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 *Волошиної Валерії Вікторівни*
(ПІБ)

академічної групи *122-18-2*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Особливості 3D моделювання та анімації для ігор на прикладі створення ігрового застосунку за допомогою Unity 3D*

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

Дніпро
2022

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

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

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2022 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-18-2* (група) *Волошиної Валерії Вікторівни* (прізвище та ініціали)

тема кваліфікаційної роботи *Особливості 3D моделювання та анімації для ігор на прикладі створення ігрового застосунку за допомогою Unity 3D*

затверджена наказом ректора НТУ «ДП» від 18.05.2022 № 268-с

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2022 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2022 р.***

Завдання видав _____ *доц. Приходченко С.Д.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Волошина В.В.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2022 р.*

Термін подання кваліфікаційної роботи до ЕК: *13.06.2022 р.*

РЕФЕРАТ

Пояснювальна записка: 77 с., 24 рис., 3 дод., 20 джерел.

Об'єкт розробки: ігровий додаток в жанрі 3D-платформеру розроблений за допомогою Unity.

Мета кваліфікаційної роботи: розробити ігровий додаток жанру платформер в двигуні Unity та створити візуальний стиль проєкту за допомогою 3D-графіки.

У вступі виконується аналіз сучасного стану проблеми, уточнюється постановка завдання, мета кваліфікаційної роботи та галузь її застосування, обґрунтовується актуальність теми.

У першому розділі проводиться дослідження предметної області та існуючих рішень, визначається актуальність завдання та призначення розробки, розроблюється постановка завдання.

У другому розділі обирається платформа для розробки, виконується проектування програми і її розробка, наводиться опис алгоритму і структури функціонування системи, визначаються вхідні і вихідні дані, наводяться характеристики складу параметрів технічних засобів, описується робота програми.

В економічному розділі визначається трудомісткість розробленого програмного продукту, проводиться підрахунок вартості роботи по створенню застосунку та розраховується час на його створення.

Практичне значення полягає у розробці ігрового додатку в жанрі платформеру та візуального стилю проєкту за допомогою 3D-графіки.

Актуальність програмного продукту визначається великою популярністю комп’ютерних ігор та великим зростом важливості комп’ютерної 3D-графіки в останні роки.

Список ключових слів: ГРА, ІГРОВИЙ ДОДАТОК, ПЛАТФОРМЕР, ІГРОВИЙ ДВИГУН, 3D ГРАФІКА, UNITY, ВІЗУАЛЬНИЙ СТИЛЬ.

ABSTRACT

Explanatory note: 77 p., 24 figs., 3 appx., 20 sources.

Object of development: 3D-platformer game application developed with Unity.
Purpose of the qualification work: Develop a platformer game application in the Unity engine and create a visual style of the project using 3D graphics.

In the introduction it is analyzed the current state of the problem, clarified the problem, the purpose of the qualification work and the scope of its application, substantiated the relevance of the topic.

In the first section the research of the subject area and existing decisions is carried out, the urgency of the task and purpose of development is defined, the statement of the task is developed.

In the second section the platform for development is chosen, the program design and its development is carried out, the description of algorithm and structure of functioning of system is given, input and output data are defined, characteristics of structure of parameters of technical means are given, work of the program is described.

In the economic section it is determined the complexity of the developed software product, calculated the cost of work to create an application and calculated the time to create it.

Of practical importance is the development of a game application in the genre of platformer and visual style of the project using 3D graphics.

The relevance of the software product is determined by the great popularity of computer games and the growing importance of computer 3D graphics in recent years.

Keywords: GAME, GAME APPLICATION, PLATFORMER, GAME ENGINE, 3D GRAPHICS, UNITY, VISUAL STYLE.

.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

OS - операційна система;

GPU - графічний процесор;

CPU - центральний процесор;

ZIP - формат архівації файлів і стиснення даних без втрат;

Rigging - процес створення невидимого скелету для анімації;

Skinning - процес прив’язування скелету до 3D-моделі;

Highpoly - високополігональна сітка 3D-моделі;

Lowpoly - низькополігональна сітка 3D-моделі;

3D - тривимірний.

ВСТУП

Перші досягнення в історії 3D-моделювання відбулися, коли в 1960-х роках почали виходити перші комерційно доступні системи CAD. Найбільшим проривом став додаток Sketchpad, який розробив Іван Сазерленд, також відомий як «Робот-кресляр», з його революційним інтерфейсом. Sketchpad встановив, що комп'ютери можуть використовуватися не лише для інженерії чи повторюваного малювання, але й інтерактивно дизайнерами та, можливо, художниками.

Тепер із розвитком технологій тривимірне моделювання та візуалізація стали доступними, як ніколи. Фотореалізм, який створюється за допомогою 3D, важко відрізнити від реальності, будь-який сучасний фільм вже неможливо уявити без додавання графіки, а відеоігрова індустрія стала однією з найбільш передових гілок сфери інформаційних технологій.

Хоча неможливо точно передбачити, як технології та нові програми вплинуть на його майбутнє, дивлячись на попередню динаміку розвитку 3D-моделювання та його взаємозв'язку з розвитком технологій можна зробити висновок, що зростання потужностей призведе до подальшого активного прогресу.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Тривимірна (3D) графіка — це розділ комп'ютерної графіки який, за допомогою моделювання об'єктів у трьох вимірах, створює зображення, відео або віртуальних персонажів та оточення.

Основною задачею 3D-моделювання є розробка візуально об'ємного образу бажаного об'єкту. 3D-моделі можна створювати автоматично або вручну. Процес ручного моделювання підготовки геометричних даних для 3D комп'ютерної графіки схожий на пластичне мистецтво, наприклад ліплення. 3D-модель може бути фізично створена за допомогою пристроїв для 3D-друку, які формують 2D-шари моделі з тривимірним матеріалом, по одному шару.

Під час створення 3D-моделі виконавець повинен дотримуватися певних правил та інструкцій аби забезпечити точне та вчасне виконання завдання. Правильний порядок дій, що допомагає 3D-художнику дотримуватися стандартів індустрії називається пайплайном. Добре складений пайплайн допомагає розподілити навантаження, відповідає за оптимізацію часу та трудовитрат на кожному з етапів створення моделі, забезпечує ефективність роботи та вирішує цілий ряд технічних та художніх задач:

- В якому стилі має бути модель;
- Яке обмеження полігонів в низькополігональній сітці;
- Чи потрібно запікати карту нерівностей (normal);
- Який спосіб текстурингу буде використовуватись;
- Яка роздільна здатність текстур на квадратний метр (тексель), і якого розміру самі текстури;
- Чи потрібно створювати анімацію для моделі.

В ігровій індустрії процес починається з блокінгу та закінчується готовою моделлю всередині проєкту. Одним з найважливіших питань перед розробкою ігрового пайплайну є вибір між стилізацією та реалізмом.

Основною задачею реалізму є імітування життя. Такий самий об'єкт не обов'язково існує у реальності, але він має виглядати реально. Реалізм нашого часу досить сильно відрізняється від того, що було 10 років тому. На прикладі гри "Uncharted" 2007 року та 2017, можна побачити, як візуальна та технічна якість покращилась, дивіться рис.1.1.

Рис 1.1. Приклад змін вигляду головного героя

Прикладами підвищення якості деталей в моделі головного героя є шкіра, зморшки та одяг, зроблені за рахунок появлення PBR, якого не існувало 10 років тому.

Основні принципи стилізації полягають в тому, щоб передавати суть об'єктів з меншою кількістю деталей та робити акценти на колір та форму. Зазвичай стилізація використовується в мультиках, в ігровій індустрії вона з'явилась через необхідність оптимізації графіки для старих ПК, які просто не могли обробити необхідну для реалістичної моделі кількість полігонів. Приклад стилізованого персонажу наведено на рис. 1.2.

Рис 1.2. Приклад розвитку стилізації

Створення стилізованого персонажа є гарним прикладом пайплайну в ігровій індустрії:

- Вибір концепту та створення дошки референсів:

Перш за все необхідно зрозуміти яким повинен бути фінальний результат. Для цього проводиться пошук основного концепту і додаткових референсів.

Концепт-арт — це форма візуального мистецтва, яка використовується для передачі ідеї для використання у фільмах, відеоіграх, анімації, коміксах чи інших медіа, перш ніж вона буде втілена в кінцевий продукт. Концепт-арт розробляється через кілька ітерацій. Перед тим, як зупинитися на остаточному дизайні, досліджуються кілька рішень. Концепт-арт використовується не лише для розробки роботи, а й для того, щоб показати прогрес проекту директорам, клієнтам та інвесторам. Після завершення розробки роботи концепт-арт може бути перероблений і використаний для рекламних матеріалів.

Приклад концепт-арту для 3D-художника наведено на рис. 1.3.

Рис 1.3. Приклад концепту

- Драфт:

Етап створення спрощеної версії всієї моделі. Будь-який драфт починається з блокінгу (начерк моделі з примітивів), який передає суть об'єкта.

- Скульптинг (або створення highpoly сітки):

На цьому етапі створюється та деталізується уся модель без обмежень за кількістю полігонів, сітка моделі настільки щільна, що можна ліпити все, що завгодно, як зі шматка пластиліну. Високополігональна сітка потрібна лише для запікання деталей на lowpoly. Приклад highpoly моделі наведено на рис. 1.4.

Рис 1.4. Приклад високополігонального персонажа

- Ретопологія (або створення lowpoly сітки):

Завдання ретопології - знайти ідеальний баланс між виразністю та простотою моделі. Для цього кожний елемент моделі повинен мати функціональне завдання: впливати на силует або виправляти відблиск.

На цьому етапі створюється та сама модель, що буде завантажуватись в ігровий двигун. Приклад такої моделі наведено на рис. 1.5.

- Розгортка:

На цьому етапі ми розгортаємо lowpoly модель на площину для того, щоб запечені карти та текстури, які являються простими плоскими картинками, могли застосуватися до об'ємної моделі. Приклад розгортки наведено на рис 1.5.

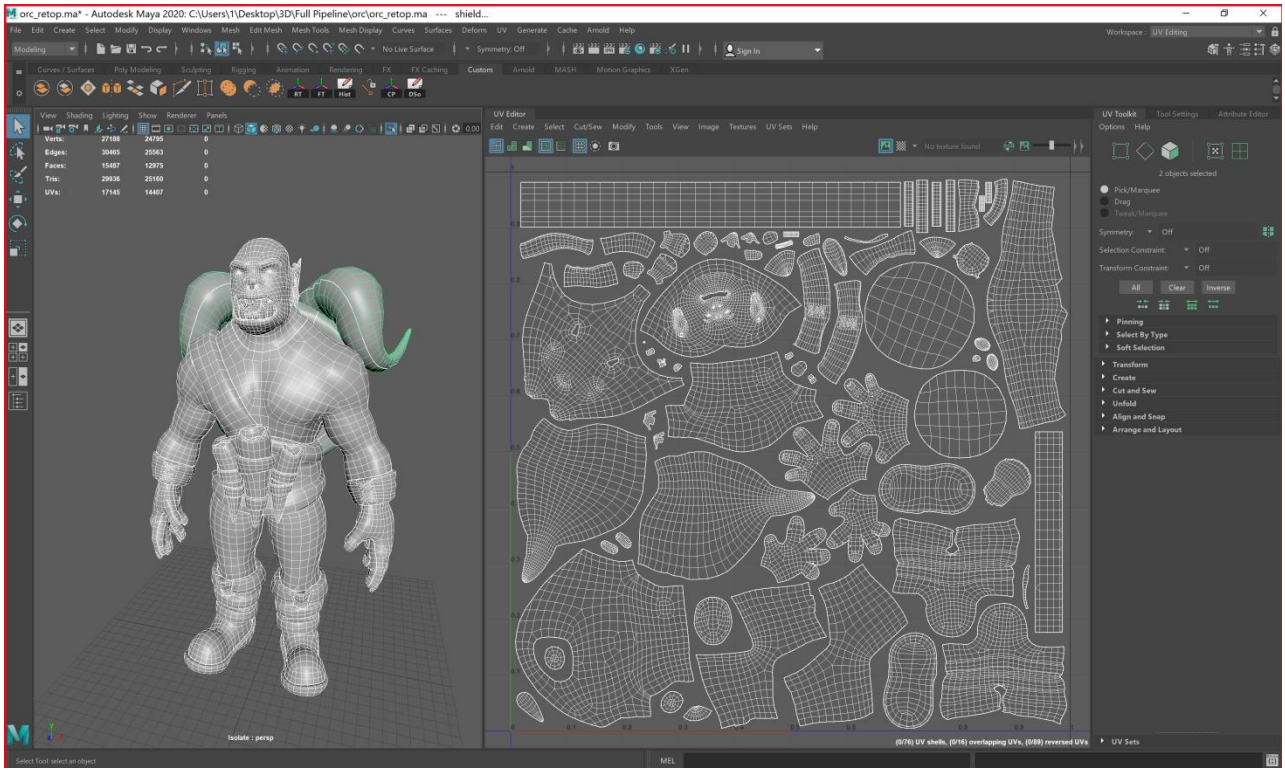

Рис. 1.5. Приклад ретопології та розгортки персонажа

- Baking maps:

Етап перенесення деталізації з високополігональної сітки на низькополігональну. Карта нормалей, яка запікається на цьому етапі, імітує високу деталізацію на оптимізованих низькополігональних моделях. Так само з високополігональної моделі запікається карта тіней та ID-map, яка допомагає у текстуруванні. Приклад карт наведено на рис. 1.6.

