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 *Дубіни Єгора Сергійовича*
(ПІБ)

академічної групи *122-20-4*
(шифр)

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

освітньої програми Комп'ютерні науки
(назва освітньої програми)

на тему: *Розробка ігрового додатку на рушії Unity з використанням мови програмування C#*

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

Дніпро
2024

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

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

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2024 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-20-4* (група) Дубіни Єгора Сергійовича (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка ігрового додатку на рушії Unity з використанням мови програмування C#*

затверджена наказом ректора НТУ «ДП» від *23.05.2024 р.* № *469-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***	***20.05.2024.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***03.06.2024 р.***

Завдання видав _____ *доц. Спірінцев В.В*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Дубіна Є.С*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2024 р.*

Термін подання кваліфікаційної роботи до ЕК: *10.06.2024 р.*

РЕФЕРАТ

Пояснювальна записка: 111 с., 53 рис., 3 дод., 40 джерел.

Об'єкт розробки: ігровий застосунок для інтерактивного розвитку навичок користувача.

Мета кваліфікаційної роботи: розробка програмного забезпечення на платформі ігрового рушія Unity.

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

У першому розділі проаналізовано предметну галузь, визначено актуальність завдання та призначення розробки, сформульовано постановку завдання, зазначено вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі проаналізовані наявні рішення, обрано платформу для розробки, виконано проектування і розробку системи віртуального середовища, описана робота системи, алгоритм і структура його функціонування, а також виклик та завантаження додатку, визначено вхідні і вихідні дані, охарактеризовано склад параметрів технічних засобів.

В економічному розділі визначено трудомісткість розробленого застосунку, проведений підрахунок вартості роботи по створенню додатку та розраховано час на його створення.

Практичне значення полягає у створенні тривимірного застосунку, який дозволить керувати ігровим процесом, та взаємодіяти з ігровими об’єктами, а також забезпечує реалізацію ігрових механік, звукових та візуальних ефектів.

Актуальність розробки програмного продукту полягає в попиті готових рішень в ігровій індустрії, а також постійному розвитку цієї галузі, що створює платформу для розробки інноваційних рішень.

Список ключових слів: UNITY, ТРИВИМІРНИЙ, КЛАС, КОМПОНЕНТ, ОБ’ЄКТ, МЕХАНІКА.

ABSTRACT

Explanatory note: 111 pages, 53 figures, 3 appendices, 40 sources.

Object of development: a gaming application for interactive user skills development.

The purpose of the diploma project: development of software on the Unity game engine.

The introduction includes an analysis and the current state of the problem, specifies the purpose of the qualification work and its application area, provides justification for the topic's relevance, and clarifies the task statement.

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

In the second section, the available solutions are analyzed, a platform for development is selected, the design and development of a gaming application for interactive user skills development is carried out, the operation of the system, the algorithm and structure of its functioning, as well as the call and loading of the application are described, the input and output data are determined, and the composition of the parameters of the technical means is characterized.

In the economic section, the labor intensity of the developed application is determined, the cost of application development work is calculated, and the time required for its creation is estimated.

The practical value lies in creating a three-dimensional application that allows controlling the gaming process, interacting with game objects, and also ensures the implementation of gaming mechanics, sound, and visual effects.

The relevance of the development of a gaming application for interactive user skills development is beyond doubt and is determined by the great demand for such system.

List of keywords: GAMING APPLICATION, UNITY, INTERACTIVE, SKILLS DEVELOPMENT, ALGORITHM, STRUCTURE.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ - Програмне забезпечення

3D - Тривимірний простір

2D - Двомірний простір

PEGI - Pan European Game Information

ESRB - Entertainment Software Rating Board

ОС - Операційна система

RPG - Role-Playing Game

PC - Персональний комп’ютер

SSAO - Screen space ambient occlusion

VFX - Visual effects

IDE - Integrated Development Environment

ЗМІСТ

ВСТУП

Ігрова індустрія пройшла еволюційний шлях з часів свого виникнення. Завдяки новим досягненням у сфері технологій, графіки та ігрового дизайну, вона привертає все більше шанувальників та розширює свої можливості на міжнародному ринку. Цей розвиток індустрії є результатом стрімкого технологічного прогресу, що перетворив ігри на захоплююче досвідне поле, де гравці можуть відчути себе частиною фантастичного світу та зануритися у ігрове середовище, що дозволяє їм не лише проводити час, а й досліджувати, вирішувати завдання та розвивати свої навички.

Об’єктом дослідження є ігровий застосунок для інтерактивного розвитку навичок користувача.

Мета даної кваліфікаційної роботи: розробка ігрового додатку на рушії Unity з використанням мови програмування C#.

Актуальність застосунку полягає в попиті готових рішень в ігровій індустрії, а також постійному розвитку цієї галузі, що створює платформу для розробки інноваційних рішень.

Створення даного продукту сприятиме розвитку наступних якостей у користувача:

- вирішення складних завдань, і, як наслідок, розвиток критичного та стратегічного планування;
- оцінка варіативності дій та прийняття рішень;
- здатність створення нових ідей та концепцій;
- аналіз ситуацій та пошук логічних зв’язків.

