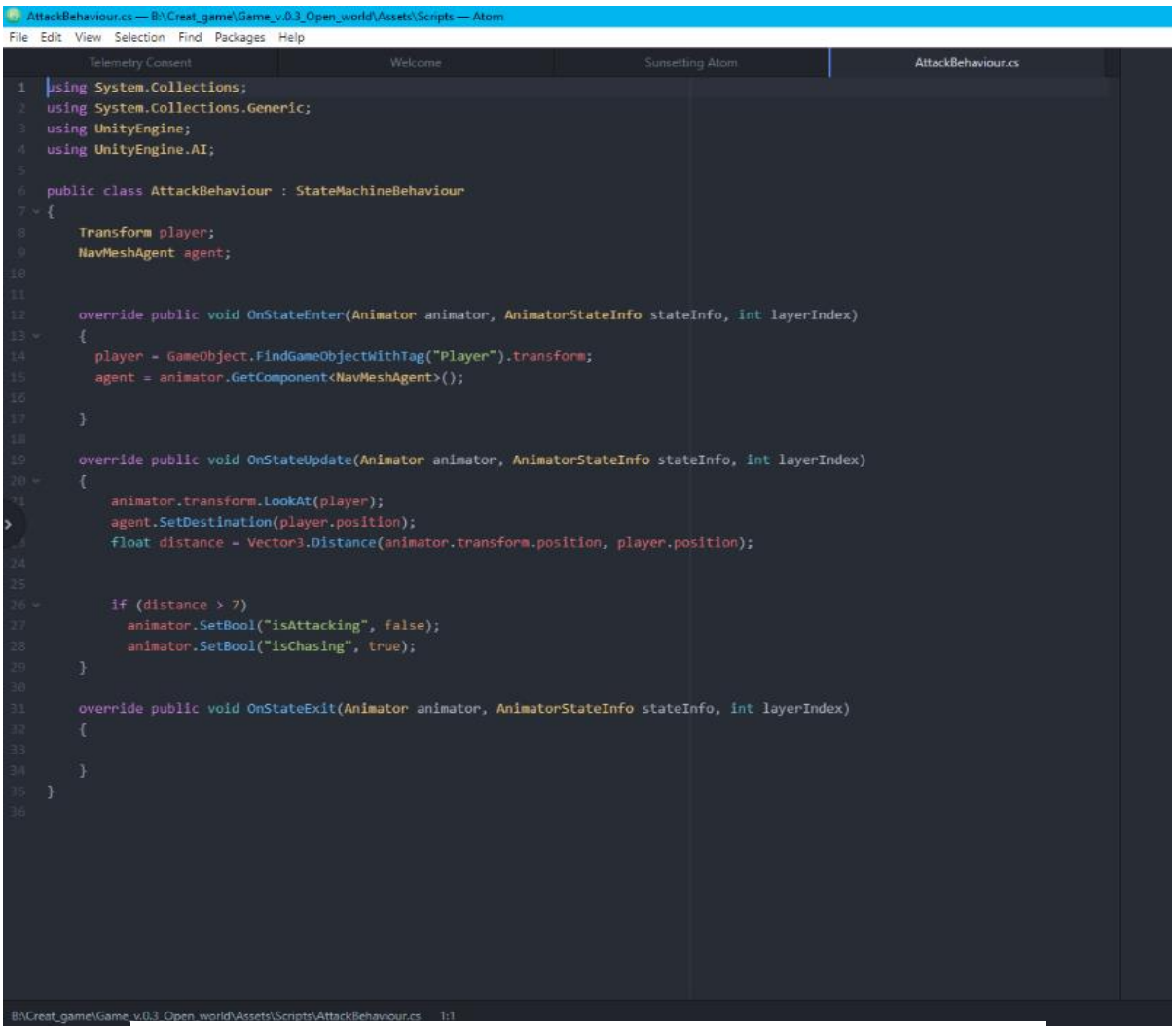

Рис. 2.14. Інтерфейс програми Atom

Inno Setup є безкоштовним, відкритим та потужним інструментом для створення інсталяційних програм для Windows. Він дозволяє розробникам створювати професійні інсталяційні пакети, які об'єднують файли програми в один компактний інсталятор.

Однією з ключових переваг Inno Setup є його простота використання. Він має зрозумілий синтаксис скриптів і простий інтерфейс, що дозволяє швидко налаштувати інсталяційний процес. Є можливим вказати шляхи файлів, створювати ярлики, налаштовувати реєстрацію, обрані компоненти та багато іншого, все це використовуючи зрозумілу скриптовий мову. Однією з найбільших переваг запаковування файлів в один інсталятор є спрощення процесу розповсюдження вашої програми. Можна легко створити один файл,

який містить усі необхідні файли, ресурси та скрипти для встановлення програми на комп'ютер користувача. Це полегшує поширення програми, забезпечує однорідний інсталяційний процес і зменшує можливість пропуску або помилок під час розпакування та встановлення. Інтерфейс програми представлений на рисунку 2.15.