- Створення текстур:

Етап фарбування низькополігональної моделі. Існує кілька технологій візуалізації картинки зі своїми вимогами до текстур. Наприклад у старих іграх малювали карту кольору з вшитим у неї світлом і тінями, і карту відблисків, але в сучасних проєктах ААА використовується фізично коректний рендер — PBR.

Приклад текстур та персонажа наведено на рис. 1.6.

Рис 1.6. Приклад текстурних карт

- Rigging та Skining:

Rigging - це створення невидимого скелету персонажа, який рухають для створення анімації. Skining - процес прив'язування скелету до моделі.

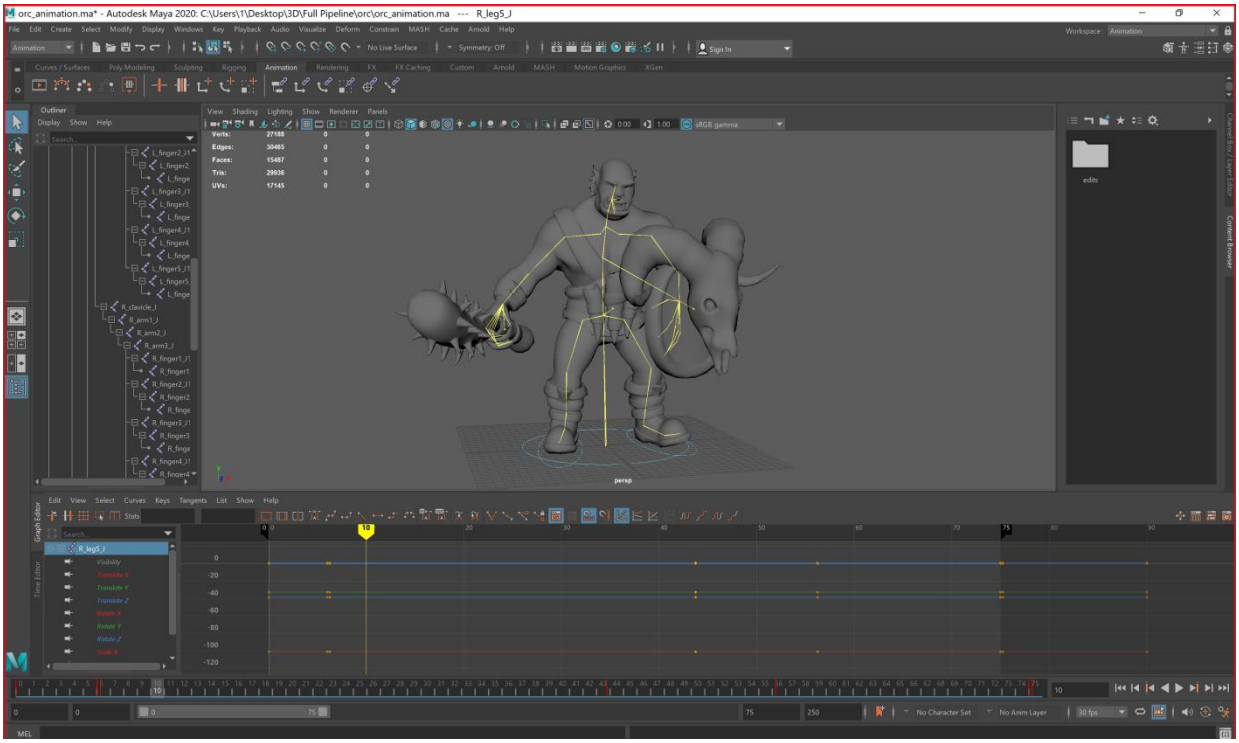

Рис 1.7. Вигляд скелету персонажа

- Анімація:

На цьому етапі необхідно проставити ключові кадри анімації, між якими програма сама прорахує необхідні рухи.

- Імпорт у двигун:

На етапі імпорту в двигун основною задачею є налаштування моделі, текстур та анімацій, щоб усе працювало як потрібно і персонаж або об’єкт виглядав відповідно концепту. Приклад наведено на рис. 1.8.

Рис 1.8. Фінальний вигляд персонажа в двигуні

Змінюються програми та технічні вимоги, проте порядок етапів зазвичай залишається незмінним. Правильно виконані етапи дають на виході якісну та оптимізовану під двигун 3D-модель.

1.2. Призначення розробки та галузь застосування

Сьогодні 3D-моделювання використовується в різних галузях:

- Медична промисловість використовує детальні моделі органів; вони можуть бути створені за допомогою кількох 2-вимірних зрізів зображення за допомогою МРТ або КТ.

- В науковому секторі використовуються як дуже деталізовані моделі хімічних сполук.

- Індустрія архітектури використовує 3D-моделі для демонстрації запропонованих будівель і ландшафтів замість традиційних фізичних архітектурних моделей.

- Археологічна спільнота зараз створює 3D-моделі культурної спадщини для дослідження та візуалізації.

- В кіноіндустрії використовуються для створення як персонажів і об’єктів для анімаційних і реальних фільмів.

- Індустрія відеоігор використовує моделі як активи для комп’ютерних і відеоігор. Завдяки сучасним можливостям 3D-моделювання є можливість створювати дуже якісні та реалістичні ігри, які розширюють межі свободи користувача.

Через покращення графіки ігор людині стає простіше переносити досвід здобутий в комп'ютерних іграх на реальне життя. Набагато легше асоціювати себе з персонажем гри якщо він виглядає майже як людина, а не як купа полігонів з текстурою обличчя на них.

Одним з прикладів користі досвіду відеоігр є прискорення процесу прийняття рішень. За результатами спостережень з'ясувалося, що шутери (наприклад, "Counter Strike"), в яких потрібно стежити за об'єктами, що несподівано виникають, і виконувати з ними певні дії, розвивають «низькорівневе сприйняття» і реакцію на дрібні цілі. Ігри жанру інтерактивне кіно (наприклад, "Beyond: Two Souls"), де основною задачею гравця є прийняття рішень від яких змінюється фінал гри, добре моделюють ситуації при яких

необхідно швидко зробити важливий вибір від якого залежить доля персонажа, або оточуючих його NPC (неігрові персонажі) та розвивають розуміння причинно-наслідкових зв'язків.

1.3. Підстави 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 122 «Комп'ютерні науки»;
- навчальний план та графік навчального процесу;
- наказ ректора Національного технічного університету «Дніпровська політехніка» № 268-с від 18.05.2022 р;
- завдання на кваліфікаційну роботу на тему "Особливості 3D-моделювання та анімації для ігор на прикладі створення ігрового застосунку за допомогою Unity 3D"

1.4. Постановка завдання

Завданням даної роботи є розробка ігрового додатку жанру платформер в двигуні Unity та створення візуального стил. проєкту за допомогою 3D-графіки.

В результаті необхідно спроєктувати та розробити гру для платформи Windows та за допомогою 3D-графіки створити персонажів та оточення.

Поставлена задача буде досягнена при виконанні таких умов:

- вивчення предметної галузі;
- написання коду програми;
- моделінг, анімація та інтегрування персонажів та оточення в двигун Unity.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Кінцевий продукт повинен дотримуватися наступних функціональних вимог:

- зрозумілий інтерфейс користувача;
- збереження налаштувань та прогресу;
- можливість керувати персонажем за допомогою клавіатури або геймпаду;

1.5.2. Вимоги до інформаційної безпеки

Для забезпечення інформаційної безпеки користувача повинні бути забезпечені такі умови:

- своєчасне встановлення оновлень;
- відсутність необхідності користувачу реєструватися та створювати акаунт всередині гри;

1.5.3. Вимоги до складу та параметрів технічних засобів

Відповідні характеристики необхідні для роботи додатку на Windows :

- OS: Windows 7 (SP1+) або вище
- CPU: Intel Core i3/i5/i7 1.8 GHz CPU dual-core. AMD 2.0 GHz dual-core;
- GPU: NVIDIA GeForce 260 / Radeon HD 4000 Series / Intel HD Graphics 4000;
- DirectX**:** Версії 10 або вище;
- накопичувач: 512мб;
- оперативна пам’ять: 4 Гб.

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 обораний C#.

Додаток не потребує особливої процедури встановлення, а просто розпаковується з zip-архіву. Оновлення відбувається шляхом перекачування архіву з мережі інтернет.

РОЗДІЛ 2

ПРОЄКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Загальні відомості з предметної галузі

Результатом даної кваліфікаційної роботи має бути ігровий додаток жанру платформер, розроблений в двигуні Unity, та візуальний стиль проєкту, створений за допомогою 3D-графіки.

Протягом ігрового процесу користувач повинен буде вирішувати головоломки за допомогою даних грою об’єктів, щоб пройти рівні гри та отримати винагороду у вигляді артефактів та золота.

Основним призначенням гри є розвиток логічного мислення та підвищення уваги користувача в розважальній формі.

2.2. Опис застосованих математичних методів

Під час створення 3D-моделей для проєкту використовувався математичний метод тріангуляції. Тріангуляцією називається процес розбиття полігональної області зі складною конфігурацією на трикутники. Будь-яку область можна розбити на трикутники, а трикутник в свою чергу є найпростішим полігоном, вершини якого однозначно задають грань. Алгоритми тріангуляції використовуються в багатьох процедурах машинної графіки. Наприклад, формування поверхонь, фарбування, видалення невидимих частин. В ігровій індустрії тріангуляція 3D-моделі є важливим етапом перед експортом до двигуна, так як тріангуляція моделі забезпечує коректне відображення геометрії, текстур та бліків в самій грі.

В іншому для роботи додатку не використовуються математичні методи, а лише прості алгоритмічні дії такі як: додавання, віднімання, ділення та множення.

2.3. Опис використаних технологій та мов програмування

Mixamo — це онлайн-база даних персонажів та мокап-анімацій, до якої кожен може отримати доступ для використання в художніх проектах, фільмах та іграх. Це надзвичайно зручна програма, орієнтована на людей, які мають замало досвіду створення та анімації персонажів. Це аж ніяк не робить його базовим програмним забезпеченням, насправді Mixamo може робити чимало. Особливо для розробника інді-ігор із обмеженим бюджетом, який шукає якісні анімації. Mixamo підтримує всі основні ігрові двигуни, включаючи Unreal Engine і Unity.

Marvelous Designer - це 3D-програмне забезпечення, яке використовуєть для моделювання тканини в основному для одягу в іграх, фільмах та інших CG-продуктах. Найважливішою технологією є реалістична симуляція тканина, яка дуже спрощує роботу 3D-художника, так як дає комфортну та коректно зроблену “болванку” одягу, яку можна експортувати в інші програми для допрацювання.

C# - це багатоцільова мова комп'ютерного програмування, яка підходить для широкого спектру потреб розробки. Хоча C# походить від мови програмування C, він представляє деякі унікальні та потужні функції, такі як делегати (які можна розглядати як безпечні для типу покажчики функцій) і лямбда-вирази, які вводять елементи функціональних мов програмування.

Ця мова програмування об’єктно-орієнтована, має велику бібліотеку класів і підтримує обробку винятків, кілька типів поліморфізму та відокремлення інтерфейсів від реалізацій. Ці функції в поєднанні з потужними інструментами розробки та підтримкою багатьох платформ роблять C# хорошим вибором для багатьох типів проектів розробки програмного забезпечення: проектів швидкої розробки додатків, проектів, реалізованих окремими особами або великими або малими командами. Його сильна типізація допомагає запобігти багатьом помилкам програмування, які є поширеними в слабодрукованих мовах.

Unity — популярна платформа розробки, яку можна використовувати безкоштовно (за деякими винятками). Двигун надає набір інструментів (активи, плагіни, бібліотеки), більшість з яких можна знайти на GitHub, Bitbucket, Unity

Asset Store та інших подібних платформах. Unity є хорошим і зручним інструментом з дуже низьким початковим рівнем. Він супроводжується великою кількістю добре описаних навчальних посібників і власною навчальною платформою. Двигун працює з C#, але також можна використовувати багато мов, таких як C++, Python, Java у формі бібліотек.