Створення додатку з використанням рушія Unity є технологічним рішенням на ринку ігрової індустрії. Таким чином, завдяки популярності та розвитку даної платформи, з’являється перелік інструментів для створення сучасних рішень.

Тому, для досягнення поставленої мети, необхідно вирішити перелік наступних задач:

– розглянути існуючі проекти в галузі ігрової індустрії та жанри для виявлення їх переваг та недоліків;

– визначити перелік інструментів та компонентів, які використовуватимуться в додатку, зокрема обрати математичні методи для взаємодії з об’єктами в просторі;

– розробити віртуальне середовище та встановити порядок взаємодії користувача з додатком, описати його структуру наповнення і алгоритми;

– з’ясувати і зазначити системні та апаратні вимоги до пристроїв користувачів;

– розрахувати трудомісткість та вартість на створення програмного засобу.

Практичне значення полягає у створенні тривимірного застосунку, який дозволить керувати ігровим процесом, та взаємодіяти з ігровими об’єктами, а також забезпечує реалізацію ігрових механік, звукових та візуальних ефектів.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

1.1.1. Аналіз та порівняння рушіїв ігрових додатків

Для порівняння з обраною платформою створення ігрових додатків, було обрано популярний рушій Unreal Engine [1], який є потужною платформою для розробки рішень, переважною та основною мовою програмування якого є C++.

Unity [2] та Unreal Engine є ключовими компонентами у процесі розробки ігор, оскільки поєднують різноманітні елементи, такі як звук, графіка та штучний інтелект, а також структуру гри.

Ці двигуни створюють середовище для розробки програмного забезпечення (рис. 1.1-1.2), що допомагає дизайнерам створювати власні відеоігри. Обидва забезпечують легку можливість розробки ігор для комп’ютерів, консолей (таких як Xbox і PS4), мобільних платформ, таких як iOS і Android, а також для операційних систем: Windows, Mac, Linux..

Механізми візуалізації графіки у форматах 2D і 3D, фізичне управління рухом, звукові ефекти, сценарії гри, анімація, системи штучного інтелекту, мережеві можливості, потокове передавання даних, оптимізація роботи з пам'яттю, інструменти віртуальної реальності та підтримка як 2D, так і 3D ігор - усе це є важливими компонентами даних платформ.

Ключова різниця між двома рушіями полягає в мові програмування платформ. Так, Unity використовує C# у редакторі та в додаткових плагінах. Unreal Engine використовує мову C++, але надає можливості використання комбінації з технологією Blueprint [3].

Unity відзначається простотою використання та швидкістю розробки ігрових застосунків. Компонентна архітектура двигуна дозволяє ефективно

створювати сценарії на C#. Окрім того, рушій має велику спільноту, форуми де можливо отримати додаткову інформацію, що спрошує процес розробки.