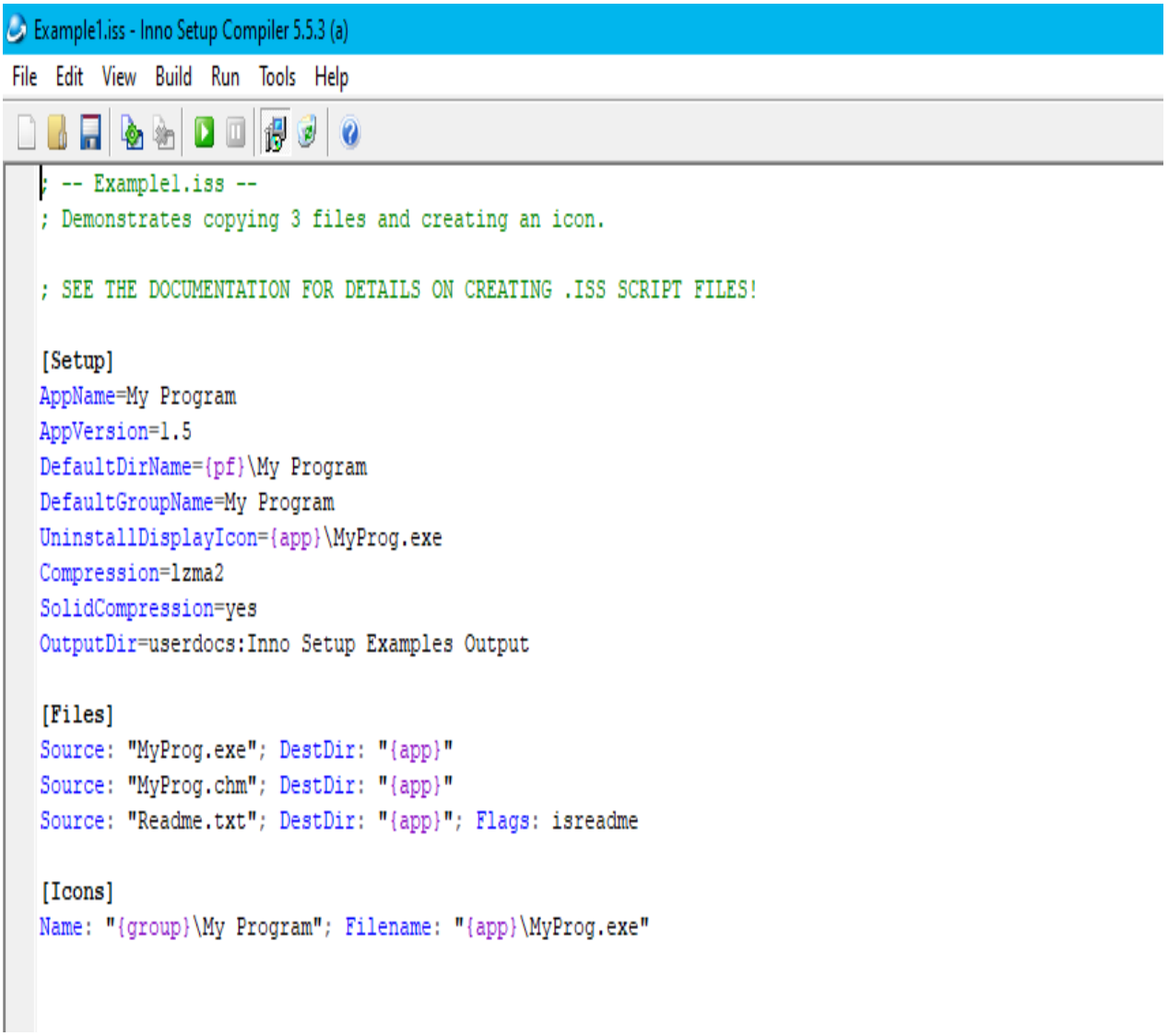

Рис. 2.15. Інтерфейс програми Inno Setup

2.6.3. Виклик та завантаження програми

Для встановлення розважального застосунку, слід запустити інсталятор, та пройти стандартні етапи завантаження програми. Зовнішній вигляд ярлику запуску інсталятора наведено на рисунку 2.16.

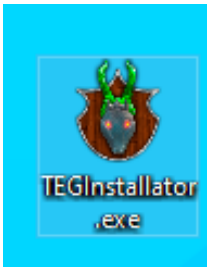

Рис. 2.16. Виконавчий файл, для встановлення застосунку

Інсталятор містить стандартні етапи встановлення (рис. 2.17), вибір мов, серед який англійська та українська, прийняття ліцензійної угоди, вибір директорії встановлення, вибір папки в меню пуск, створення ярлику на робочому столі.

Після встановлення додатку, на робочому столі з’явиться ярлик, для запуску програми.

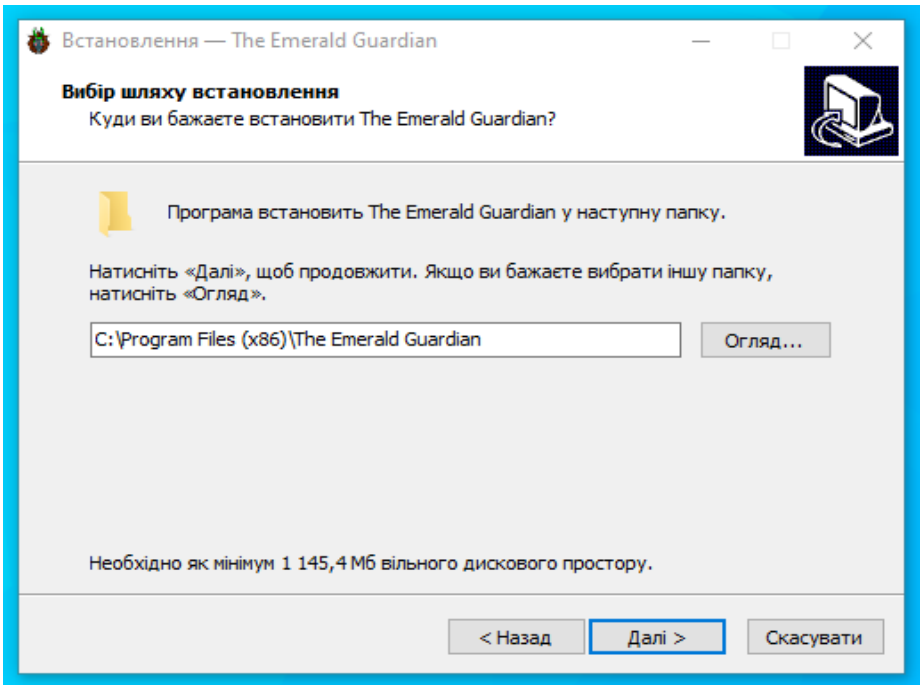

Рис. 2.17. Етап встановлення програми, вибір директорії

2.6.4 Опис інтерфейсу користувача

Рис. 2.18. Назва гри при запуску

Рис. 2.19. Головне меню гри

Рис. 2.20. Екран завантаження ігрової локації "Місто Аєтернум"

Рис. 2.21. Локація "Місто Аєтернум". Скриншот 1

Рис. 2.22. Локація "Місто Аєтернум". Скриншот 2

Рис. 2.23. Локація "Місто Аєтернум". Скриншот 3

Рис. 2.24. Меню персонажа. Вкладка Інвентар

Рис. 2.25. Меню персонажа. Вкладка Завдання

Рис. 2.26. Меню персонажа. Вкладка Глоссарій

Рис. 2.27. Меню персонажа. Вкладка Навички

Рис. 2.28. Меню персонажа. Вкладка Стародавня мова

Рис. 2.29. Меню персонажа. Вкладка Глобальна мапа

Рис. 2.30. Екран завантаження ігрової локації "Відкритий світ "