2.4. Опис структури системи та алгоритмів її функціонування

Для опису сценаріїв використання користувачем програмного продукту та взаємозв'язків у самому ігровому додатку я створила ULM діаграми Use Case:

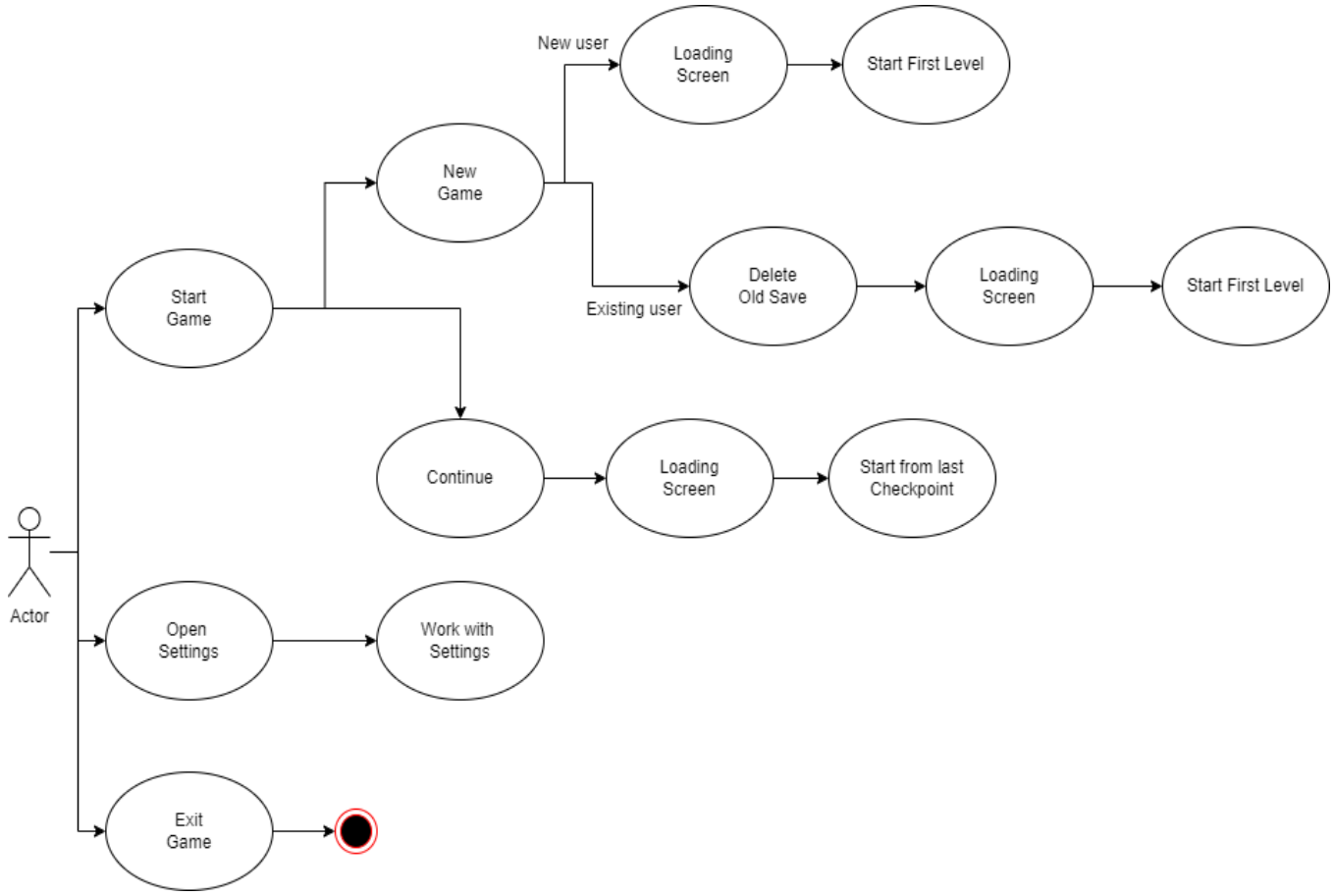

Рис. 2.1. UML діаграма UseCase для головного меню гри.

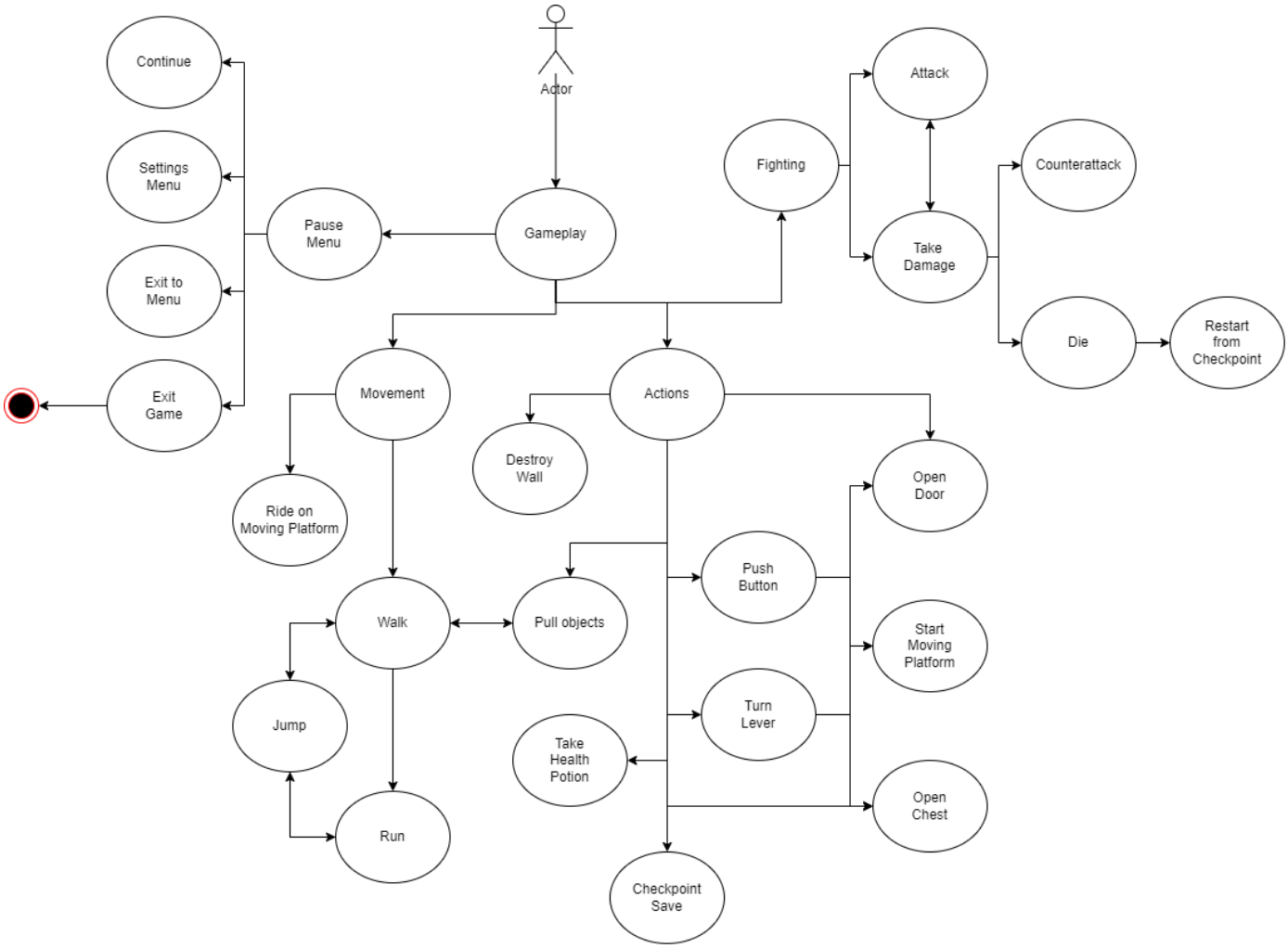

Рис. 2.2. UML діаграма UseCase для основного ігрового процесу.

Дії користувача можуть відрізнятися в залежності від того запустив він додаток вперше або вже мав певний прогрес у грі. У першому випадку він може почати грати лише з першого рівня і йти далі лише після його проходження. Якщо ж він уже запускав додаток та мав певну прогресію, то може обрати між продовженням з місця, де він зупинився минулого разу, або ж розпочати все спочатку. Також гравець може зайти на екран налаштувань та налаштувати частину гри під себе або ж вийти з гри через відповідну кнопку. Усі варіанти дій вказані на рис 2.1.

В основній грі користувачу необхідно проходити різні рівні вирішуючи головоломки за допомогою наявних в грі об’єктів. В гравця завжди є можливість

поставити гру на паузу, щоб змінити налаштування, повернутися на екран головного меню, або вийти з гри.

Протягом гри на рівнях розташовані чекпоінти (точки збереження прогресу) в вигляді табору з багаттям, які активуються і зберігають прогрес гри, коли гравець проходить повз них. Користувач розуміє, що процес гри збережено, коли в багатті спалахує вогонь, і на початку наступної ігрової сесії гравець зможе продовжити з цього моменту.

Основою геймплею є рух персонажа від початку ігрового рівня до фінішу, тому для покращення відчуттів від ігрового процесу була використана нова Input System Unity. У цій системі геймпад зазвичай відноситься до контролера з сучасним розташуванням, таким як джойстики, тригери та набір з чотирьох кнопок. Це дозволяє додати загальний геймпад-контролер, призначити йому кнопки і полегшити узгодження макетів керування грою на різних платформах і типах геймпадів. Завдяки розширеним налаштуванням та інтуїтивно зрозумілому інтерфейсу, вийшло зробити максимально комфортну для гравця механіку руху. У грі персонаж може пересуватися ходьбою та бігом, також реалізована механіка стрибків та рухомі платформи.

Для поліпшення геймплею реалізовано такі об’єкти та механіки:

- Рухома коробка - є одним з найважливіших об’єктів для ігор такого жанру. Гравець використовує її для того, щоб дістатися високо розташованих місць або може використовувати як укриття від атак ворогів.

- Стінка, яку можна зруйнувати використовується для розвитку уваги до деталей. Частіше за все, приховує за собою не обов’язковий для проходження рівня прохід, в якому може бути прихований бонус у вигляді цілющого зілля, тощо.

- Натискна плита може бути позитивним для ігрока активатором необхідного для проходження елементу або відкривати прохід до додаткової скрині. Також може використовуватись як пастка, наприклад при натиску випускати шипи, що наносять шкоду персонажу, або зачиняти гравця в кімнаті з ворогами, яких необхідно перемогти для виходу з кімнати.

- Важіль активації зазвичай відчиняє двері, активує рухомі платформи, тощо. Дуже поширена практика коли важіль активації захований десь на рівні та користувачу, який не може пройти рівень далі через заперті двері, необхідно шукати прохід, який він не помітив.

- Рухома платформа використовується як засіб проходження пастки. Зазвичай перевозить персонажа над якимось небезпечними поверхнями, наприклад лава або отрута. Гравцю необхідно у вдалий момент застрибнути на цю платформу і не звалитись в рідину, яка вб’є персонажа і змусить проходити рівень з останнього чекпоінту.

Також у грі реалізована бойова система. Протягом рівня персонаж зустрічає ворогів у вигляді скелетів, яких необхідно перемогти, щоб отримати нагороду або просто просунутись далі в проходженні рівня. Противники мають свою певну ділянку на карті, яку охороняють, і якщо користувач заходить на неї та потрапляє в їх поле огляду, то противник починає атакувати персонажа. Гравець повинен ухилятися від атак ворогів, щоб не отримати шкоду, та нанести достатню кількість ударів противнику, щоб перемогти його. Отримана персонажем шкода у певній кількості може привести до смерті та користувачу доведеться починати гру з останнього чекпоінту. Якщо персонаж втратив здоров’я, але все ж таки переміг ворога, то при подальшому проходженні рівня в нього є велика ймовірність знайти цілюще зілля, яке відновить його рівень здоров’я для подальшого проходження.

Для успішного проходження рівня гравцю необхідно пройти всі випробування і дістатися фінальної скрині, в якій на нього чекає винагорода у вигляді артефактів і золота. Усі вищеописані дії можна переглянути на рис. 2.2.

2.5. Обґрунтування та організація вхідних та вихідних даних програми

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

2.6. Опис розробленої системи

2.6.1. Вимоги до функціональних характеристик

Для розробки була використана електронно-обчислювальна машина(ноутбук) з нижченаведеними характеристиками:

- OS: Windows 10
- CPU: Intel Core i7-11370H (3.3 - 4.8 ГГц)
- GPU: NVIDIA GeForce GTX 3050
- Memory: SSD 1 TB
- RAM: 16 GB.
- Screen: 2880x1800 (WQXGA+)

А також такі периферійні засоби, як графічний планшет Huion INSPIROY H1161 та цифрова ручка PW100.

2.6.2. Використані програмні засоби

Autodesk Maya, який зазвичай скорочується до просто Maya, — це програма для комп’ютерної 3D-графіки, яка використовується для створення активів для інтерактивних 3D-додатків (включаючи відеоігри), анімаційних фільмів, серіалів та візуальних ефектів.

Користувачі визначають віртуальну робочу область (сцену) для реалізації та редагування конкретного медіа-проєкту. Maya пропонує архітектуру графа вузлів. Елементи сцени засновані на вузлах, кожен вузол має свої власні атрибути та налаштування. В результаті візуальне уявлення сцени повністю базується на мережі вузлів, що з'єднуються, залежно від інформації один одного. Для зручності перегляду цих мереж є залежність і орієнтований ациклічний граф. Приклад інтерфейсу Autodesk Maya наведено на рис. 2.3.