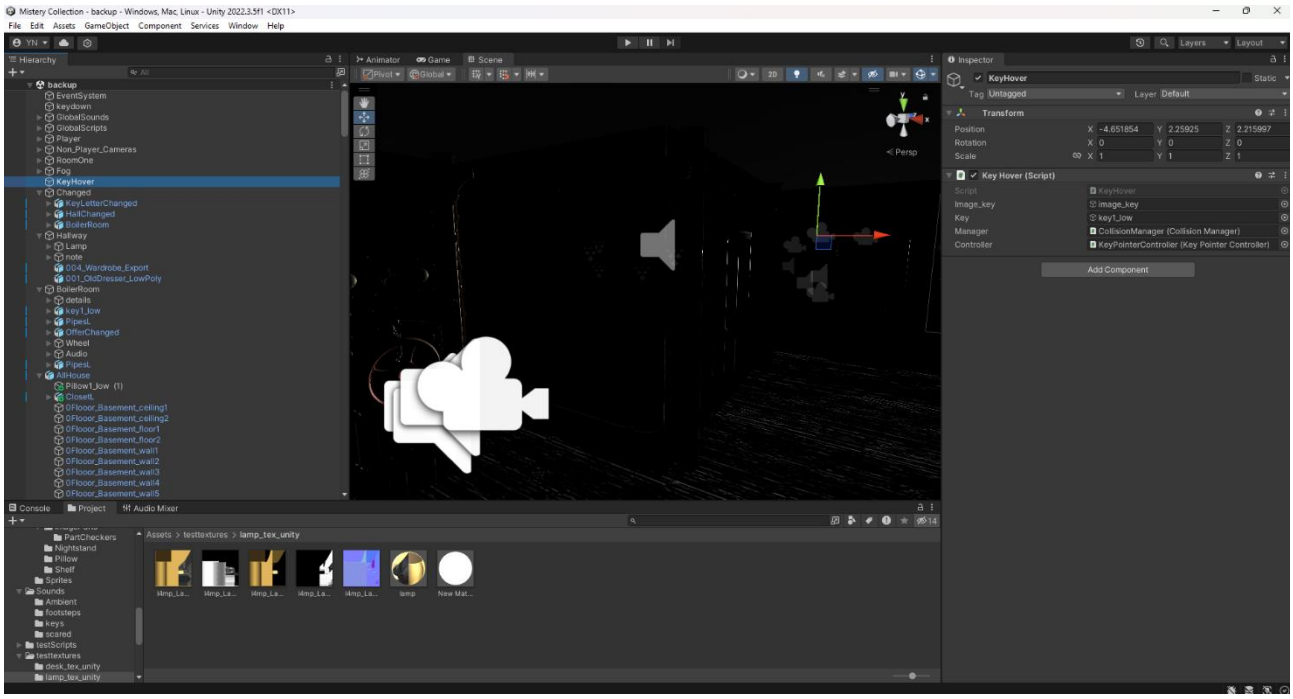

Рис. 1.1. Інтерфейс редактора Unity

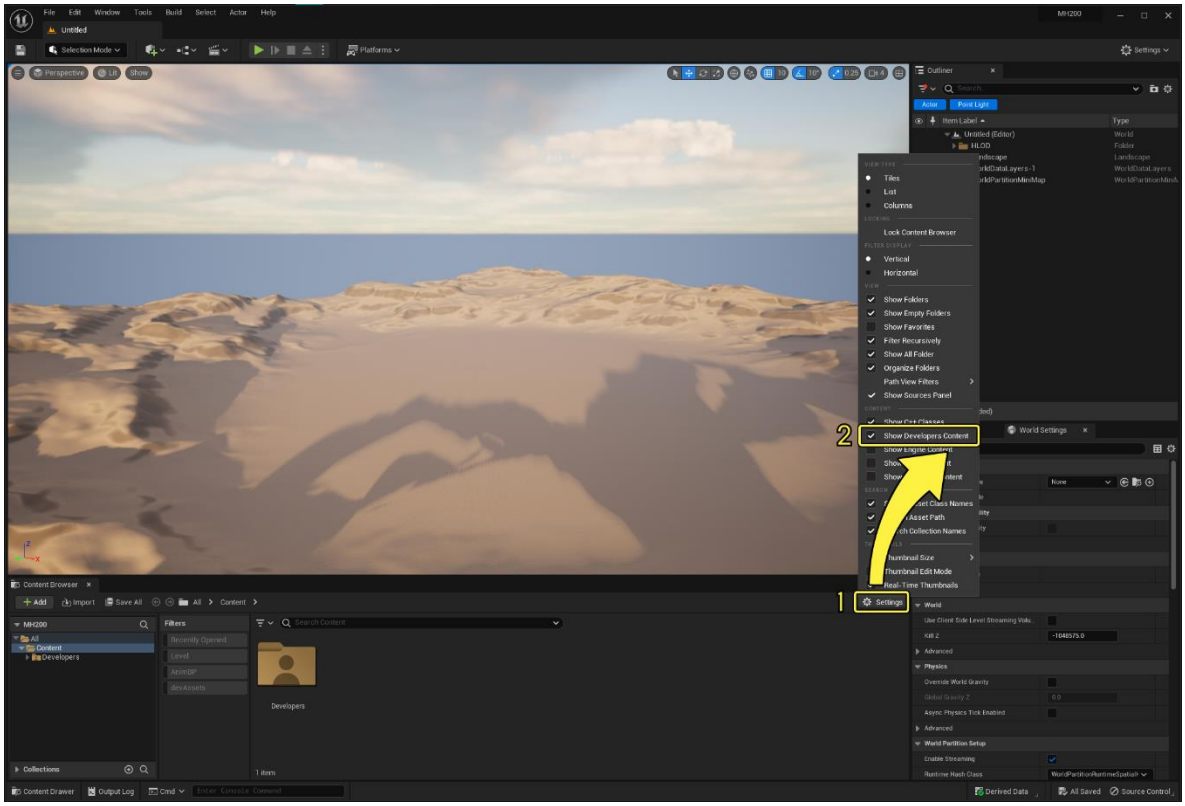

Рис. 1.2. Інтерфейс редактора Unreal Engine

Архітектура даного інструменту є компонентною [4], тому основа сцени – це ігрові об’єкти, які служать контейнерами для цих компонентів. Є декілька основних компонентів, а саме: Transform, Mesh Renderer, Material [5] що реалізують відповідну функціональність для управління елементами сцени.

Одною з важливих концепцій в Unity є префаб [6]. Даний елемент дозволяє налаштувати ігрові об’єкти з компонентами та зв’язками як єдину одиницю. Він зберігає усі параметри, і надає можливості створення точної копії під час виконання, що спрощує повторне використання складних конструкцій.

Однак Unity відстає за іншими ігровими движками у плані візуальних ефектів з моменту свого початку. З випуском версії Unity 5 був зроблений значний крок вперед за допомогою шейдерів Unity Standard, фізичного затінення та методів освітлення на основі зображень. Незважаючи на це, у порівнянні з Unreal або, Unity все ще має певні недоліки в графічному плані.