Рис. 2.31. Локація "Відкритий світ". Скриншот 1

Рис. 2.32. Локація "Відкритий світ". Скриншот 2

Рис. 2.33. Локація "Відкритий світ". Скриншот 3

Рис. 2.34. Локація "Відкритий світ". Ворог 1

Рис. 2.35. Локація "Відкритий світ". Ворог 2

Рис. 2.36. Головне меню у грі

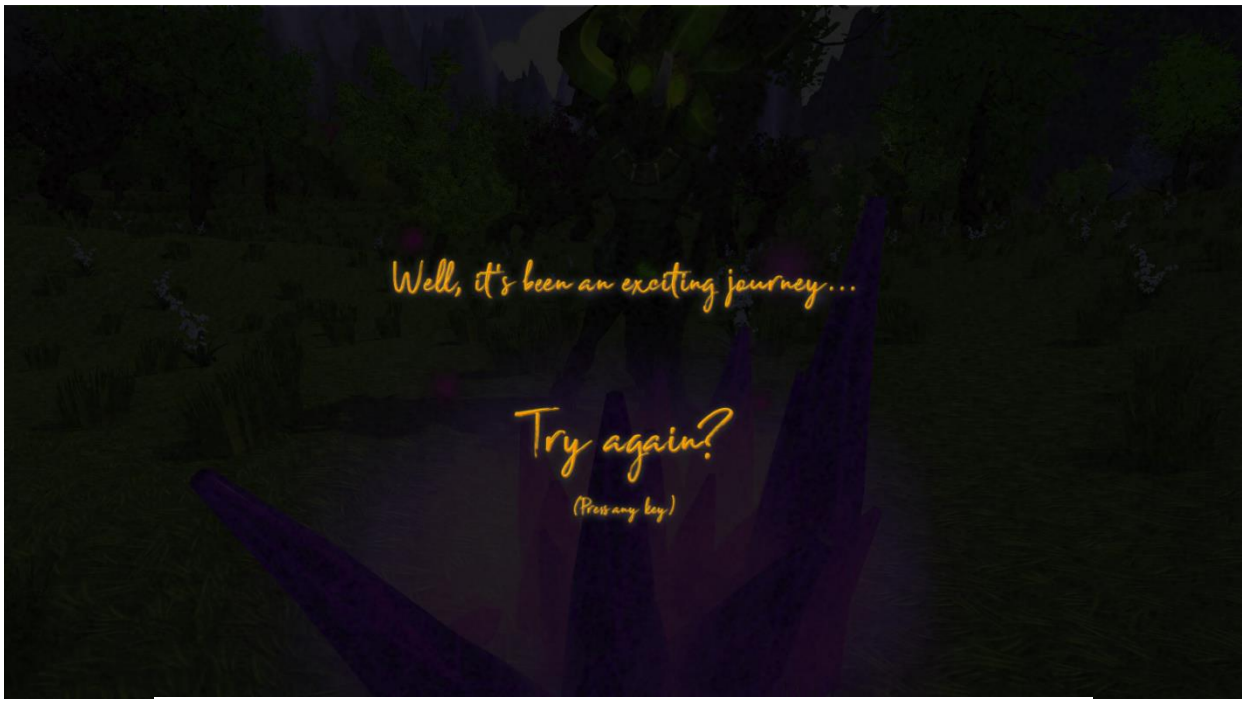

Рис. 2.37. Екран загибелі головного героя

РОЗДІЛ 3
ЕКОНОМІЧНІЙ РОЗДІЛ

3.1.Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 2000;

2. коефіцієнт складності програми – 1,6;

3. коефіцієнт корекції програми в ході її розробки – 0,1;

4. годинна заробітна плата програміста – 297 грн/год[13];

5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,3;

6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,2;

7. вартість машино-години ЕОМ – 1,99 грн/год.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин,} \qquad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50);

t_u – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

t_n – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ;

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p), \qquad (3.2)$$

де q – передбачуване число операторів (2000);

C – коефіцієнт складності програми (1,6);

p – коефіцієнт кореляції програми в ході її розробки (0,1).

$$Q = 2000 \cdot 1{,}6 \cdot (1 + 0{,}1) = 3520;$$

Витрати праці на вивчення опису задачі ti визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75 \ldots 85) \cdot K}, \text{ людино-годин} \qquad (3.3)$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі (1,3);

K – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності (1,2);

$$t_u = 3520 \cdot 1{,}3/(85 \cdot 1{,}2) = 44{,}8, \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20 \ldots 25) \cdot K}; \qquad (3.4)$$

$$t_a = 3520/(20 \cdot 1{,}2) = 146{,}7, \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25)\cdot K}; \quad (3.5)$$

t_n = 3520/(25 · 1,2) = 117, людино-годин.

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4...5)\cdot K}; \quad (3.6)$$

$t_{отл}$ = 3520/(5 · 1,2) = 586,7, людино-годин,

– за умови комплексного налагодження завдання:

$$t^{к}_{отл} = 1{,}2 \cdot t_{отл}; \quad (3.7)$$

$t^{к}_{отл}$ = 1,2 · 586,7 = 704, людино-годин,

Витрати праці на підготовку документації:

$$t_д = t_{дp} + t_{до}; \quad (3.8)$$

де $t_{дp}$ – трудомісткість підготовки матеріалів і рукопису;

$$t_д = \frac{Q}{(15...20)\cdot K}; \quad (3.9)$$

$t_д$ = 3520/(20 · 1,2) = 146,7, людино-годин,

де $t_{до}$ – трудомісткість редагування, печатки й оформлення документації;

$$t_{до} = 0{,}75\cdot t_{др}; \quad (3.10)$$

$t_{до}$ = 0,75 · 146,7 = 131,3, людино-годин.

$t_д$ = 146,7 + 131,3 = 278, людино-годин.

Отримаємо трудомісткість розробки програмного забезпечення:

t = 50 + 44,8+ 146,7 + 117 + 586,7 + 278 = 1223, людино-годин.

У результаті ми розрахували, що в загальній складності необхідно 1223 людино-годин для розробки даного програмного забезпечення.

3.2. Рахунок витрат на створення програми

Витрати на створення ПЗ *Кпо* включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн,} \qquad (3.11)$$

$З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн,} \qquad (3.12)$$

де t – загальна трудомісткість, людино-годин.

$С_{ПР}$ – середня годинна заробітна плата програміста, грн/година.

З урахуванням того, що середня годинна зарплата програміста становить 297 грн/год, то отримаємо:

$$З_{ЗП} = 1223 \cdot 297 = 363231, \text{ грн.}$$

Вартість машинного часу $З_{МВ,}$ необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{МВ} = t_{отл} \cdot С_{М}, \text{ грн,} \qquad (3.13)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год;

$C_{МЧ}$ – вартість машино-години ЕОМ, грн/год.

$$З_{МВ} = 586,7 \cdot 1,99 = 1168 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{ПО} = 363231 + 1168 = 364399 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ мес.} \qquad (3.14)$$

де B_k – число виконавців;

$F_р$ – місячний фонд робочого часу (при 40 годинному робочому тижні Fр=176 годин).

Витрати на створення програмного продукту:

$$T = 1223/(1 \cdot 176) = 6,9 \text{ міс.}$$

Висновки. Додаток має вартість 364399 грн. Ймовірний очікуваний час розробки – 6,9 місяці при стандартному 40-годинному робочому тижні і 176-годинному робочому місяці. Цей термін пов’язаний з кількістю операторів і включає в себе час для дослідження та розробку алгоритму розв’язання задачі, розробку дизайну і створення документації. На розробку додатку буде витрачено 1223 людино-годин.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено програмне забезпечення розважальний застосунок в Unity з дотриманням комерційного пайплайну для 3D моделей. Головною метою цього програмного забезпечення було створення розважального застосунку в середовищі Unity, з використанням мови програмування C#, з обов'язковим дотриманням комерційних норм та ліцензій для проекту. Створення 3d-моделей по виробничому пайплайну.

Програма була реалізована для операційної системи Windows, яка є основною платформою для цільової аудиторії продукту. Розробка додатку здійснювалась з використанням мови програмування C# та ігрового двигуна Unity. Для створення візуальної частини додатку були використані такі сучасні програми, як Blender, ZBrush, Photoshop, та Substance Painter.