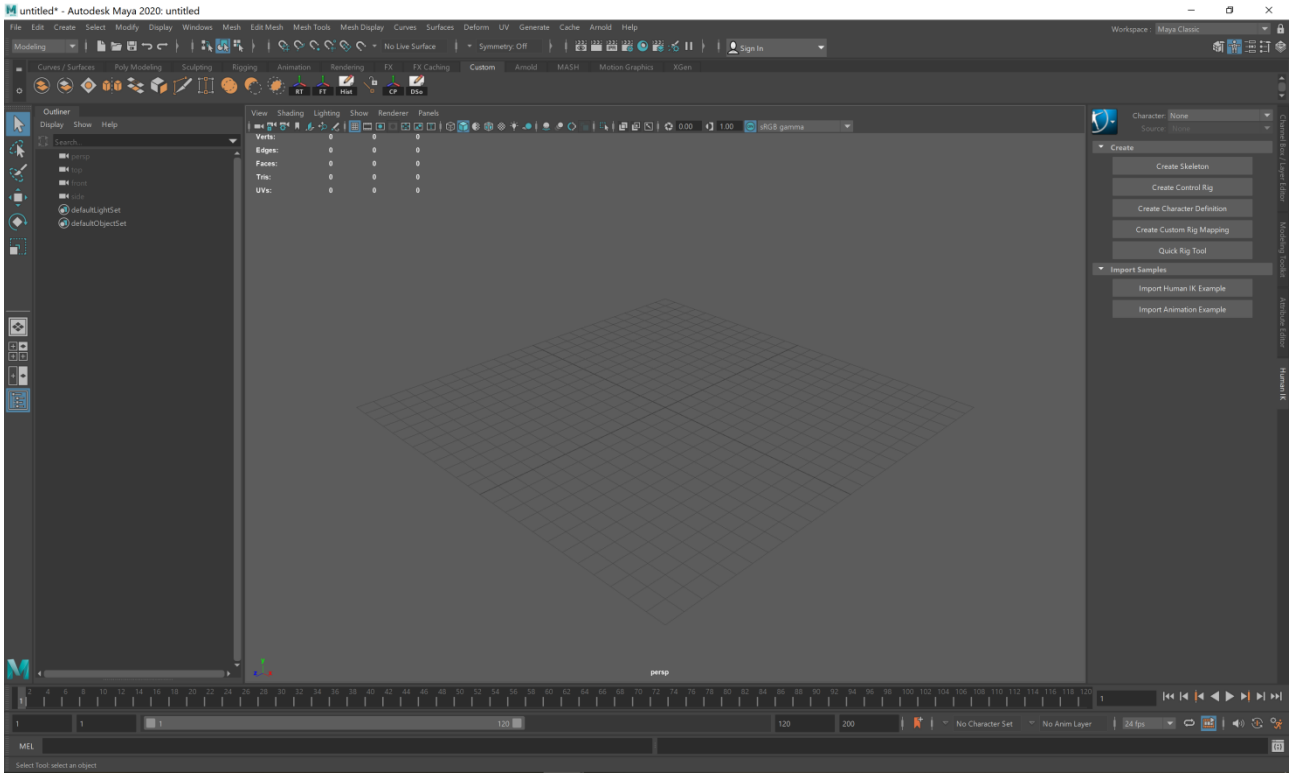

Рис. 2.3. Інтерфейс програми Maya.

Pixologic ZBrush – це інструмент для цифрової скульптури, який поєднує 3D/2,5D моделювання, текстурування та фарбування. Він використовує запатентовану технологію «піксол», яка зберігає інформацію про освітлення, колір, матеріал, орієнтацію та глибину для точок, що утворюють усі об'єкти на екрані. Основна відмінність ZBrush від більш традиційних пакетів моделювання полягає в тому, що він більше схожий на традиційне ліплення. ZBrush використовується для створення моделей з «високою роздільною здатністю» (здатних досягати понад 40 мільйонів полігонів) для використання у фільмах,

іграх та анімації компаніями, починаючи від ILM і Weta Digital, закінчуючи Epic Games і Electronic Arts.

ZBrush використовує динамічні рівні роздільної здатності, щоб скульптори могли вносити глобальні або локальні зміни до своїх моделей. Отримані деталі сітки потім можна експортувати як карти нормалей для використання на низькополігональній версії тієї ж моделі.

Приклад інтерфейсу ZBrush наведено на рис. 2.4.

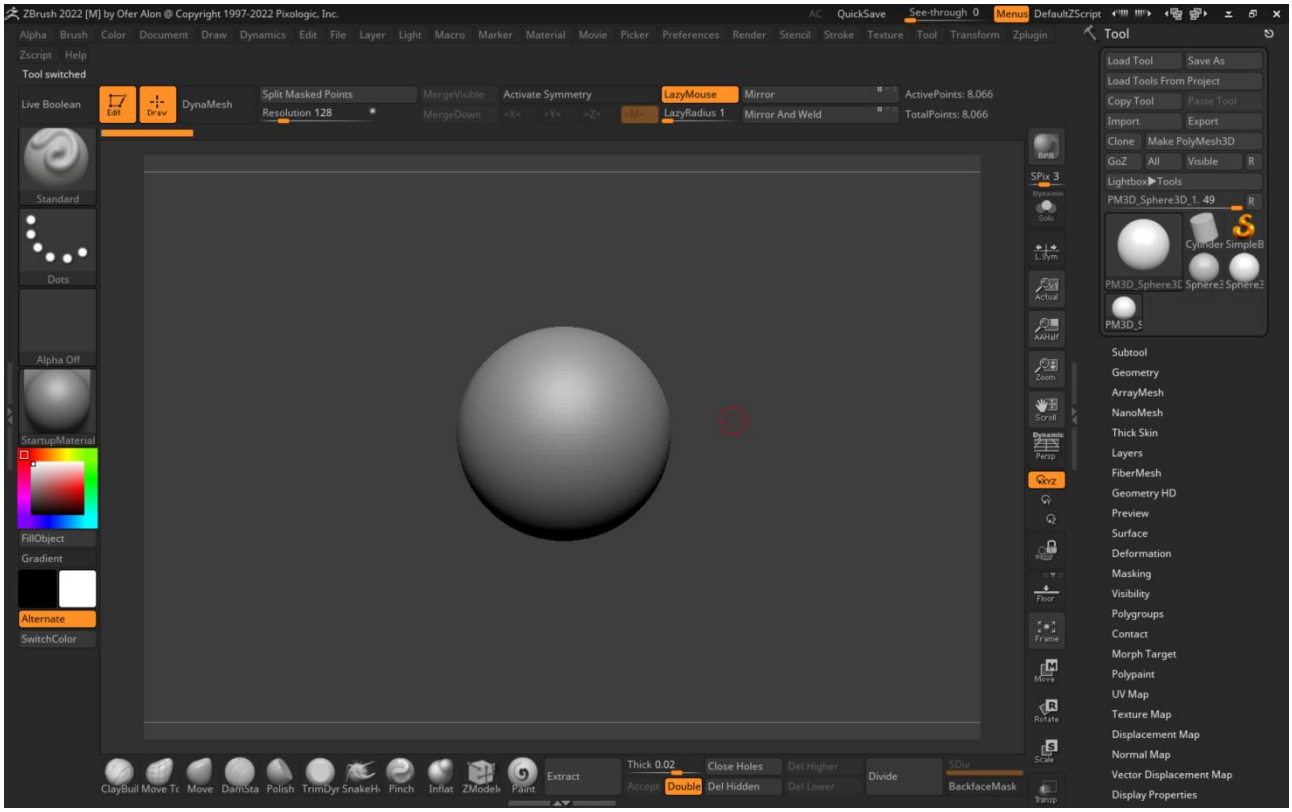

Рис. 2.4. Інтерфейс програми ZBrush.

Substance Painter – це неймовірно потужний інструмент для 3D-малювання. Його можна порівняти з 3D-версією Adobe Photoshop для цифрового малювання. Основна мета Substance Painter — текстурувати моделі. Його вдосконалені інструменти маскування та процедурного текстурування дозволяють створювати текстури, які набагато важче досягти в суто двовимірних програмах, таких як Photoshop.

Він має кілька дуже корисних функцій, наприклад, випікання текстур 8k, робочі процеси PBR матеріалів у реальному часі та багато попередніх

налаштувань матеріалів. Substance Painter надає можливість малювати як на 2D-картах, так і безпосередньо на 3D-моделі у вікні перегляду 3D.

Substance Painter відображає та експортує всі текстури у форматі PBR, тому вони імпортуються прямо в ігровий двигун з точно таким же результатом. Інтерфейс програми Substance Painter.наведено на рис. 2.5.

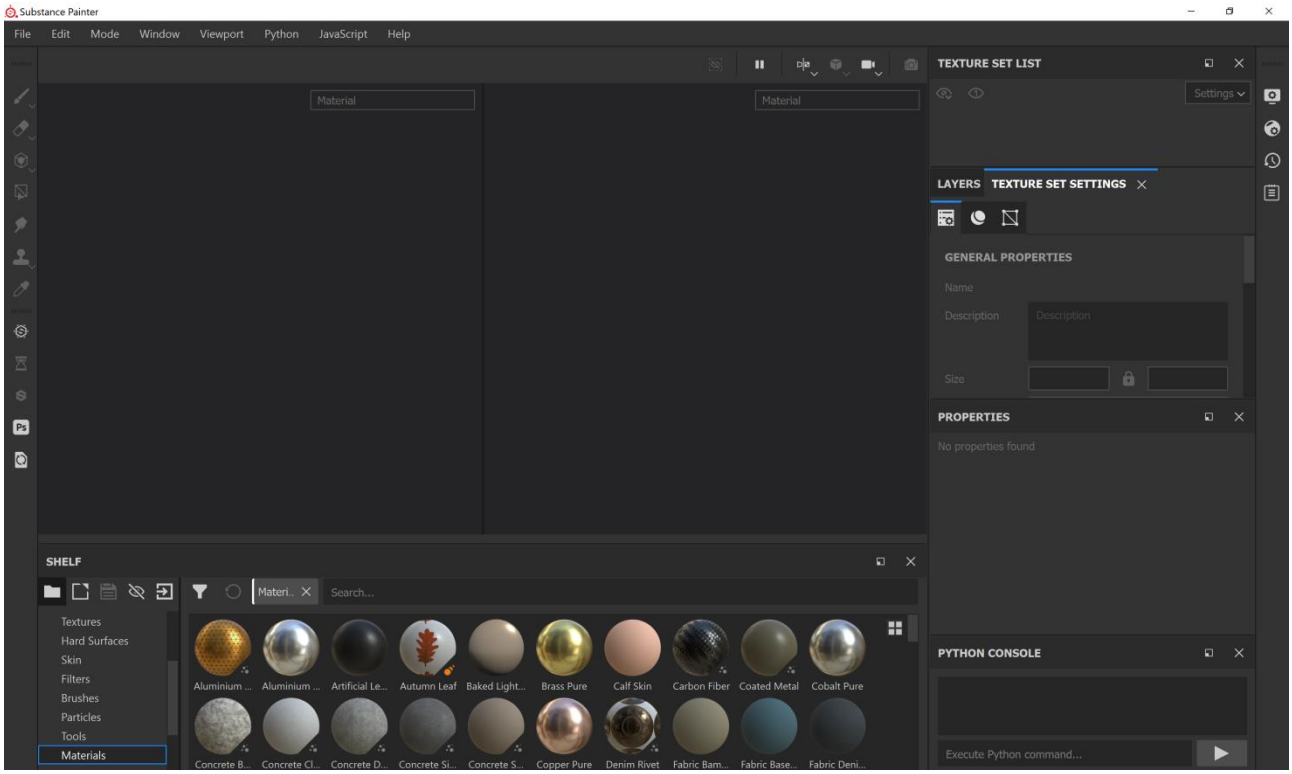

Рис. 2.5. Інтерфейс програми Substance Painter.

Headus UVLayout — це програма для створення та редагування координат UV-текстур для 3D-полісіток і поверхонь. Унікальний підхід UVLayout, який використовується професіоналами в індустрії ігор та візуальних ефектів, любителями всіх типів і студентами, дає художникам текстури інструменти, необхідні для створення високоякісних UV-текстур із низьким викривленням за значно менший час, ніж традиційними методами.

Marmoset Tollbag 4 - це мультиінструмент 3D-художника, який використовується для вирішення великої кількості художніх задач: рендер, текстуринг, налаштування сцени та запікання карт (normal map, ID map і тд).

Фізична візуалізація в реальному часі та освітлення на основі зображень є основою чудової якості зображення Toolbag.