Unreal Engine, в свою чергу, має Blueprint, який є системою візуальних сценаріїв. Даний інструмент дозволяє створювати елементи ігрового процесу через інтерфейс, на основі вузлів. Цей функціонал полегшує роботу дизайнерів, через те, що стає можливим налаштовувати поведінку об’єкта без необхідності писати код вручну. Ці об’єкти мають властивість повторного використання, що дозволяє зручне використання складних конструкцій.

Аналізуючи джерела щодо порівняння системи освітлення рушіїв [7], можна прийти до висновку, що розмір проекції є значним у Unreal Engine, але не впливає на Unity, як показано на рис. 1.3. Unity підтримує постійний час кадру, тоді як Unreal Engine прискорюється майже втричі. Хоча обидва двигуни використовують відкладене затінення, Unreal Engine має кращу оптимізацію порівняно з Unity.

Однак, Unreal Engine має потужну і більш розвинену систему освітлення, яка потребує ресурси системи й більшої уваги до оптимізації.

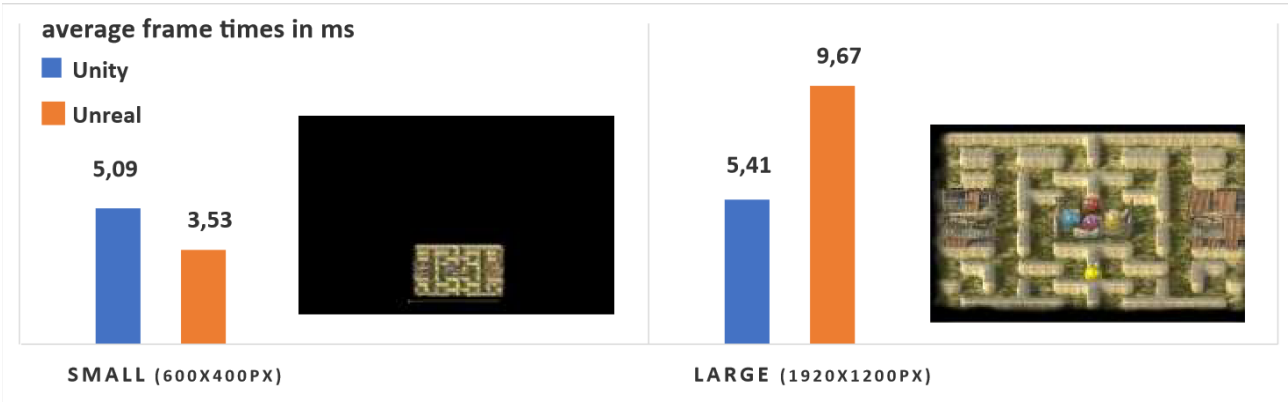

Рис. 1.3. Вплив розміру проекції на частоту кадрів

На рис. 1.3 можна побачити аналіз середнього часу кадру, заснований на контрольному показнику. Початковий час кадру в Unity є меншим, ніж в Unreal, але зростає приблизно на 4 мс з кожним новим складним об’єктом, незалежно від типу обладнання (ПК або ноутбук).

Unreal стартує з 10 мс для одного об’єкту на ПК, це вдвічі більше, ніж в Unity, але показник залишається стабільним навіть при збільшенні кількості лабіринтів, іноді навіть трохи знижуючись. Це свідчить про те, що Unreal оптимізовано для потужних систем і складних сценаріїв. Кожен об’єкт містить приблизно 1,2 мільйона вершин, тому версія 7 включає понад 8 мільйонів вершин. Для Unity це призводить до значного уповільнення (у шість разів повільніше, ніж з одним лабіринтом), тоді як для Unreal це не становить проблему. Причини таких результатів полягають в тому, що Unity динамічно пакує виклики малювання в реальному часі, тоді як Unreal пакує статичні сітки перед компіляцією, що робить його більш ефективним для великих сцен. Іншою причиною є каскадні тіні: Unity підтримує їх починаючи з версії 5.6, тоді як версія 5.3.3, яку я використовував, цього не підтримує. Це також впливає на продуктивність Unity при розрахунку високоякісних тіней на великій відстані.