Особливу увагу в розробленому програмному забезпеченні було приділено скриптам для керування ворогами, їх максимальній універсальності, що дозволяє створювати нових персонажів, без потреби зміни у програмному коді. Також було важливим дотримання всій ліцензійних угод на графічні та аудіо матеріали.

В "Економічному розділі" дипломної роботи були проведені розрахунки для визначення трудомісткості розробленого додатку (1223 людино-годин), вартості роботи по його створенню (364399 грн) та приблизного часу, витраченого на розробку (6,9 місяці).

Розроблений додаток відповідає поставленим цілям та завданням дипломної роботи і може бути надалі у розробці, а в результаті бути опублікованим.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Що таке відеоігри AAA URL: https://techukraine.net/%D1%89%D0%BE-%D1%82%D0%B0%D0%BA%D0%B5-%D0%B2%D1%96%D0%B4%D0%B5%D0%BE%D1%96%D0%B3%D1%80%D0%B8-aaa-triple-a/. Дата звернення 15.05.2023

2. Титани індустрії. 23 найкращі студії-розробники відеоігор URL: https://techno.nv.ua/ukr/technoblogs/krashchi-studiji-videoigor-50126837.html

3. Застосування тріангуляції Делоне для розв’язання евклідової задачі Штейн URL: https://ekmair.ukma.edu.ua/server/api/core/bitstreams/d87d676e-e28a-45ac-8caf-ea69083d2001/content. Дата звернення 15.05.2023

4. Blender URL: https://mysoft.com.ua/3d-modelyuvannya/4-blender.html. Дата звернення 15.05.2023

5. Unity Official Website URL: https://unity.com/ru/learn/get-started. Дата звернення 15.05.2023

6. Substance 3D Painter – Навчання й підтримка URL: https://helpx.adobe.com/ua/support/substance-3d-painter.html. Дата звернення 15.05.2023

7. Методичні рекомендації “Комп’ютерні технології. ZBrush” URL: http://eprints.kname.edu.ua/60190/1/2021%20%D0%9F%D0%95%D0%A7_535%D0%9C%20Zbrush.pdf. Дата звернення 16.05.2023

8. Blender Official Website URL: https://www.blender.org/about/. Дата звернення 15.05.2023

9. Substance 3D Painter Official Website URL: https://substance3d.adobe.com/education/. Дата звернення 15.05.2023

10. Maxon Official Website URL: https://www.maxon.net/en/zbrush/features?categories=1200575. Дата звернення 15.05.2023

11. Photoshop tutorials URL: https://helpx.adobe.com/ua/photoshop/tutorials.html. Дата звернення 17.05.2023

12. Wikipedia Atomm(Текстовий редактор) URL: https://uk.wikipedia.org/wiki/Atom_(%D1%82%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D0%B8%D0%B9_%D1%80%D0%B5%D0%B4%D0%B0%D0%BA%D1%82%D0%BE%D1%80). Дата звернення 17.05.2023

13. Unity developer: середня зарплата в Україні: https://www.work.ua/salary-unity+developer/#:~:text=%D0%92%20%D1%81%D0%B5%D1%80%D0%B5%D0%B4%D0%BD%D1%8C%D0%BE%D0%BC%D1%83%20%C2%ABUnity%20developer%C2%BB%20%D0%B2,Unity%203D%20%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BD%D0%B8%D0%BA%C2%BB%20%D1%82%D0%B0%20%D1%96%D0%BD. Дата звернення 17.05.2023

14. Що таке Unity? Курс Unity для митців URL: https://gamedev.dou.ua/forums/topic/38048/. Дата звернення 17.05.2023

15. Detecting Video Game-Specific Bad Smells in Unity Projects URL: https://dl.acm.org/doi/10.1145/3379597.3387454. Дата звернення 18.05.2023

16. C# Game Programming Cookbook for Unity 3D by Jeff W. Murray URL: https://www.taylorfrancis.com/books/mono/10.1201/9780429317132/game-programming-cookbook-unity-3d-jeff-murray. Дата звернення 18.05.2023

17. The Unity Game Engine and the Circuits of Cultural Software, Authors: Benjamin Nicoll, Brendan Keogh URL: https://link.springer.com/book/10.1007/978-3-030-25012-6. Дата звернення 19.05.2023

18. 3D Modelling and Visualization Based on the Unity Game Engine – Advatages and Challenges URL: https://isprs-annals.copernicus.org/articles/IV-4-W4/161/2017/isprs-annals-IV-4-W4-161-2017.pdf. Дата звернення 19.05.2023

19. Beginning 3D Game Assets Development Pipeline URL: https://link.springer.com/book/10.1007/978-1-4842-7196-4. Дата звернення 19.05.2023

20. Unity 3D animation modeling based on machine vision and embedded system URL: https://www.sciencedirect.com/science/article/abs/pii/S0141933121001137. Дата звернення 20.05.2023

ДОДАТОК А

КОД ПРОГРАМИ

Лістинг AmbientMusicController.cs

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

public class AmbientMusicController : MonoBehaviour
{
    public AudioClip ambientMusic;
    public AudioClip combatMusic;
    public float transitionTime;
    public float distanceEnemy;

    private AudioSource audioSource;
    private bool isCombatMusicPlaying = false;
    private List<GameObject> enemies;

    private void Start()
    {
        audioSource = GetComponent<AudioSource>();
        audioSource.clip = ambientMusic;
        audioSource.Play();

        enemies = new List<GameObject>(GameObject.FindGameObjectsWithTag("Enemy"));
    }

    private void Update()
    {
        bool shouldSwitchToCombatMusic = CheckPlayerProximityToEnemies();

        if (shouldSwitchToCombatMusic && !isCombatMusicPlaying)
        {
            StartCoroutine(TransitionMusic(combatMusic));
            isCombatMusicPlaying = true;
        }
        else if (!shouldSwitchToCombatMusic && isCombatMusicPlaying)
        {
            StartCoroutine(TransitionMusic(ambientMusic));
            isCombatMusicPlaying = false;
        }
    }

    private bool CheckPlayerProximityToEnemies()
    {
        GameObject player = GameObject.FindGameObjectWithTag("Player");
        for (int i = enemies.Count - 1; i >= 0; i--)
        {
            GameObject enemy = enemies[i];
            if (enemy != null)
```