Toolbag використовує потужність вашого графічного процесора (відеокарти) для створення надзвичайно швидкого випікання. Під час внесення змін до локальних ділянок вашої сітки він перезапікає лише уражену область, що призведе до майже миттєвого оновлення попереднього результату. Потужний механізм візуалізації GPU Toolbag здатний обробляти сітки дуже високої роздільної здатності, а також мільйони полігонів при використанні з сучасними відеокартами. Toolbag не має обмеження на багатокутники, однак продуктивність залежить від конкретного графічного процесора, який ви використовуєте. Інтерфейс Marmoset Toolbag.наведено на рис. 2.6

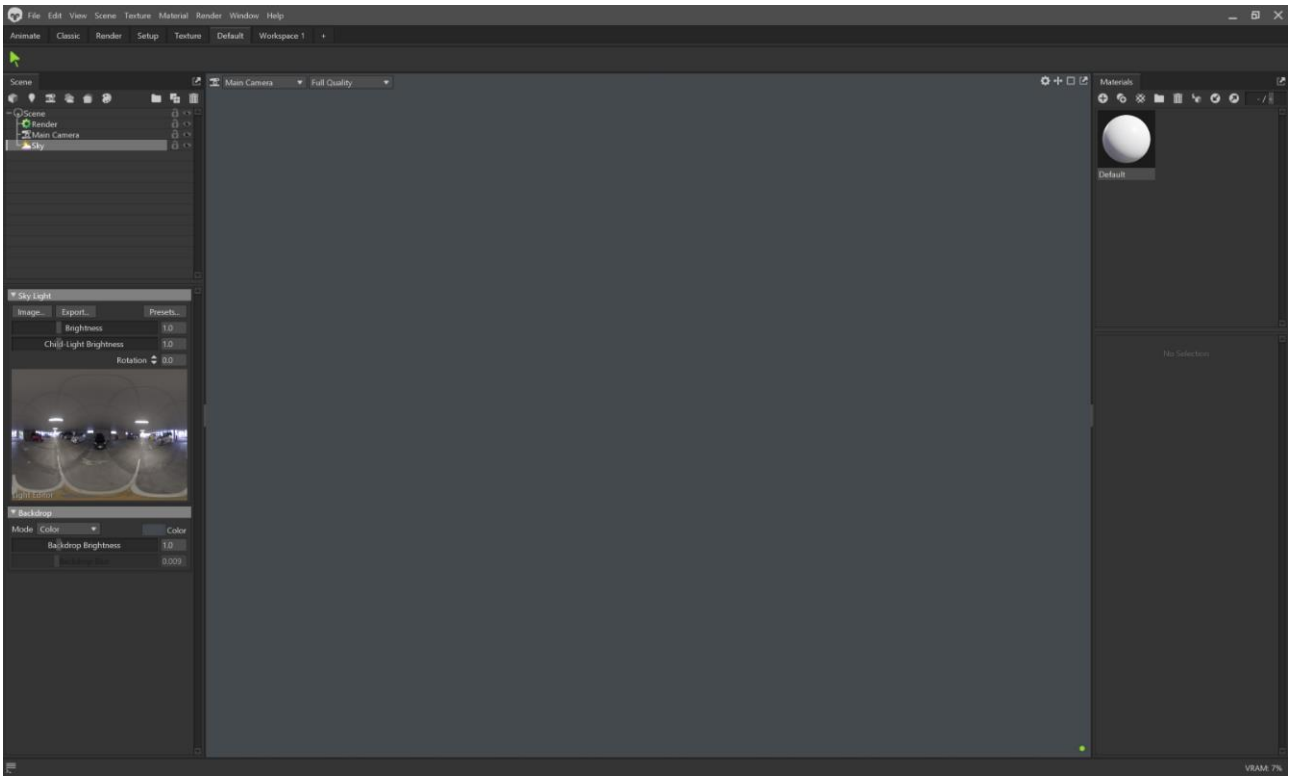

Рис. 2.6. Інтерфейс програми Marmoset Toolbag.

Jira - спочатку створювалась як система відстеження помилок, призначена для організації взаємодії з користувачами. Але дуже часто її використовують як

програма для управління проєктами. Основний елемент обліку в системі – завдання. Завдання включає в себе назву проекту, тип, тему, пріоритет, зміст та компоненти. Є можливість розширення завдання додатковими полями, додатками (наприклад - фотографіями, знімками екрану) або коментарями. Завдання може редагуватися або змінювати статус, наприклад, із «відкритий» до «закритого». Які переходи між станами можливі, визначається через потік операцій, що налаштовується. Будь-які зміни задачі протоколюються в журнал.

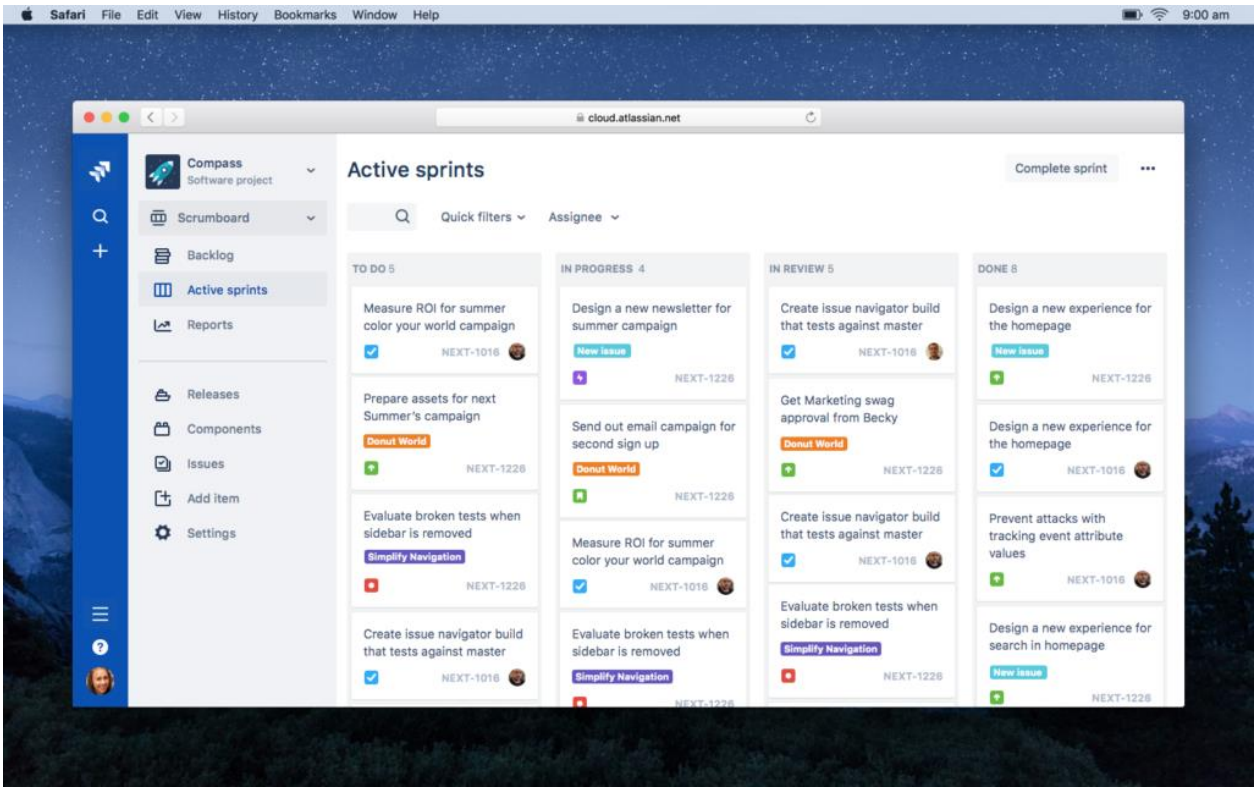

Рис. 2.7. Приклад інтерфейсу програми Jira

2.6.3. Виклик та завантаження програми

Для запуску проєкту необхідно завантажити архів з додатком із мережі Інтернет та розархівувати через будь-який архіватор. Після розпаковки архіву необхідно просто запустити виконавчий файл у папці програми - додаткової інсталяції не потрібно. Розпакований вигляд архіву наведено на рис. 2.7.

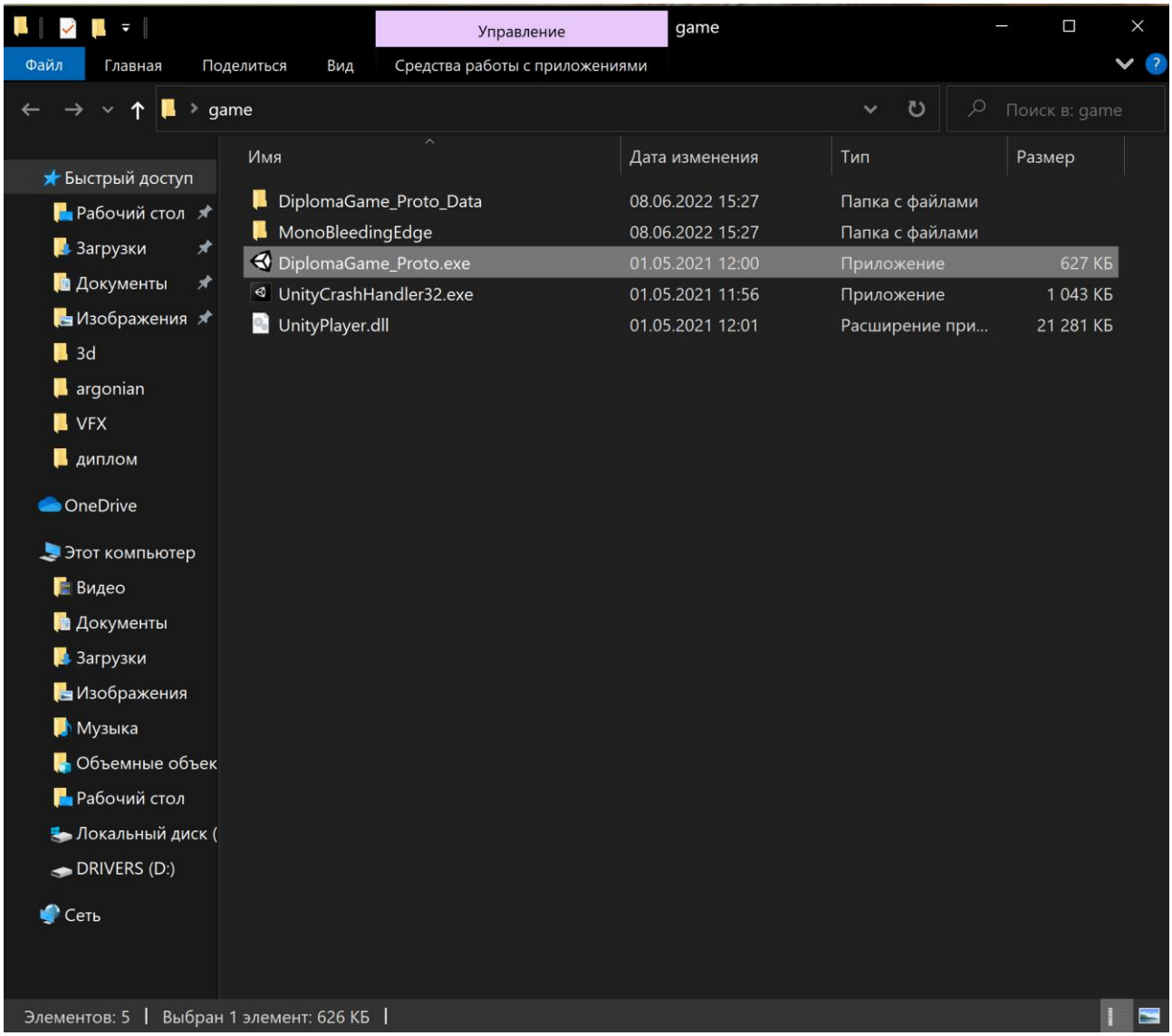

Рис. 2.8. Приклад папки з додатком після розпакування архіву.

2.6.4. Опис інтерфейсу користувача

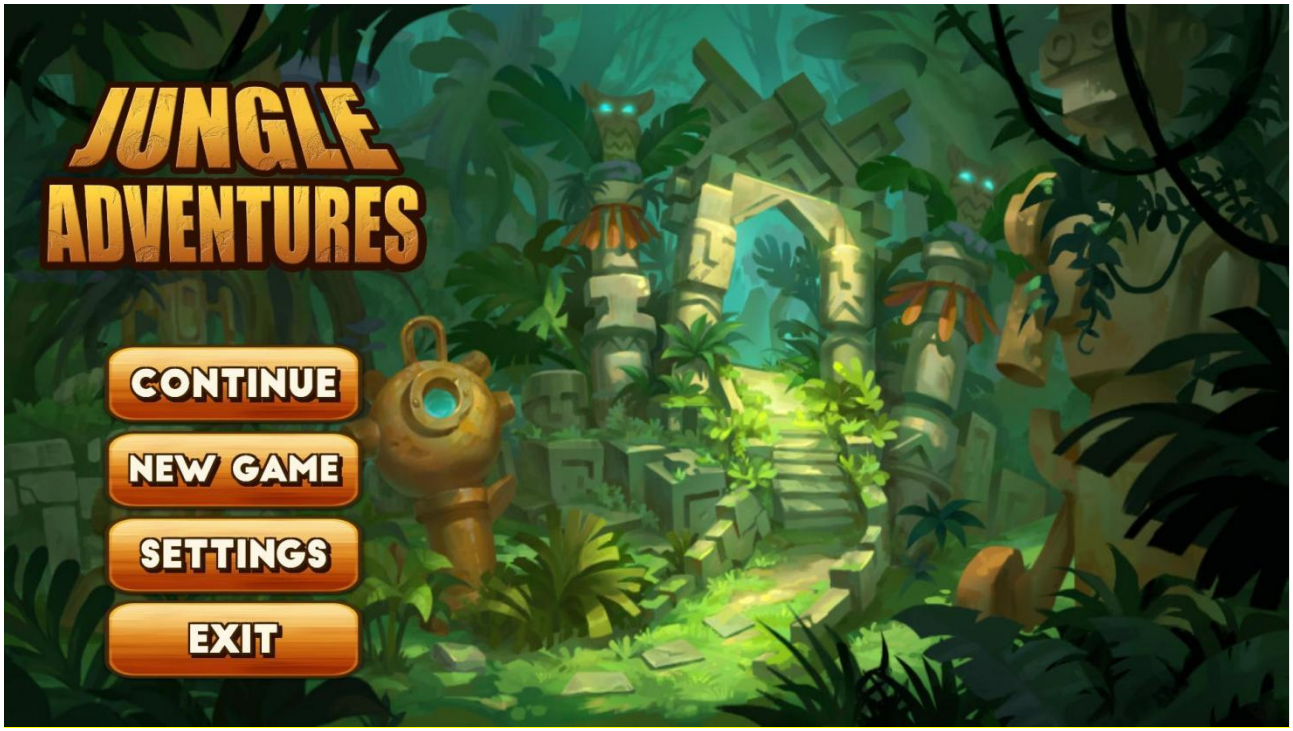

Рис. 2.9. Головне меню.