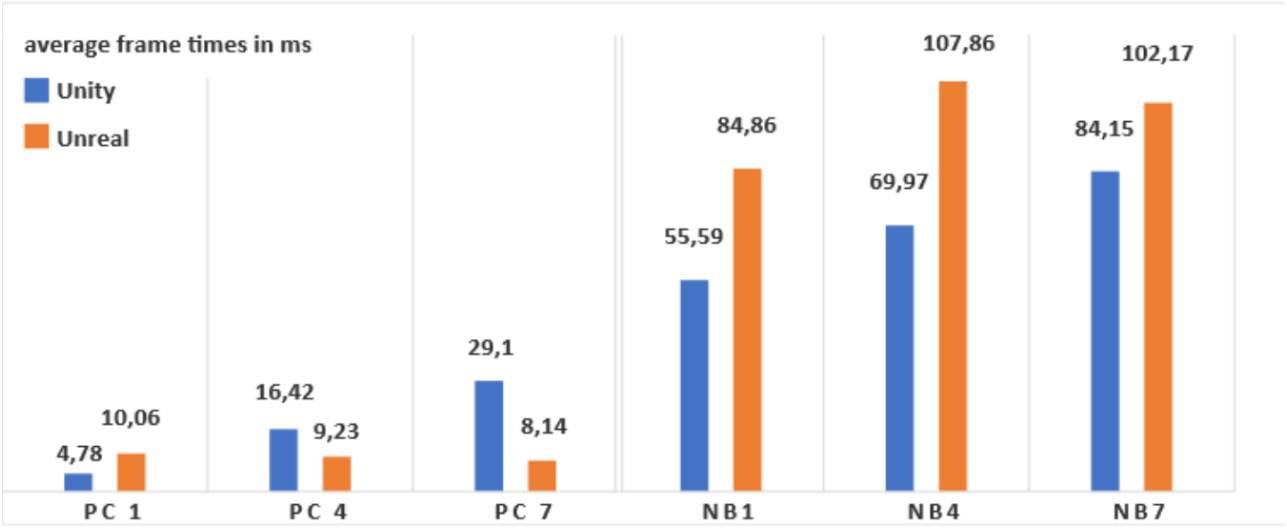

Рис. 1.4. Час кадрів на ПК і ноутбуці. Версії бенчмарка (1,2 - 8,4 млн вершин)

В цілому, перевага Unity полягає у значній спільності розробників, що порівняно з Unreal надає можливості створювати продукти, та отримати підтримку з приводу проблем під час розробки. Проекти на Unity можуть бути набагато легшими для апаратної частини системи, коли Unreal Engine використовується для більш масштабних продуктів і вимагає більшої уваги до розробника в плані оптимізації програмного коду, що робить його менш доступним в використанні.

1.1.2. Класифікація та характеристика ігрових жанрів

Враховуючи тематику досліджень кваліфікаційної роботи, розрізняють наступні ігрові жанри: пригоди, стратегії, рольові ігри, платформери, бойовики, головоломки.

Аналіз електронних джерел показує, що найбільш успішними іграми за останні десятиліття є пригоди, виходячи з переліку найвідоміших продуктів за всі часи [8]. Тоді як рольова гра (RPG) [9] представлена другою за рейтингом. Розподіл жанрів вибірки рейтингових ігор зображено на рис. 1.5.

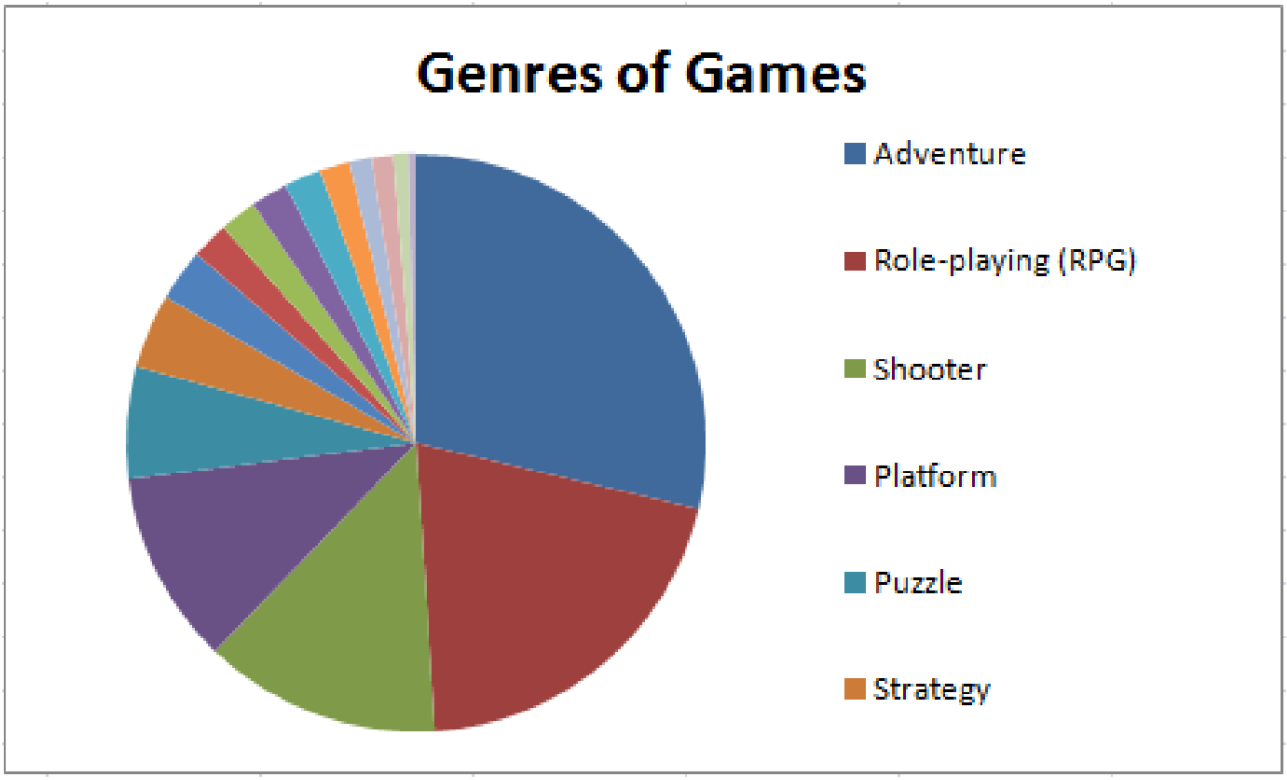

Рис. 1.5. Класифікація ігор за жанрами

Характеризувати розглянути ігрові жанри можна таким чином:

– Пригоди. Відносяться до ігор, якій мають спільну історію, що керує ігровим процесом, та зазвичай асоціюється з квестовою структурою.