```
            {
                float distance = Vector3.Distance(player.transform.position,
enemy.transform.position);
                if (distance <= distanceEnemy)
                {
                    return true;
                }
            }
            else
            {
                enemies.RemoveAt(i);
            }
        }
        return false;
    }

    private IEnumerator TransitionMusic(AudioClip newClip)
    {
        float elapsedTime = 0.0f;
        float startVolume = audioSource.volume;

        while (elapsedTime < transitionTime)
        {
            elapsedTime += Time.deltaTime;
            audioSource.volume = Mathf.Lerp(startVolume, 0.0f, elapsedTime / transitionTime);
            yield return null;
        }

        audioSource.clip = newClip;
        audioSource.Play();

        elapsedTime = 0.0f;

        while (elapsedTime < transitionTime)
        {
            elapsedTime += Time.deltaTime;
            audioSource.volume = Mathf.Lerp(0.0f, startVolume, elapsedTime / transitionTime);
            yield return null;
        }

        audioSource.volume = startVolume;
    }
}
```

Лістинг AttackBehaviour.cs

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

public class AttackBehaviour : StateMachineBehaviour
```

```
{
    Transform player;
    NavMeshAgent agent;


    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
      player = GameObject.FindGameObjectWithTag("Player").transform;
      agent = animator.GetComponent<NavMeshAgent>();

    }

    override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
       animator.transform.LookAt(player);
       agent.SetDestination(player.position);
       float distance = Vector3.Distance(animator.transform.position, player.position);


       if (distance > 7)
         animator.SetBool("isAttacking", false);
         animator.SetBool("isChasing", true);
    }

    override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {

    }
}
```

Лістинг Attacks_sound.cs

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

public class Attacks_sounds : MonoBehaviour
{
    public AudioClip[] stepSounds_AR;
    private AudioSource audioSource;

    private void Start()
    {
        audioSource = GetComponent<AudioSource>();
    }

    public void attack_1(){
      audioSource.PlayOneShot(stepSounds_AR[0]);
    }
```

```
    public void attack_2(){
      audioSource.PlayOneShot(stepSounds_AR[1]);
    }

    public void attack_3(){
      audioSource.PlayOneShot(stepSounds_AR[2]);
    }
  }
```

Лістинг BtnFX.cs

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

public class BtnFX : MonoBehaviour
{
  public AudioSource myFx;
  public AudioClip hoverFx;
  public AudioClip clickFx;

  public void HoverSound()
  {
    myFx.PlayOneShot(hoverFx);
  }

  public void ClickSound()
  {
    myFx.PlayOneShot(clickFx);
  }
}
```

Лістинг ChaseBehaviour.cs

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

public class ChaseBehaviour : StateMachineBehaviour
{
    NavMeshAgent agent;
    Transform player;
    float attackRange = 7;
    float chaseRange = 15;
    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int
layerIndex)
    {
        agent = animator.GetComponent<NavMeshAgent>();
        agent.speed = 4;
        player = GameObject.FindGameObjectWithTag("Player").transform;
```

```
        }
        override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
        {
            agent.SetDestination(player.position);
            float distance = Vector3.Distance(animator.transform.position, player.position);

            if (distance < attackRange)
              animator.SetBool("isChasing", false);
              animator.SetBool("isAttacking", true);

            if (distance > 15)
            {
              animator.SetBool("isChasing", false);
              animator.SetBool("isPatroling", true);
            }
        }

        override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
        {
            agent.SetDestination(agent.transform.position);
            agent.speed = 2;
        }
    }
```

Лістинг CursorManager.cs

```
using UnityEngine;

public class CursorManager : MonoBehaviour
{
    private bool isCursorVisible = false;

    private void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            isCursorVisible = !isCursorVisible;

            if (isCursorVisible)
            {
                Cursor.lockState = CursorLockMode.None;
                Cursor.visible = true;
            }
```

```
            else
            {
                Cursor.lockState = CursorLockMode.Locked;
                Cursor.visible = false;
            }
        }
    }
}
```

Лістинг Enemy_Shaman.cs

```
using System.Collections;
using UnityEngine;
using UnityEngine.AI;

public class Enemy_Shaman : MonoBehaviour
{
    public GameObject particle_atack;
    public float height;
    public float rotation;
    public float time_destroy;
    public float initialDelay;
    public float spawnInterval;
    public float chanceToSpawnWithOffset;
    public float spawnOffsetRange;
    public AudioClip[] spawnSounds;
    private AudioSource audioSource;


    private Coroutine spawnCoroutine;
    private Animator animator;
    private NavMeshAgent navMeshAgent;
    private Transform playerTransform;

    private void Start()
    {
        animator = GetComponent<Animator>();
        navMeshAgent = GetComponent<NavMeshAgent>();
        playerTransform = GameObject.FindGameObjectWithTag("Player").transform;

        spawnCoroutine = StartCoroutine(SpawnObjects());
        audioSource = GetComponent<AudioSource>();
    }

    private void OnDestroy()
    {
        if (spawnCoroutine != null)
        {
            StopCoroutine(spawnCoroutine);
        }
    }
```

```
    private IEnumerator SpawnObjects()
    {
        yield return new WaitForSeconds(initialDelay);

        while (true)
        {
            yield return new WaitUntil(() => IsAttackAnimationPlaying());

            Vector3 spawnPosition;

            // С вероятностью randomChance объект будет спавниться с погрешностью
            float randomChance = Random.value;
            if (randomChance <= chanceToSpawnWithOffset)
            {
                spawnPosition = GetSpawnPositionWithOffset();
            }
            else
            {
                spawnPosition = GetSpawnPosition();
            }

            CreateObject(spawnPosition);

            yield return new WaitForSeconds(spawnInterval);
        }
    }

    private bool IsAttackAnimationPlaying()
    {
        if (animator != null)
        {
            AnimatorStateInfo stateInfo = animator.GetCurrentAnimatorStateInfo(0);
            return stateInfo.IsName("Attack");
        }
        return false;
    }

    private Vector3 GetSpawnPosition()
    {
        // Получаем позицию игрока на поверхности Terraine с использованием
NavMeshAgent
        Vector3 playerPosition = playerTransform.position;
        NavMeshHit hit;
        if (NavMesh.SamplePosition(playerPosition, out hit, 10f, NavMesh.AllAreas))
        {
            return hit.position;
        }
        else
        {
            // Если не удалось найти позицию, возвращаем позицию игрока без изменений
            return playerPosition;
        }
```

```
    }

    private Vector3 GetSpawnPositionWithOffset()
  {
    Vector3 playerPosition = playerTransform.position;
    Vector3 spawnOffset = new Vector3(Random.Range(-1f, 1f), 0f, Random.Range(-1f, 1f)).normalized * spawnOffsetRange;

    // Add the spawnOffset to the playerPosition
    Vector3 spawnPosition = playerPosition + spawnOffset;