Рис. 2.10. Меню налаштувань.

Рис. 2.11. Вікно запиту для початку нової гри.

Рис. 2.12. Екран завантаження з головного меню до гри.

Рис. 2.13. Екран ігрового процесу.

Рис. 2.14. Екран паузи.

Рис. 2.15. Екран смерті.

Рис. 2.16. Екран перемоги.

РОЗДІЛ 3

ЕКОНОМІЧНІЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 1200;
2. коефіцієнт складності програми – 1,3;
3. коефіцієнт корекції програми в ході її розробки – 0,06;
4. годинна заробітна плата програміста – 112 грн/год;
5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,2;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,1;
7. вартість машино-години ЕОМ – 14 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 (2500);

C – коефіцієнт складності програми (1,4);

p – коефіцієнт кореляції програми в ході її розробки (0,06).

$$Q = 1200 \cdot 1{,}3 \cdot (1 + 0{,}06) = 1653.6;$$

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 (1,2);

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

$$t_u = \frac{1653{,}6 \cdot 1{,}2}{85 \cdot 1{,}1} = 21{,}2, \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

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

$$t_a = \frac{1653{,}6}{20 \cdot 1{,}1} = 75{,}2, \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

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

$$t_n = \frac{1653,6}{25 \cdot 1,1} = 60,1, \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

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

$$t_{отл} = \frac{1653,6}{5 \cdot 1,1} = 300,7, \text{ людино-годин,}$$

– за умови комплексного налагодження завдання:

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

$$t_{отл}^{к} = 1,2 \cdot 300,7 = 360,8, \text{ людино-годин,}$$

Витрати праці на підготовку документації:

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

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

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

$$t_{\partial} = \frac{1653,6}{20 \cdot 1,1} = 75,2, \text{ людино-годин,}$$

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

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

$$t_{\partial о} = 0,75 \cdot 75,2 = 56,4, \text{ людино-годин.}$$

$$t_{\partial} = 75,2 + 56,4 = 131,6, \text{ людино-годин.}$$

Отримаємо трудомісткість розробки програмного забезпечення:

t = 50 + 21,2+ 75,2 + 60,1 + 300,7 + 131,6 = 638,8, людино-годин.

У результаті ми розрахували, що в загальній складності необхідно 638,8 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;

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

З урахуванням того, що середня годинна зарплата програміста становить 112 грн/год, то отримаємо:

$$З_{ЗП} = 638{,}8 \cdot 112 = 71545{,}6, \text{ грн.}$$

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

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

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

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

$$З_{МВ} = 300{,}7 \cdot 14 = 4209{,}8 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{ПО} = 71545{,}6 + 4209{,}8 = 75755{,}4 \text{ грн.}$$

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:

$$T = \frac{638{,}8}{1 \cdot 176} = 3{,}6 \text{ міс.}$$

Висновки. Додаток має вартість 75755,4 грн. Ймовірний очікуваний час розробки – 3,6 місяці при стандартному 40-годинному робочому тижні і 178-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 638,8 людино-годин.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи був розроблений ігровий додаток жанру платформер в двигуні Unity та створено візуальний стиль проєкту за допомогою 3D-графіки.

Основним призначенням гри є розвиток логічного мислення та підвищення уваги користувача в розважальній формі. Протягом ігрового процесу користувач повинен вирішувати головоломки за допомогою даних грою об’єктів, щоб пройти рівні гри та отримати винагороду у вигляді артефактів та золота.

Програма працює на операційній системі Windows, яка широко використовується цільовою аудиторією продукту. Додаток реалізований на мові програмування C# за допомогою ігрового двигуна Unity. Також для створення візуальної частини додатку використовувались такі сучасні програми для 3D-моделювання: Maya, ZBrush, Headus UVLayout, Marmoset Toolbag, Substance Painter.

Під час виконання кваліфікаційної роботи у економічному розділі були проведені підрахунки для визначення трудомісткості розробленого додатку (638,8 людино-годин), вартості роботи по її створенню (75755,4 грн) та часу на витраченного на створення (3.6 міс).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is 3D Modeling & What's It Used For?. URL: https://conceptartempire.com/what-is-3d-modeling/ (дата звернення: 21.04.2022).
2. Vaughan W. Digital Modeling. New Riders Pub, 2017. 410 p.
3. What Is 3D Modeling?. URL: https://www.lifewire.com/what-is-3d-modeling-2164 (дата звернення: 22.04.2022).
4. Chopine A. 3D Art Essentials: The Fundamentals of 3D Modeling, Texturing, and Animation. Routledge, 2017. 288 p.
5. 3D virtual reality models help yield better surgical outcomes: Innovative technology improves visualization of patient anatomy, study finds. URL: https://www.sciencedaily.com/releases/2019/09/190918131457.htm (дата звернення 24.04.2022).
6. 3D Modelling Pipeline. URL:https://medium.com/@homicidalnacho/3d-modelling-pipeline-bd9be7dba136 (дата звернення: 25.04.2022)
7. What is Mixamo and How Can it be Used in Games. URL: https://cgobsession.com/what-is-mixamo-and-how-can-it-be-used-in-games/ (дата звернення 25.04.2022).
8. Albahari J. C# 10 in a Nutshell: The Definitive Reference. O`Reilly Media, 2022. 1058 p.
9. C#/.NET History Lesson. URL: http://jameskovacs.com/2007/09/07/cnet-history-lesson/ (дата звернення 26.04.2022).
10. Unity architecture. URL: https://docs.unity3d.com/Manual/unity-architecture.html (дата звернення 28.04.2022).
11. Lavieri E. Getting Started with Unity 2018: A Beginner's Guide to 2D and 3D game development with Unity, 3rd Edition. Packt Publishing, 2018. 336 p.

12. Unity 3 brings very expensive dev tools at a very low price. URL: https://arstechnica.com/information-technology/2010/09/unity-3-brings-very-expensive-dev-tools-at-a-very-low-price/ (дата звернення 29.04.2022).

13. Robert C. Martin’s Principle Collection. URL: http://principles-wiki.net/collections:robert_c._martin_s_principle_collection (дата звернення 29.04.2022).

14. Tickoo S. Autodesk Maya 2022: A Comprehensive Guide. CADCIM Technologies, 2021. 674 p.

15. ZBrush Features URL: Pixologic.com (дата звернення 1.05.2022).

16. What is Substance Painter. URL: https://conceptartempire.com/what-is-substance-painter (дата звернення 1.05.2022)

17. Headus UVLayout. URL: https://www.uvlayout.com/ (дата звернення 1.05.2022)

18. Ultimate Guide to Marvellous Designer with Camille Kleinman. URL: https://discover.therookies.co/2019/09/13/ultimate-guide-to-marvellous-designer-with-camille-kleinman/ (дата звернення 4.05.2022).

19. Jira. URL: Atlassian.com (дата звернення 4.05.2022).

20. Методичні вказівки з виконання економічного розділу в дипломних проектах студентів спеціальності “Комп’ютерні системи” / Укладачі О.Г.Вагонова, Нікітіна О.Б. Н.Н. Романюк – Дніпропетровськ: Національний гірничий університет. – 2013. – 23с.

ДОДАТОК А

КОД ПРОГРАМИ

Лістинг CharacterControl.cs:

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

public class CharacterControl : MonoBehaviour
{
    CharacterController playerController;
    Animator animator;
    FemalePlayerInput playerInput;

    int isRunningHash;
    Vector2 currentMovementInput;
    Vector3 currentMovement;
    bool isMovementPressed;

    public float rotationFactorPerFrame = 10f;
    public float speed = 3f;

    public float gravity = -9.8f;
    public float groundedGravity = -0.05f;

    bool isJumpPressed = false;
    float initialJumpVelocity;
    float maxJumpHeight = 4f;
    float maxJumpTime = 0.75f;
    int isJumpingHash;
    int jumpCountHash;
    bool isJumpAnimating;
    int jumpCount = 0;
    Dictionary<int, float> initialJumpVelocities = new Dictionary<int, float>();
    Dictionary<int, float> jumpGravities = new Dictionary<int, float>();

    public Vector3 externalMove;

    private void Awake()
    {
        playerInput = new FemalePlayerInput();
        playerController = GetComponent<CharacterController>();
        animator = GetComponent<Animator>();
        isRunningHash = Animator.StringToHash("isRun");
        isJumpingHash = Animator.StringToHash("isJump");
        jumpCountHash = Animator.StringToHash("JumpCount");
        playerInput.FemaleControls.Move.started += onMovementInput;
```

```
        playerInput.FemaleControls.Move.canceled += onMovementInput;
        playerInput.FemaleControls.Move.performed += onMovementInput;
        playerInput.FemaleControls.Jump.started += onJump;
        playerInput.FemaleControls.Jump.canceled += onJump;
        playerInput.FemaleControls.Hit.started += onHit;
        playerInput.FemaleControls.Hit.canceled += onHit;
        SetUpJumpVariables();
    }


    private void SetUpJumpVariables()
    {
        float timeToApex = maxJumpTime / 2;
        gravity = (-2 * maxJumpHeight) / Mathf.Pow(timeToApex,2);
        initialJumpVelocity = (2 * maxJumpHeight) / timeToApex;
        float secondJumpGravity = (-2 * (maxJumpHeight + 2)) / Mathf.Pow((timeToApex * 1.25f),
2);
        float secondJumpInitialVelocity = (2 * (maxJumpHeight + 2)) / (timeToApex * 1.25f);
        initialJumpVelocities.Add(1, initialJumpVelocity);
        initialJumpVelocities.Add(2, secondJumpInitialVelocity);
        jumpGravities.Add(0, gravity);
        jumpGravities.Add(1, gravity);
        jumpGravities.Add(2, gravity);
    }

    private void onJump(InputAction.CallbackContext context)
    {
        isJumpPressed = context.ReadValueAsButton();
        if (!isJumpPressed)
        {
            jumpCount = 0;
        }
    }

    private void onHit(InputAction.CallbackContext context)
    {
        bool isAttack = context.ReadValueAsButton();
        animator.SetBool("isAttack", isAttack);
    }

    void onMovementInput(InputAction.CallbackContext context)
    {
        currentMovementInput = context.ReadValue<Vector2>();
        currentMovement.x = currentMovementInput.x * speed;
        isMovementPressed = currentMovementInput.x != 0;// || currentMovementInput.y != 0;
    }

    void FixedUpdate()
    {
        handleRotation();
        handleAnimation();
       // if (currentMovement != Vector3.zero)
```

```
        {
            playerController.Move((currentMovement + externalMove) * Time.deltaTime);
        }
        handleGravity();
        handleJump();
    }

    private void LateUpdate()
    {
        transform.position = new Vector3(transform.position.x, transform.position.y, 0);
    }

    private void handleJump()
    {
        if (playerController.isGrounded && isJumpPressed)
        {
            animator.SetBool(isJumpingHash, true);
            isJumpAnimating = true;
            if (jumpCount == 1)
            {
                jumpCount = 0;
            }
            jumpCount += 1;
            animator.SetInteger(jumpCountHash, jumpCount);
            currentMovement.y = initialJumpVelocities[jumpCount] * 0.5f;
            isJumpPressed = false;
        }
    }


    void handleGravity()
    {
        bool isFalling = currentMovement.y <= 0.0f || !isJumpPressed;
        float fallMultuplier = 2f;
        if (playerController.isGrounded)
        {
            if (isJumpAnimating)
            {
                animator.SetBool(isJumpingHash, false);
                isJumpAnimating = false;
                animator.SetInteger(jumpCountHash, jumpCount);
            }
            currentMovement.y = groundedGravity;
        }
        else if (isFalling)
        {
            float previousYVelocity = currentMovement.y;
            float newYVelocity = currentMovement.y + (jumpGravities[jumpCount] * fallMultuplier *
Time.deltaTime);
            float nextYVelocity = Mathf.Max((previousYVelocity + newYVelocity) * 0.5f,-20f);
            currentMovement.y = nextYVelocity;
        }
```