– Рольова гра. Уявний світ, де гравці можуть вільно обирати, як досліджувати ігровий світ, з точки зору шляху, яким вони пройдуть, і можуть повернутися до раніше досліджених територій. Обсяг віртуального середовища, потенційно доступний для дослідження, зазвичай великий.

– Шутер. Вирізняється швидкими руховими реакціями гравців, проте характеризується обмеженою здатністю скасовувати вже розпочаті дії.

– Платформер. Персонажі та параметри видно зсередини а не зверху, створюючи графічне відчуття «вгору» та «вниз».

– Головоломка. Концепція полягає в поєднанні шляху та набору шляхів для досягнення ігрової мети.

– Стратегія. Має основні риси складності рішень, які приймає гравець у змодельованому або фантастичному політичному, економічному чи військовому всесвіті.

Окрім цього існують піджанри. Так, стратегія може поділятися на стратегію в реальному часі або покрокову стратегію, а шутер є підрозділом бойовиків.

Одним з жанрів пригод є горор [10], що намагається налякати гравця за допомогою елементів фантастики жахів. Такі ігри пропонують емоційний досвід в умовах логічних загадок та головоломок гри з ціллю просування по сюжету.

Отже, було розглянуто основний перелік класифікацій ігрових жанрів і надано відповідні характеристики до них. Проаналізовані джерела на предмет відмінності та популярності жанрів показав, що «пригоди» користуються найбільшим попитом користувачів, і пропонують квестову структуру та дослідження віртуального миру. Окрім цього, жанр горор є захоплюючим увагу гравця піджанром, що в поєднанні з головоломками сприятиме покращенню ігрового досвіду.

Таким чином, програмне рішення в поточній кваліфікаційній роботі буде спрямовано на пригодницький жанр з головоломками та елементами горору.

1.1.2. Аналіз існуючих програмних рішень

Для аналізу рішень обрано проекти, що відповідають критеріям тематики та жанру, а саме додатки «Myst» [11], «Alan Wake» [12] та «Nancy Drew» [13]. Зображення загального концепту середовищ ігрових застосунків зображено на рис. 1.6. – рис. 1.8.

Рис. 1.6. Загальний вигляд «Myst»

Рис. 1.7. Загальний вигляд «Nancy Drew»

Рис. 1.8. Візуальне представлення «Alan Wake»

Серія ігор «Nancy Drew» є популярною серією квестів, що є детективом, та має наступні характеристики та взаємодію з користувачем:

– квести з фокусом на розв’язання загадок, викриття таємниць, дослідження та взаємодіє з персонажами;

– гравець отримує роль детектива, що вирішує різні таємниці та розслідує злочини;

– реалістичний дизайн з елементами містики та таємниці;

– пошук підказок та доказів, що допомагають просунутися в ігровому процесі;

– наявність інвентарю та підбор предметів, які використовуються для розв’язання головоломок;

– комбінування предметів для взаємодії з іншими об’єктами сцени;

– основний фокус на великій кількості інтерактивних об’єктів, з якими можливо взаємодіяти та досліджувати.

Дана серія ігор має перелік частин франшизи, що складається з 34 ігор . Усі призначені користувачам віком від десяти років, та мають рейтинг «E», тобто «Усі» від ESRB [14]. Ігри виконані в стилі пригодницького жанру, де гравець

керує персонажем у віртуальному середовищі, спілкується з іншими персонажами, розгадує головоломки та розкриває злочини.

Так само, проект «Myst» має візуальний вид з елементами реалізму, а також деталізацію. Даний продукт має пригодницький жанр з елементами головоломок. Пересування виконується за допомогою контролерів, як і взаємодія з об'єктами. Крім того, при фокусуванні на певних тригерах(головоломках) камера має властивість зміщуватися. Тобто, «Myst» надає наступні можливості взаємодії:

– збирання предметів та взаємодія з об'єктами сцени;

– розв'язання головоломок. При цьому камера фокусується на поточній головоломці;

– читання та аналіз текстових матеріалів, таких як журнали та книги, що містять підказки та історію світу. Це важливо для розуміння контексту подальшого ігрового процесу.