    // Use NavMesh.SamplePosition to get the position on the NavMesh
    NavMeshHit hit;
    if (NavMesh.SamplePosition(spawnPosition, out hit, 10f, NavMesh.AllAreas))
    {
      return hit.position;
    }
    else
    {
      // If failed to find a position, return the original player position without any offset
      return playerPosition;
    }
  }

    private void CreateObject(Vector3 spawnPosition)
    {
      GameObject newObject = Instantiate(particle_atack, spawnPosition, Quaternion.identity);
      newObject.transform.Translate(Vector3.down * height, Space.Self);
      newObject.transform.Rotate(Vector3.up, rotation);
      Destroy(newObject, time_destroy);

      if (spawnSounds.Length > 0)
    {
      int randomIndex = Random.Range(0, spawnSounds.Length);
      AudioClip randomSound = spawnSounds[randomIndex];
      AudioSource soundSource = gameObject.AddComponent<AudioSource>();
      soundSource.clip = randomSound;
      soundSource.Play();
      Destroy(soundSource, randomSound.length);
    }
    }
  }
```

Лістинг EnemyHealth.cs

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

public class EnemyHealth : MonoBehaviour
```

```
{
    public int maxHealth = 100;
    public Image HP_bar;
    public float delayBeforeDisappear = 15f;
    public AudioClip[] hitSounds;

    private int currentHealth;
    private Animator animator;
    private Collider enemyCollider;
    private bool isDead = false;
    private SkinnedMeshRenderer enemyRenderer;
    private AudioSource audioSource;

    private void Start()
    {
        currentHealth = maxHealth;
        animator = GetComponent<Animator>();
        enemyCollider = GetComponent<Collider>();
        enemyRenderer = GetComponentInChildren<SkinnedMeshRenderer>();
        audioSource = GetComponent<AudioSource>(); AudioSource
        UpdateHealthBar();
    }

    public void TakeDamage(int damageAmount)
    {
        if (isDead)
            return;

        currentHealth -= damageAmount;

        UpdateHealthBar();

        if (currentHealth <= 0 && !isDead)
        {
            Die();
        }
        else
        {
            PlayRandomHitSound();
        }
    }

    private void PlayRandomHitSound()
    {
        if (hitSounds.Length > 0 && audioSource != null)
        {
            int randomIndex = Random.Range(0, hitSounds.Length);
            audioSource.PlayOneShot(hitSounds[randomIndex]);
        }
    }

    private void UpdateHealthBar()
```

```
    {
        float healthPercentage = (float)currentHealth / maxHealth;
        HP_bar.fillAmount = healthPercentage;
    }

    private void Die()
    {
        isDead = true;

        int randomTrigger = Random.Range(0, 2);
        string dieTrigger = (randomTrigger == 0) ? "Die_1" : "Die_2";
        animator.SetTrigger(dieTrigger);


        enemyCollider.enabled = false;

        // Отключаем скрипты NPC Footsteps и Rik_sound_controller
        NPCFootsteps npcFootsteps = GetComponent<NPCFootsteps>();
        if (npcFootsteps != null)
            npcFootsteps.enabled = false;

        Rik_sound_controller rikSoundController = GetComponent<Rik_sound_controller>();
        if (rikSoundController != null)
            rikSoundController.enabled = false;

        PlayRandomHitSound();

        Destroy(gameObject, delayBeforeDisappear);
    }

    private IEnumerator Disappear()
    {
        yield return new WaitForSeconds(delayBeforeDisappear);

        Destroy(gameObject);
    }
}
```

Лістинг FPS_DISPLAY.cs

```
using UnityEngine;
using TMPro;

public class FPS_DISPLAY : MonoBehaviour
{
    public TextMeshProUGUI FpsText;

    private float pollingTime = 1f;
    private float time;
    private int frameCount;

    void Update()
```

```
    {
        time += Time.deltaTime;

        frameCount++;

        if(time >= pollingTime){
          int frameRate = Mathf.RoundToInt(frameCount/time);
          FpsText.text = frameRate.ToString() + " FPS";

          time -= pollingTime;
          frameCount = 0;
        }
    }
}
```

Лістинг Jump_sound.cs

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

public class Jump_sound : MonoBehaviour
{
    public AudioClip[] stepSounds_AR;
    private AudioSource audioSource;

    private void Start()
    {
        audioSource = GetComponent<AudioSource>();
    }

    public void Jump_up(){
      audioSource.PlayOneShot(stepSounds_AR[0]);
    }

    public void Jump_down(){
      audioSource.PlayOneShot(stepSounds_AR[1]);
    }
}
```

Лістинг Kuvirok_sound.cs

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

public class Kuvirok_sound : MonoBehaviour
{
    public AudioClip[] stepSounds_AR;
    private AudioSource audioSource;

    private void Start()
```

```
        {
            audioSource = GetComponent<AudioSource>();
        }

        public void Kuverok_sound_play(){
          audioSource.PlayOneShot(stepSounds_AR[Random.Range(0,
stepSounds_AR.Length)]);
        }
    }
```

Лістинг Load_OW.cs

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

public class Load_OW : MonoBehaviour
{
    public int numberScene;
    public string playerTag;
    public Image image;

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag(playerTag))
        {
            Debug.Log("Player entered the trigger!");
            image.enabled = true;
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag(playerTag))
        {
            Debug.Log("Player exited the trigger!");
            image.enabled = false;
        }
    }

    private void Start(){
      image.enabled = false;
    }

    private void OnTriggerStay(Collider other)
    {
        if (other.CompareTag(playerTag))
        {
            if (Input.GetKeyDown(KeyCode.F))
            {
```

```
                SceneManager.LoadScene(numberScene);
            }
        }
    }
}
```

Лістинг Load_script.cs

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

public class Load_script : MonoBehaviour
{

    public string next_scene_name = "Town_Scene";
    public GameObject progress_bar;
    public Image progress_bar_image;
    public GameObject press_key_hint;

    AsyncOperation async_operation;


    public void Intermediate_scene_load_button()
    {
      System.Threading.Thread.Sleep(2000);
      PlayerPrefs.SetString("current_scene", next_scene_name);
      SceneManager.LoadScene("LoadScreen");
    }

    // Start is called before the first frame update
    void Start()
    {
        if(SceneManager.GetActiveScene().name == "LoadScreen")
StartCoroutine("Async_load_press_key_COR", PlayerPrefs.GetString("current_scene"));
//PlayerPrefs.GetString("current_scene")
    }


    IEnumerator Async_load_press_key_COR(string scene_name)
    {
      float loading_progress;
      async_operation = SceneManager.LoadSceneAsync(scene_name);
      progress_bar.SetActive(true);

      async_operation.allowSceneActivation = false;
      while(async_operation.progress < 0.9f)
      {
        loading_progress = Mathf.Clamp01(async_operation.progress/0.9f);
        progress_bar_image.fillAmount = loading_progress;
```

```
            yield return true;
          }


          //progress_bar.SetActive(false);
          progress_bar_image.fillAmount = 1;
          press_key_hint.SetActive(true);
        }

        // Update is called once per frame
        void Update()
        {
          if(press_key_hint.activeSelf)
          {
            if(Input.anyKeyDown) async_operation.allowSceneActivation = true;
          }
        }
      }
```

Лістинг Long_attack_player.cs

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

      public class Long_attack_player : MonoBehaviour
      {

        public string enemyTag = "Enemy";
        public int baseDamage = 10;
        public float damageVariation = 5f;

        private void OnTriggerEnter(Collider other)
        {
          if (other.CompareTag(enemyTag))
          {
            Debug.Log("Yes!");
            GameObject enemy = other.gameObject;
            if (enemy != null)
            {
              int damage = CalculateDamage();
              enemy.GetComponent<EnemyHealth>().TakeDamage(damage);
            }
          }
        }

          private int CalculateDamage()
          {
            int randomVariation = Random.Range(-Mathf.RoundToInt(damageVariation),
Mathf.RoundToInt(damageVariation));
            int finalDamage = baseDamage + randomVariation;
            return finalDamage;
```