```
        else
        {
            float previousYVelocity = currentMovement.y;
            float newYVelocity = currentMovement.y + (jumpGravities[jumpCount] *
Time.deltaTime);
            float nextYVelocity = (previousYVelocity + newYVelocity) * 0.5f;
            currentMovement.y = nextYVelocity;
        }
    }

    void handleRotation()
    {
        Vector3 positionToLookAt;

        positionToLookAt.x = currentMovement.x;
        positionToLookAt.y = 0f;
        positionToLookAt.z = currentMovement.z;

        Quaternion currentRotation = transform.rotation;

        if (isMovementPressed)
        {
            Quaternion targetRotation = Quaternion.LookRotation(positionToLookAt);
            transform.rotation = Quaternion.Slerp(currentRotation, targetRotation,
rotationFactorPerFrame * Time.deltaTime);
        }
    }

    void handleAnimation()
    {
        bool isRun = animator.GetBool(isRunningHash);

        if (isMovementPressed && !isRun)
        {
            animator.SetBool(isRunningHash, true);
        }
        else if (!isMovementPressed && isRun)
        {
            animator.SetBool(isRunningHash, false);
        }
    }

    private void OnEnable()
    {
        handleAnimation();
        playerInput.FemaleControls.Enable();
    }

    private void OnDisable()
    {
        playerInput.FemaleControls.Disable();
    }
```

```
    }
```

Лістинг PlayerController.cs:

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

namespace SkeletonEditor
{

    public class PlayerController : MonoBehaviour
    {
        public float mouseRotateSpeed = 0.3f;

        private Animator animator;
        private Quaternion initRotation;


        private int currentAnimation;
        private List<string> animations;


        private bool startMouseRotate;
        private Vector3 prevMousePosition;

        public static PlayerController Instance { get; private set; }

        void Awake() {
            if (Instance != null) {
                Destroy(this.gameObject);
            }
            Instance = this;
        }

        void Start() {
            animator = GetComponent<Animator>();
            initRotation = transform.rotation;


            animations = new List<string>()
            {
                "Hit1",
                "Fall1",
                "Attack1h1",
            };
        }

        void Update() {

            if (Input.GetMouseButtonDown(1)) {
                startMouseRotate = true;
```

```
            prevMousePosition = Input.mousePosition;
        }
        if (Input.GetMouseButtonUp(1)) {
            startMouseRotate = false;
        }
        if (Input.GetMouseButton(1)) {
            transform.Rotate(new Vector3(0, (Input.mousePosition.x - prevMousePosition.x) *
mouseRotateSpeed, 0));
            prevMousePosition = Input.mousePosition;
        }
        if (Input.GetKeyDown(KeyCode.E)) {
            animator.SetTrigger("Attack1h1");
        }
        if (Input.GetKeyDown(KeyCode.R))
        {
            animator.SetTrigger("Hit1");
        }
        if (Input.GetKeyDown(KeyCode.T))
        {
            animator.SetTrigger("Fall1");
        }
        if (Input.GetKeyDown(KeyCode.Y))
        {
            animator.SetTrigger("Up");
        }

        float h = Input.GetAxis("Horizontal");
        float v = Input.GetAxis("Vertical");

        if (Mathf.Abs(h) > 0.001f)
            v = 0;


        if (!startMouseRotate) {
            if (h > 0.5f) {
                transform.rotation = Quaternion.Euler(initRotation.eulerAngles + new Vector3(0, -90,
0));
            }
            if (h < -0.5f) {
                transform.rotation = Quaternion.Euler(initRotation.eulerAngles + new Vector3(0, 90,
0));
            }
            if (v > 0.5f) {
                transform.rotation = Quaternion.Euler(initRotation.eulerAngles + new Vector3(0, -180,
0));
            }
            if (v < -0.5f) {
                transform.rotation = Quaternion.Euler(initRotation.eulerAngles);
            }
        }

        var speed = Mathf.Max(Mathf.Abs(h), Mathf.Abs(v));
```

```
            animator.SetFloat("speedv", speed);
        }
    }
        }
```

Лістинг CamFollower.cs:

```
using UnityEngine;

public class CamFollower : MonoBehaviour
{
#pragma warning disable 0649

    public Transform Target;

    [Space][Header("Vector")]
    [SerializeField] private float _up;
    [SerializeField] private float _back;

    [Space][Space]
    [SerializeField] private bool _dropTarget;

#pragma warning restore 0649

    private Vector3 _temp;
    public static CamFollower McThis;
    private void Awake()
    {
        McThis = this;
    }
    private void FixedUpdate()
    {
        if (!Target) return;
        MoveVector();
    }
    void MoveVector()
    {
        _temp = Target.position;
        _temp.y += _up;
        _temp.z += _back;

        transform.position = Vector3.Lerp(transform.position,_temp,Time.deltaTime*3);
    }
        }
```

Лістинг GameManager.cs:

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

public class GameManager : MonoBehaviour
{
    public static GameManager Instance;
    private UIManager _uiManager;
    private CheckPointManager _checkPointManager;
    public Action<Vector3> Restart;

    public bool GameIsPaused = false;

    private void OnEnable()
    {
        if (Instance != null)
        {
            Destroy(Instance.gameObject);
        }

        Instance = this;
        DontDestroyOnLoad(gameObject);
    }

    public void GameOver()
    {
        if (!_uiManager)
        {
            _uiManager = FindObjectOfType<UIManager>();
        }
        _uiManager.ShowGameOver();
    }

    public void RestartFromCheckPoint()
    {
        if (!_checkPointManager)
        {
            _checkPointManager = FindObjectOfType<CheckPointManager>();
        }
        Invoke(nameof(Revive), 1.5f);
    }

    private void Revive()
    {
        Restart?.Invoke(_checkPointManager.GetCheckPointPosition());
    }
```

```
    public void SetCheckPoint(CheckPoint checkPoint)
    {
        if (!_checkPointManager)
        {
            _checkPointManager = FindObjectOfType<CheckPointManager>();
        }
        _checkPointManager.SetCheckPoint(checkPoint);
    }
}
```

Лістинг UIManager.cs:

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

public class UIManager : MonoBehaviour
{
    [SerializeField]
    private GameObject _gameOverPanel;
    [SerializeField]
    private Button _restartButton;
    [SerializeField]
    private Animator _blackPanelAnim;
    private FemalePlayerInput _uiInput;
    [SerializeField]
    private PauseMenu _pauseMenu;


    private void Awake()
    {
        _uiInput = new FemalePlayerInput();
        _restartButton.onClick.AddListener(Restart);
        _uiInput.UI.Pause.started += onPause;
    }

    private void onPause(InputAction.CallbackContext obj)
    {
        _pauseMenu.HandleEscape();
    }

    private void Restart()
    {
        GameManager.Instance.RestartFromCheckPoint();
        _gameOverPanel.SetActive(false);
        _blackPanelAnim.SetBool("isPlay", true);
        Invoke(nameof(RemoveBlackPanel), 1f);
```

```
    }

    private void RemoveBlackPanel()
    {
        _blackPanelAnim.SetBool("isPlay", false);
    }
    public void ShowGameOver()
    {
        _gameOverPanel.SetActive(true);
    }

    private void OnEnable()
    {
        _uiInput.UI.Enable();
    }

    private void OnDisable()
    {
        _uiInput.UI.Disable();
    }

}
```

Лістинг MainMenu.cs:

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

public class MainMenu : MonoBehaviour
{
    [SerializeField]
    private Button _newGameButton;
    [SerializeField]
    private Button _continueButton;
    [SerializeField]
    private Button _settingsButton;
    [SerializeField]
    private Button _exitButton;
    [SerializeField]
    private SettingsMenu _settingsMenu;
    [SerializeField]
    private Image _loaderImage;
    [SerializeField]
    private Text _loaderText;
    [SerializeField]
    private GameObject _loaderScreen;
    private string _nextLevel = "Level 1";
```

```
    private void Start()
    {
        _newGameButton.onClick.AddListener(StartGame);
        _continueButton.onClick.AddListener(StartGame);
        _settingsButton.onClick.AddListener(Settings);
        _exitButton.onClick.AddListener(ExitGame);
        if(!SavesSystem.Instance.hasSaves)
        {
            _continueButton.gameObject.SetActive(false);
        }
        else
        {
            _nextLevel = SavesSystem.Instance.LoadGame();
        }
    }

    private void StartGame()
    {
        StartCoroutine(LoadScreen());
    }

    private void Settings()
    {
        _settingsMenu.gameObject.SetActive(true);
    }

    private void ExitGame()
    {
        Application.Quit();
    }

    private IEnumerator LoadScreen()
    {
        AsyncOperation asyncOperation = SceneManager.LoadSceneAsync(_nextLevel);
        asyncOperation.allowSceneActivation = false;
        float loadingProgress = 0;
        _loaderImage.fillAmount = loadingProgress;
        _loaderScreen.SetActive(true);
        while (!asyncOperation.isDone)
        {
            _loaderImage.fillAmount = loadingProgress/100;
            _loaderText.text = loadingProgress + "%";
            loadingProgress += 1;
            if (loadingProgress > 90)
            {
                asyncOperation.allowSceneActivation = true;
            }
            yield return null;
        }
    }
}
```

Лістинг SettingsMenu.cs:

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

public class SettingsMenu : MonoBehaviour
{
   [SerializeField]
   private Button _closeButton;
   [SerializeField]
   private SettingsMenu _settingsMenu;

   private void Start()
   {
      _closeButton.onClick.AddListener(CloseSettingsMenu);
   }

   private void CloseSettingsMenu()
   {
      _settingsMenu.gameObject.SetActive(false);
   }
}
```

Лістинг PauseMenu.cs:

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

public class PauseMenu : MonoBehaviour
{
   [SerializeField]
   private GameObject _pauseMenu;
   [SerializeField]
   private Button _continueButton;
   [SerializeField]
   private Button _settingsButton;
   [SerializeField]
   private SettingsMenu _settingsMenu;
   [SerializeField]
   private Button _exitButton;

   private void Start()
   {
      _continueButton.onClick.AddListener(Resume);
```

```
        _settingsButton.onClick.AddListener(Settings);
        _exitButton.onClick.AddListener(ExitGame);
    }

    public void HandleEscape()
    {
        if (GameManager.Instance.GameIsPaused)
        {
            Resume();
        }
        else
        {
            Pause();
        }
        GameManager.Instance.GameIsPaused = !GameManager.Instance.GameIsPaused;
    }

    private void Resume()
    {
        Time.timeScale = 1;
        _pauseMenu.gameObject.SetActive(false);
    }

    private void Pause()
    {
        Time.timeScale = 0;
        _pauseMenu.gameObject.SetActive(true);
    }

    private void Settings()
    {
        _settingsMenu.gameObject.SetActive(true);
    }

    private void ExitGame()
    {
        Application.Quit();
    }
}
```

Лістинг SavesSystem.cs:

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

public class SavesSystem : MonoBehaviour
{
    public static SavesSystem Instance;
```

```
    private string _yourSavePass = "progress";
    public bool hasSaves;

    private void OnEnable()
    {
        if (Instance != null)
        {
            Destroy(Instance.gameObject);
        }

        Instance = this;
        DontDestroyOnLoad(gameObject);
    }

    private void Awake()
    {
        string level = PlayerPrefs.GetString(_yourSavePass);
        Debug.LogError(level);
        if(!string.IsNullOrEmpty(level))
        {
            hasSaves = true;
        }
    }

    public void SaveGame(string levelName)
    {
        PlayerPrefs.SetString(_yourSavePass, levelName);
        PlayerPrefs.Save();
    }

    public string LoadGame()
    {
        return PlayerPrefs.GetString(_yourSavePass);
    }
}
```

Лістинг CheckPoint.cs:

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

public class CheckPoint : MonoBehaviour
{
    private void OnTriggerEnter(Collider other)
    {
        GameManager.Instance.SetCheckPoint(this);
        SavesSystem.Instance.SaveGame(SceneManager.GetActiveScene().name);
    }
```

```
}
```

Лістинг CheckPointManager.cs:

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

public class CheckPointManager : MonoBehaviour
{
    private CheckPoint _checkPoint;
    public void SetCheckPoint(CheckPoint checkPoint)
    {
        _checkPoint = checkPoint;
    }

    public Vector3 GetCheckPointPosition()
    {
        return _checkPoint.transform.position;
    }
}
```

Лістинг DestructibleWall.cs:

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

public class DestructibleWall : MonoBehaviour
{
    private bool _playerInTrigger;
    private FemalePlayerInput playerInput;

    private void Awake()
    {
        playerInput = new FemalePlayerInput();
        playerInput.FemaleControls.Hit.started += GetHit;
    }

    private void GetHit(InputAction.CallbackContext context)
    {
        if (_playerInTrigger && context.ReadValueAsButton())
        {
            Invoke(nameof(DestroyWall), 1);
        }
    }
```