Гра «Alan Wake» є пригодницьким трилером з елементами горору, з переліком характеристик та взаємодії з користувачем:

– гравець виконує роль письменника Алана Вейка, який проводить розслідування стикаючись з надприродними явищами;

– поєднання напруженої атмосфери і постійної небезпеки;

– використання світла для боротьби з ворогами, зокрема в темряві ігрової локації;

– дослідження ігрового світу, знаходження підказок та об'єктів;

– комбінування об'єктів і використання їх для розгадування головоломок та взаємодії з навколишнім середовищем.

Детально розглянувши процес геймплею (рис. 1.9) слід зазначити, що серії ігор «Nancy Drew» не є повністю 3D іграми в порівнянні з «Myst». Вони використовують комбінацію двовимірних и тривимірних елементів. Основне середовище та більшість сцен статичні та виконані у 2D, що створюють ілюзію тривимірного простору. Гравець пересувається між статичними екранами, які змінюються по кліку.

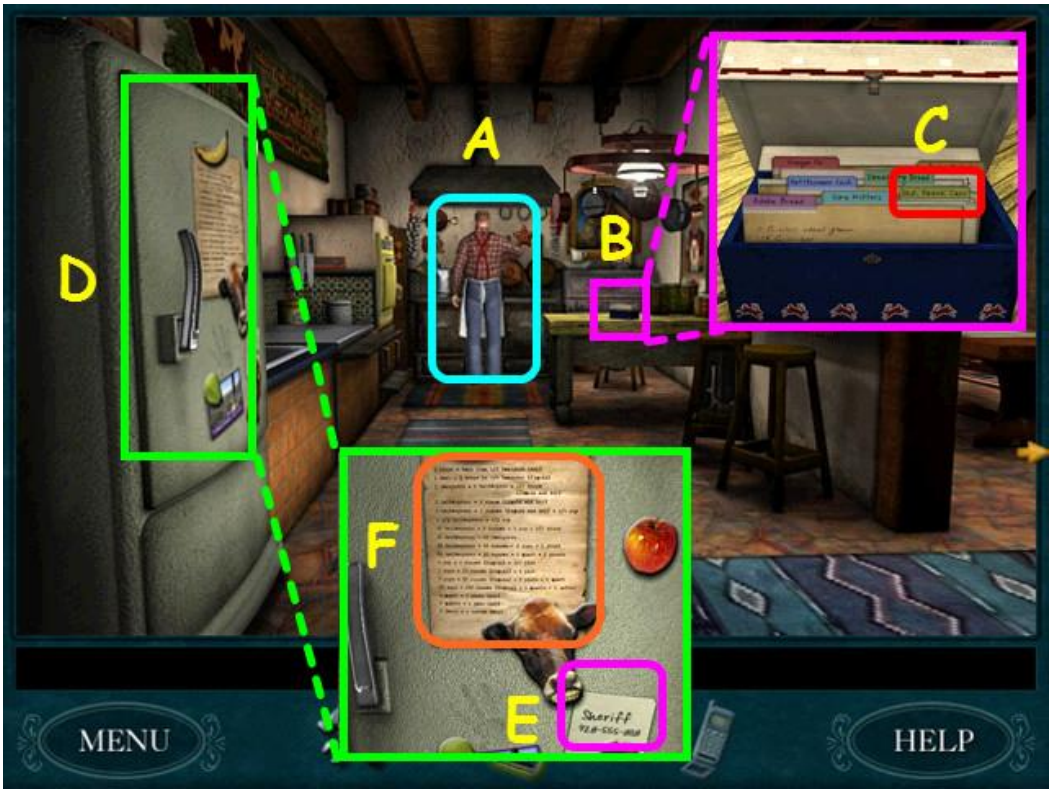

Рис. 1.9. Ігрова сцена «Nancy Drew» та об’єкти для взаємодії

В «Myst» гравець може пересуватися в тривимірному просторі, але існує функціонал переміщення через інтерактивні точки. Таким чином стає можливим потрапити на іншу локацію. Але, більшість зображень мають попередній рендеринг та є 2D зображеннями.

«Alan Wake» надає меншу кількість елементів графічного інтерфейсу (рис. 1.11). Ігровий процес виконується в рамках обмежених коридорних зон, які мають розвиток подій згідно лінійної структури сценарію. Це сприяє створенню відчуття напруженості, характерні для цього ігрового жанру. Гра виконується в тримірному середовище що сприяє більшій ступені імерсивності.

Розглянуті ігри включають розв'язання різноманітних головоломок і загадок. Гравцеві доводиться логічно мислити та шукати ключі та індикатори, щоб просуватися вперед у сюжеті (рис. 1.10).

Рис. 1.10. Головоломка «сейф» в Myst

Рис. 1.11. Інтерфейс та віртуальне середовище Alan Wake

Спираючись на перелік характеристик готових програмних рішень, можна покращити залучення користувачів до ігрового процесу шляхом реалізації правил взаємодії з додатком та візуального представлення, що притаманні жанру горор у поєднанні з елементами пригод і головоломок. Таким чином, прийнято рішення щодо впровадження функціоналу зазначеного жанру у поточній кваліфікаційній роботі. Ці функції були реалізовані наступними способами:

– наділення гри елементами горор, шляхом зменшення кількості світла ігрової сцени, та наділення джерелам світла більшої важливості;

– створення додаткових візуальних ефектів, таких як телепортації, та системи частинок в недоступних для користувача зонах;

– створення тривимірної моделі гравця і методів пересування по ігровій локації ;

– виконання правил взаємодії з ігровим застосунком та послідовності подій (механік).