```
    }

}
```

Лістинг LookAtClosestEnemy.cs

```
using UnityEngine;

public class LookAtClosestEnemy : MonoBehaviour
{
    public string enemyTag = "Enemy";

    private Transform closestEnemy;

    private void Update()
    {
        FindClosestEnemy();
        RotateTowardsEnemy();
    }

    private void FindClosestEnemy()
    {
        GameObject[] enemies = GameObject.FindGameObjectsWithTag(enemyTag);
        float closestDistance = Mathf.Infinity;
        Transform playerTransform = transform;

        foreach (GameObject enemy in enemies)
        {
            float distance = Vector3.Distance(playerTransform.position,
enemy.transform.position);
            if (distance < closestDistance)
            {
                closestDistance = distance;
                closestEnemy = enemy.transform;
            }
        }
    }

    private void RotateTowardsEnemy()
    {
        if (closestEnemy != null)
        {
            Vector3 direction = closestEnemy.position - transform.position;
            Quaternion rotation = Quaternion.LookRotation(direction);
            transform.rotation = rotation;
        }
    }
}
```

Лістинг main_menu.cs

```
using System.Collections;
```

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

public class Main_menu : MonoBehaviour
{
  public void PlayGame()
  {
     SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
  }

  public void ExitGame()
  {
     Debug.Log("Game closed.");
     Application.Quit();
  }
}
```

Лістинг MinimapController.cs

```
using UnityEngine;
using UnityEngine.UI;

public class MinimapController : MonoBehaviour
{
    public Transform playerTransform;
    public Image mapImage;
    public Image frameImage;
    public Image mapMaskImage;
    public Vector3 offset;

    private Vector3 initialMapPosition;
    private Quaternion initialFrameRotation;

    private void Start()
    {
        initialMapPosition = mapImage.rectTransform.localPosition;
        initialFrameRotation = frameImage.rectTransform.localRotation;
    }

    private void LateUpdate()
    {
        Vector3 targetPosition = playerTransform.position + offset;
        mapImage.rectTransform.localPosition = initialMapPosition + new Vector3(-
targetPosition.x, -targetPosition.z, 0f) * mapImage.rectTransform.localScale.x;

        frameImage.rectTransform.localRotation = initialFrameRotation * Quaternion.Euler(0f,
0f, -playerTransform.eulerAngles.y);
    }
}
```

Лістинг movement.cs

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

public class movement : MonoBehaviour
{
    private Animator animator;
    public float Force;
    public CharacterController controller;
    public float smoothTime;
    float smoothVelocity;
    public Transform firstCamera;

    public float gravity;
    public float jumpForce;
    private float jspeed = 0;

    public float maxStamina = 100f;
    public float staminaCostPerRoll = 20f;
    public float staminaRecoveryRate = 10f;
    private float currentStamina;

    public Image staminaBar;

    private bool isRolling = false;
    private int rollingStateHash;

    private float rollCooldown = 0f;
    public float rollCooldownTime = 1.2f;

    private void Start()
    {
        animator = GetComponent<Animator>();
        controller = GetComponent<CharacterController>();
        currentStamina = maxStamina;
        UpdateStaminaUI();

        rollingStateHash = Animator.StringToHash("Base Layer.Roll");
    }

    private void Update()
    {
        float horizontal = Input.GetAxisRaw("Horizontal");
        float vertical = Input.GetAxisRaw("Vertical");
        Vector3 direction = new Vector3(horizontal, 0f, vertical).normalized;

        if (Input.GetKeyDown(KeyCode.Mouse1))
        {
            animator.SetBool("Block", true);
        }
```

```
        if (Input.GetKeyUp(KeyCode.Mouse1))
        {
            animator.SetBool("Block", false);
        }

        if (Input.GetKeyDown(KeyCode.LeftShift) && currentStamina >= staminaCostPerRoll
&& controller.isGrounded && !isRolling && Time.time > rollCooldown)
        {
            animator.SetTrigger("Dive");
            isRolling = true;
            currentStamina -= staminaCostPerRoll;
            rollCooldown = Time.time + rollCooldownTime;
            UpdateStaminaUI();
        }

        if (controller.isGrounded)
        {
            animator.SetBool("inAir", false);
            jspeed = 0;
            if (Input.GetKeyDown(KeyCode.Space) && !isRolling)
            {
                animator.SetTrigger("Jump");
                jspeed = jumpForce;
                animator.SetBool("inAir", true);
            }
        }

        jspeed -= gravity * Time.deltaTime * 0.4f;
        Vector3 v_jump = new Vector3(0f, jspeed * Time.deltaTime, 0f);
        controller.Move(v_jump);

        if (direction.magnitude >= 0.1f)
        {
            float rotationAngle = Mathf.Atan2(direction.x, direction.z) * Mathf.Rad2Deg +
firstCamera.eulerAngles.y;
            float angle = Mathf.SmoothDampAngle(transform.eulerAngles.y, rotationAngle, ref
smoothVelocity, smoothTime);
            transform.rotation = Quaternion.Euler(0f, angle, 0f);
            Vector3 move = Quaternion.Euler(0f, rotationAngle, 0f) * Vector3.forward;

            controller.Move(move.normalized * Force * Time.deltaTime);
            animator.SetBool("isMoving", true);
        }
        else
        {
            animator.SetBool("isMoving", false);
        }

        AnimatorStateInfo stateInfo = animator.GetCurrentAnimatorStateInfo(0);
        if (stateInfo.fullPathHash == rollingStateHash)
        {
```

```
            if (!isRolling)
            {
                isRolling = true;
            }
        }
        else
        {
            if (isRolling)
            {
                isRolling = false;
                currentStamina = Mathf.Clamp(currentStamina, 0f, maxStamina);
                UpdateStaminaUI();
            }
        }

        if (currentStamina < maxStamina && !isRolling)
        {
            currentStamina += staminaRecoveryRate * Time.deltaTime;
            currentStamina = Mathf.Clamp(currentStamina, 0f, maxStamina);
            UpdateStaminaUI();
        }
    }

    private void UpdateStaminaUI()
    {
        float staminaPercentage = currentStamina / maxStamina;
        staminaBar.fillAmount = staminaPercentage;
    }
}
```

Лістинг NPCFootsteps.cs

```
using UnityEngine;

public class NPCFootsteps : MonoBehaviour
{
    public AudioClip[] footstepSounds;
    public float maxDistance = 10f;
    public float minDistance = 2f;
    public float playerDistanceThreshold = 15f;

    private AudioSource audioSource;
    private Animator animator;
    private Transform playerTransform;
    private bool isPatroling;
    private bool isChasing;
    private bool isAttacking;

    private void Start()
    {
        audioSource = GetComponent<AudioSource>();
        animator = GetComponent<Animator>();
```