```
    private void DestroyWall()
    {
        for (int i = 0; i < transform.childCount; i++)
        {
            Rigidbody rb = transform.GetChild(i).GetComponent<Rigidbody>();
            rb.isKinematic = false;
            rb.WakeUp();
            rb.AddForce(Vector3.right * 250);
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            _playerInTrigger = true;
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            _playerInTrigger = false;
        }
    }

    private void OnEnable()
    {
        playerInput.FemaleControls.Enable();
    }

    private void OnDisable()
    {
        playerInput.FemaleControls.Disable();
    }
}
```

Лістинг Health.cs:

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

public class Health : MonoBehaviour
{
    private bool _isDamaged;
    [SerializeField]
```

```
private float _health;
[SerializeField]
private Image _healthUI;
public bool isDead;
[SerializeField]
private Animator _animator;
private Transform _ui;
private Transform _camera;
[SerializeField]
private bool _lookAtCamera;
[SerializeField]
private bool _isPlayer;
private float _multiplier;
private float _startHealth;

private void Start()
{
    _startHealth = _health;
    _ui = _healthUI.transform.parent;
    _camera = Camera.main.transform;
    _multiplier = 100 / _health;
    if (_isPlayer)
    {
        GameManager.Instance.Restart += Restart;
    }
}

private void Restart(Vector3 position)
{
    PeriodicalDamage periodicalDamage = GetComponent<PeriodicalDamage>();
    if (periodicalDamage)
    {
        Destroy(periodicalDamage);
    }
    isDead = false;
    _animator.SetBool("isDead", isDead);
    _ui.gameObject.SetActive(true);
    SetColliders(isDead);
    transform.position = position;
    GetComponent<CharacterController>().enabled = true;
    GetComponent<CharacterControl>().enabled = true;
    _health = _startHealth;
    _healthUI.fillAmount = 1;
}

private void Update()
{
    if (_lookAtCamera)
    {
        _ui.LookAt(_camera);
    }
}
```

```
private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Weapon") && !_isDamaged)
    {
        Weapon weapon = other.GetComponent<Weapon>();
        if (weapon.owner != transform)
        {
            TakeDamage(weapon.Hit());
        }
        Debug.LogError(other.name);
    }
}

public void TakeDamage(float damage)
{
    _health -= damage;
    _healthUI.fillAmount -= 1 / damage * _multiplier;
    if (_health > 0)
    {
        _isDamaged = true;
        _animator.SetBool("isDamage", _isDamaged);
        Invoke(nameof(ReadyForAttack), 1);
    }
    else
    {
        Death();
    }
}

private void ReadyForAttack()
{
    _isDamaged = false;
    _animator.SetBool("isDamage", _isDamaged);
}

private void Death()
{
    isDead = true;
    _animator.SetBool("isDead", isDead);
    _ui.gameObject.SetActive(false);
    SetColliders(isDead);
    if (_isPlayer)
    {
        GetComponent<CharacterControl>().enabled = false;
        GameManager.Instance.GameOver();
    }
}
public void RestoreHealth()
{
    _health = _startHealth;
    _healthUI.fillAmount = 1;
```

```
    }

    private void SetColliders(bool isActive)
    {
        Collider[] colliders = GetComponents<Collider>();
        for (int i=0; i<colliders.Length; i++)
        {
            colliders[i].enabled = isActive;
        }
    }

    public bool isAlive()
    {
        return !isDead;
    }

    private void OnDestroy()
    {
        if (_isPlayer)
        {
            GameManager.Instance.Restart -= Restart;
        }
    }
}
```

Лістинг Lever.cs:

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

public class Lever : MonoBehaviour
{
    private bool _playerInTrigger;
    private FemalePlayerInput playerInput;

    [SerializeField]
    private Animator _animator1;
    [SerializeField]
    private Animator _animator2;
    [SerializeField]
    private string _animationName1;
    [SerializeField]
    private string _animationName2;

    private void Awake()
    {
        playerInput = new FemalePlayerInput();
```

```
        playerInput.FemaleControls.Action.started += UseLever;
    }

    private void UseLever(InputAction.CallbackContext context)
    {
        if (_playerInTrigger && context.ReadValueAsButton())
        {
            _animator1.SetBool(_animationName1, true);
            Invoke(nameof(OpenDoor), 1);
        }
    }
    private void OpenDoor()
    {
        _animator2.SetBool(_animationName2, true);
    }
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            _playerInTrigger = true;
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            _playerInTrigger = false;
        }
    }

    private void OnEnable()
    {
        playerInput.FemaleControls.Enable();
    }

    private void OnDisable()
    {
        playerInput.FemaleControls.Disable();
    }
}
```

Лістинг LockPicking.cs:

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

public class Lockpicking : MonoBehaviour
```

```
{
    private bool _playerInTrigger;
    private FemalePlayerInput playerInput;

    [SerializeField]
    private Animator _animator;
    [SerializeField]
    private string _animationName;
    private void Awake()
    {
        playerInput = new FemalePlayerInput();
        playerInput.FemaleControls.Action.started += OpenLock;
    }

    private void OpenLock(InputAction.CallbackContext context)
    {
        if (_playerInTrigger && context.ReadValueAsButton())
        {
            _animator.SetBool(_animationName, true);
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            _playerInTrigger = true;
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            _playerInTrigger = false;
        }
    }

    private void OnEnable()
    {
        playerInput.FemaleControls.Enable();
    }

    private void OnDisable()
    {
        playerInput.FemaleControls.Disable();
    }
}
```

Лістинг MovingPlatform.cs:

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

public class MovingPlatform : MonoBehaviour
{
    [SerializeField]
    private Transform _target1;
    [SerializeField]
    private Transform _target2;
    private Transform _currentTarget;
    [SerializeField]
    private Rigidbody _rigidbody;
    [SerializeField]
    private float _speed;

    private void Start()
    {
        transform.position = _target1.position;
        _currentTarget = _target2;
    }
    private void FixedUpdate()
    {
        Move();
    }
    private void Move()
    {
        Vector3 direction = (_currentTarget.position - transform.position).normalized;
        _rigidbody.MovePosition(transform.position + direction * Time.fixedDeltaTime * _speed);
        if (Vector3.Distance(transform.position, _currentTarget.position) < 1)
        {
            ChangeTarget();

        }
    }

    private void ChangeTarget()
    {
        if (_currentTarget.Equals(_target1))
        {
            _currentTarget = _target2;
        }
        else
        {
            _currentTarget = _target1;
        }
    }
```

```
    private void OnTriggerEnter(Collider other)
    {

    }

    private void OnTriggerStay(Collider other)
    {
        if (other.tag.Equals("Player"))
        {
            other.GetComponent<CharacterControl>().externalMove = _rigidbody.velocity;
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (other.tag.Equals("Player"))
        {
            other.GetComponent<CharacterControl>().externalMove = Vector3.zero;
        }
    }
}
```

Лістинг Obstacle.cs:

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

public class Obstacle : MonoBehaviour
{
    [SerializeField]
    private float _damage;
    [SerializeField]
    private float _pause;

    private void OnTriggerEnter(Collider other)
    {
        Health health = other.GetComponent<Health>();
        if (health)
        {
            PeriodicalDamage periodicalDamage =
other.gameObject.AddComponent<PeriodicalDamage>();
            periodicalDamage.StartDealingDamage(health, _damage, _pause);
        }
    }

    private void OnTriggerExit(Collider other)
    {
        PeriodicalDamage periodicalDamage = other.GetComponent<PeriodicalDamage>();
        if (periodicalDamage)
```

```
        {
            Destroy(periodicalDamage);
        }
    }
}
```

Лістинг PeriodicalDamage.cs:

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

public class PeriodicalDamage : MonoBehaviour
{
    private Health _health;
    private float _damage;
    private float _pause;

    public void StartDealingDamage(Health health, float damage, float pause)
    {
        _health = health;
        _damage = damage;
        _pause = pause;

        StartCoroutine(DealDamage());
    }
    private IEnumerator DealDamage()
    {
        while (!_health.isDead)
        {
            _health.TakeDamage(_damage);
            yield return new WaitForSeconds(_pause);
        }
    }

}
```

Лістинг Potion.cs:

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

public class Potion : MonoBehaviour
{
    private void OnTriggerEnter(Collider other)
    {
        Health health = other.GetComponent<Health>();
```

```
        if (health)
        {
            health.RestoreHealth();
            Destroy(gameObject);
        }
    }

}
```

Лістинг PressableButton.cs:

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

public class PressableButton : MonoBehaviour
{
    private bool _playerInTrigger;

    [SerializeField]
    private Animator _animator1;
    [SerializeField]
    private Animator _animator2;
    [SerializeField]
    private string _animationName1;
    [SerializeField]
    private string _animationName2;
    [SerializeField]
    private bool _isPlatform;
    [SerializeField]
    private MovingPlatform _platform;

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            _playerInTrigger = true;
            _animator1.SetBool(_animationName1, true);
            Invoke(nameof(OpenDoor), 1);
        }
    }
    private void OpenDoor()
    {
        if(!_isPlatform)
        {
            _animator2.SetBool(_animationName2, true);
        }
        else
        {
```

```
            _platform.enabled = true;
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Player") || other.CompareTag("MovingBox"))
        {
            _playerInTrigger = false;
        }
    }

}
```

Лістинг PushBox.cs:

```
using UnityEngine;

public class PushBoxScript : MonoBehaviour
{
    [SerializeField]
    private float _force;

    private void OnControllerColliderHit(ControllerColliderHit hit)
    {
        if(!hit.gameObject.tag.Equals("MovingBox"))
        {
            return;
        }
        Rigidbody rigidBody = hit.collider.attachedRigidbody;

        if (rigidBody != null && transform.position.y < hit.gameObject.transform.position.y)
        {
            Vector3 direction = hit.gameObject.transform.position - transform.position;
            direction.y = 0;
            direction.Normalize();
            rigidBody.AddForceAtPosition(direction * _force, transform.position, ForceMode.Impulse);
        }
    }
}
```

Лістинг Weapon.cs:

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

public class Weapon : MonoBehaviour
{
    [SerializeField]
    private float _damage;
    public Transform owner;

    public float Hit()
    {
        return _damage;
    }
}
```

Лістинг Enemy.cs:

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

public class Enemy : MonoBehaviour
{
    [SerializeField]
    private Transform _target1;
    [SerializeField]
    private Transform _target2;
    private Transform _currentTarget;
    [SerializeField]
    private Rigidbody _rigidbody;
    [SerializeField]
    private float _speed;
    [SerializeField]
    private Animator _animator;
    private bool _seeEnemy;
    [SerializeField]
    private Health _health;
    private Transform _enemy;

    private void Start()
    {
        SelectRandom();
        _animator.SetBool("isWalk", true);
    }

    private void SelectRandom()
```

```
{
    int random = Random.Range(0, 2);
    if (random == 0)
    {
        _currentTarget = _target1;
    }
    else
    {
        _currentTarget = _target2;
    }
}

private void FixedUpdate()
{
    if (_currentTarget && _health.isAlive())
    {
        if (!_enemy)
        {
            Move(false);

        }
        else
        {
            Move(true);
        }
    }
}

private void Move(bool isAttack)
{
    if (!isAttack || Vector3.Distance(transform.position, _currentTarget.position) > 2)
    {
        _animator.SetBool("isWalk", true);
        _animator.SetBool("isAttack", false);
        Vector3 lookDirection = _currentTarget.transform.position;
        lookDirection.y = transform.position.y;
        transform.LookAt(lookDirection);
        Vector3 direction = (_currentTarget.position - transform.position).normalized;
        direction.y = 0;
        direction.z = 0;
        _rigidbody.MovePosition(transform.position + direction * Time.fixedDeltaTime * _speed);
    }
    if (Vector3.Distance(transform.position, _currentTarget.position) < 1 && !isAttack)
    {
        ChangeTarget();
    }
    else if (Vector3.Distance(transform.position, _currentTarget.position) < 2 && isAttack)
    {
        _animator.SetBool("isWalk", false);
        _animator.SetBool("isAttack", true);
    }
}
```

```
    private void ChangeTarget()
    {
        if (_currentTarget.Equals(_target1))
        {
            _currentTarget = _target2;
        }
        else if (_currentTarget.Equals(_target2))
        {
            _currentTarget = _target1;
        }
        else
        {
            SelectRandom();
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
          _enemy = other.transform;
           _currentTarget = _enemy;
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Player"))
        {
          // _enemy = null;
          // SelectRandom();
        }
    }
}
```

ДОДАТОК Б

ВІДГУК
керівника економічного розділу
на кваліфікаційну роботу бакалавра
на тему:
"Особливості 3D моделювання та анімації для ігор на прикладі створення ігрового застосунку за допомогою Unity 3D"
студентки групи 122-18-2 Волошиної Валерії Вікторівни

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 Волошина.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота Волошина.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Волошина.rar	Архів. Містить коди програми і скомпільовану програму
Презентація	
Волошина.pptx	Презентація кваліфікаційної роботи