```
            playerTransform = GameObject.FindGameObjectWithTag("Player").transform;
        }

        private void Update()
        {
            float distanceToPlayer = Vector3.Distance(transform.position, playerTransform.position);

            isPatroling = animator.GetBool("isPatroling");
            isChasing = animator.GetBool("isChasing");
            isAttacking = animator.GetBool("isAttacking");

            if (distanceToPlayer < playerDistanceThreshold && (isPatroling || isChasing) && distanceToPlayer >= 7.0f)
            {
                float volume = Mathf.Lerp(1f, 0f, Mathf.InverseLerp(minDistance, maxDistance, distanceToPlayer));
                audioSource.volume = volume;

                if (!audioSource.isPlaying)
                {
                    AudioClip footstepSound = footstepSounds[Random.Range(0, footstepSounds.Length)];
                    audioSource.PlayOneShot(footstepSound);
                }
            }
            else
            {
                audioSource.Stop();
            }
        }
    }
```

Лістинг PatrolBehaviour.cs

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

public class PatrolBehaviour : StateMachineBehaviour
{

  float timer;
  List<Transform> points = new List<Transform>();
  NavMeshAgent agent;

  Transform player;
  float chaseRange = 15;
 override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
 {
```

```
        timer = 0;
        Transform pointsObject = GameObject.FindGameObjectWithTag("Points_Shaman").transform;
        foreach (Transform t in pointsObject)
          points.Add(t);

        agent = animator.GetComponent<NavMeshAgent>();
        agent.SetDestination(points[0].position);

        player = GameObject.FindGameObjectWithTag("Player").transform;
      }


      override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
      {
        if (agent.remainingDistance <= agent.stoppingDistance)
          agent.SetDestination(points[Random.Range(0, points.Count)].position);

          float distance = Vector3.Distance(animator.transform.position, player.position);
          if (distance < chaseRange)
          {
            animator.SetBool("isChasing", true);
            animator.SetBool("isPatroling", false);
          }
      }


      override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
      {
        agent.SetDestination(agent.transform.position);
      }

      }
```

Лістинг UIAnimationController.cs

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

public class UIAnimationController : MonoBehaviour
{
    private KeyCode inventoryKey = KeyCode.I;
    private KeyCode tasksKey = KeyCode.T;
    private KeyCode glossaryKey = KeyCode.G;
    private KeyCode skillsKey = KeyCode.P;
    private KeyCode langKey = KeyCode.L;
    private KeyCode mapKey = KeyCode.M;
    private KeyCode escKey = KeyCode.Escape;
```

```
        private string inventoryParam = "Inventory";
        private string tasksParam = "Tasks";
        private string glossaryParam = "Glossary";
        private string skillsParam = "Skills";
        private string langParam = "Lang";
        private string mapParam = "Map";

        private Animator animator;
        private Dictionary<KeyCode, string> tabKeyParams = new Dictionary<KeyCode, string>();
        private Dictionary<KeyCode, GameObject> tabKeyPanels = new Dictionary<KeyCode, GameObject>();

        private GameObject currentPanel;
        private bool isEscapePanelOpen = false;

        public GameObject Inventory_panel;
        public GameObject Tasks_panel;
        public GameObject Glossary_panel;
        public GameObject Skills_panel;
        public GameObject Lang_panel;
        public GameObject Map_panel;

        public GameObject cameraObject;
        private CinemachineBrain cinemachineBrain;

        public GameObject otherImage;

        void Start()
        {
            animator = GetComponent<Animator>();

            tabKeyParams.Add(inventoryKey, inventoryParam);
            tabKeyParams.Add(tasksKey, tasksParam);
            tabKeyParams.Add(glossaryKey, glossaryParam);
            tabKeyParams.Add(skillsKey, skillsParam);
            tabKeyParams.Add(langKey, langParam);
            tabKeyParams.Add(mapKey, mapParam);

            tabKeyPanels.Add(inventoryKey, Inventory_panel);
            tabKeyPanels.Add(tasksKey, Tasks_panel);
            tabKeyPanels.Add(glossaryKey, Glossary_panel);
            tabKeyPanels.Add(skillsKey, Skills_panel);
            tabKeyPanels.Add(langKey, Lang_panel);
            tabKeyPanels.Add(mapKey, Map_panel);

            cinemachineBrain = cameraObject.GetComponent<CinemachineBrain>();
            Cursor.visible = true;
            Cursor.lockState = CursorLockMode.None;
        }
```

```
void Update()
{

    if (isEscapePanelOpen && Input.GetKeyDown(escKey))
    {
        CloseEscapePanel();
        return;
    }

    if (!isEscapePanelOpen && Input.GetKeyDown(escKey))
    {
        OpenEscapePanel();
        return;
    }

    if (isEscapePanelOpen)
    {
        return;
    }

    foreach (KeyValuePair<KeyCode, string> kvp in tabKeyParams)
    {
        KeyCode tabKey = kvp.Key;
        string tabParam = kvp.Value;

        if (Input.GetKeyDown(tabKey))
        {
            bool isSamePanel = currentPanel == tabKeyPanels[tabKey];

            foreach (KeyValuePair<KeyCode, GameObject> kvp2 in tabKeyPanels)
            {
                KeyCode otherTabKey = kvp2.Key;
                GameObject otherTabPanel = kvp2.Value;
                string otherTabParam = tabKeyParams[otherTabKey];

                animator.SetBool(otherTabParam, false);
                otherTabPanel.SetActive(false);
            }

            if (isSamePanel)
            {
                animator.SetBool(tabParam, false);
                currentPanel.SetActive(false);
                currentPanel = null;
            }
            else
            {
                animator.SetBool(tabParam, true);
                GameObject selectedPanel = tabKeyPanels[tabKey];
                selectedPanel.SetActive(true);
                currentPanel = selectedPanel;
            }
```

```
            }
        }


        if (currentPanel != null && currentPanel.activeSelf)
        {
            cinemachineBrain.enabled = false;
            Cursor.visible = true;
            Cursor.lockState = CursorLockMode.None;
        }
        else
        {
            cinemachineBrain.enabled = true;
            Cursor.visible = false;
            Cursor.lockState = CursorLockMode.Locked;
        }
    }

    private void OpenEscapePanel()
    {
        isEscapePanelOpen = true;
        otherImage.SetActive(true);
        Cursor.visible = true;
        Cursor.lockState = CursorLockMode.None;
    }

    private void CloseEscapePanel()
    {
        isEscapePanelOpen = false;
        otherImage.SetActive(false);
        Cursor.visible = false;
        Cursor.lockState = CursorLockMode.Locked;
    }
}
```

ДОДАТОК Б

ВІДГУК
керівника економічного розділу на кваліфікаційну роботу бакалавра
на тему:
"Розробка програмного забезпечення розважального застосунку в середовищі Unity з дотриманням комерційного пайплайну для 3D моделей"
студента групи 121-20ск-1 Худіка Дмитра Володимировича

Керівник економічного розділу
доц. каф. ПЕП та ПУ, к.е.н **Л.В. Касьяненко**

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Кваліфікаційна робота Худік.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота Худік.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Худік.rar	Архів. Містить коди програми і скомпільовану програму
Презентація	
Худік.pptx	Презентація кваліфікаційної роботи