1.2. Призначення розробки та галузь застосування

У межах даної кваліфікаційної роботи розглядається ігровий додаток «Enchanted Relics».

Основною причиною появи цього продукту є необхідність створення ігрового застосунку, що зверне увагу користувачів в ігровій індустрії, і сприятиме розвиненню гравцем певних якостей.

Призначення створення даного продукту полягає в наступному:

– підвищення концентрації уваги;

– розвиток логічного мислення, а також уваги до деталей;

– укріплення емоційної толерантності в критичних ситуаціях;

– вдосконалення швидкості реакцій на несподівані події;

– розвиток креативного підходу до вирішення проблем;

– зняття емоційного навантаження.

Основна термінологія містить загальні терміни з областей інформаційних технологій, розробки програмного забезпечення та ігрових додатків з використанням рушіїв. Наявні ключові слова: додаток, рушій, персонаж, герой, Unity, розробка, механіка, об’єкт, сцена, ефекти, інтерфейс, дизайн, 3D графіка.

Перелік термінів, що були використані в даній роботі, та необхідні для розуміння роботи застосунку:

– Система частинок – це значення описує елементи та місця ігрової сцени, що імітують субстанції, або частинки вогню та елементів, що генеруються з використанням великої кількості 2D зображень, що є анімованими.

– Безпечні зони – місця ігрової локації, що надають змогу користувачам зорієнтуватися з подальшими діями ігрового процесу без виклику подій віртуальної сцени, що можуть вплинути на персонажа

– Допоміжні знаки – об’єкти сцени, або звукові сигнали що надають поради гравцю для просування по сюжету.

– Ігровий досвід – сукупність вражень та емоцій, які гравець отримує під час гри.

– Імерсивність – ступінь занурення гравця в віртуально створений світ чи середовище, яке створює відчуття реальності та повної присутності.

– Механіки – правила та системи, за допомогою яких функціонує гра. Це включає в себе всі ігрові елементи, взаємодію гравця з оточенням, способи взаємодії та прогресу.

– Горор – жанр мистецтва жахів. В контексті ігрової індустрії позначає піджанр пригод, що представляють надприродні явища.

– Телепортація – процес перенесення об’єкту через простір з одного місця в інше без фізичного переміщення.

– Префаб – об’єкт або група об’єктів, які повторно використовуються в різних сценах, та є основним механізмом повторного використання об’єктів.

– Бенчмарк – вимірювальний інструмент для оцінки продуктивності системи.

– Мапи тіней – процес додавання тіні до тривимірної графіки.

– Рендеринг – процес перетворення тривимірної сцени з світловими характеристиками в двовимірне зображення.

Основними причинами для розробки ігрового застосунку є:

– популярність ігрового жанру, що розробляється;

– навчальні можливості, та дослідження розвитку користувачів;

– розширення аудиторії з ціллю розвитку ринку ігрової індустрії в даному жанровому напрямку;

– комерційній потенціал, що може принести подальший розвиток поточного продукту.

Додаток, що розроблюється, може застосовуватися в наступних областях:

– розваги та ігрова індустрія;

– освітня сфера для інтерактивного навчання;

– система віртуальної реальності для підвищення інтерактивності та відчуття присутності.

1.3. Підстави для розробки

Підставами для розробки (виконання кваліфікаційної роботи) є:

– освітня програма спеціальності 122 «Комп’ютерні науки»;

– навчальний план та графік навчального процесу;

– наказ ректора Національного технічного університету «Дніпровська

– політехніка» №469-с від 23.05.2024;

– завдання на кваліфікаційну роботу на тему «Розробка ігрового додатку на рушії Unity з використанням мови програмування C#»;

1.4. Постановка завдання

Мета цієї роботи: розробка ігрового додатку на рушії Unity з використанням мови програмування C#, яка надасть користувачам можливості імерсивного досвіду з фокусом на емоції та обмеження головного персонажа, що зазначаються в підказках гри при її початку.

Призначенням розробки є формування та покращення концентрації уваги користувача до деталей, розвиток логічного мислення в ситуаціях під тиском а

також створення ігрового досвіду для аудиторії, що цінує імерсивність, головоломки та атмосферу віртуального середовища.

Технічно-економічна сутність завдання: створення гри з реалістичною 3D графікою, орієнтованою на PC-гравців віком від Pegi 16, де головний об’єкт та інформаційна система являє собою різноманітні кімнати, в котрих можливо активувати унікальні квести та пазли.

Система маніпулює різноманітними ігровими елементами, такими як кімнати, предмети, механіки ефектів, персонаж та локації. Таким чином, кожен об’єкт має відповідну структуру та характеристики: предмети можуть бути активованими, маючи тригер подій наведення миші на об’єкт, натискання кнопки в радіусі певної зони, або підібрані, і містити інформацію для можливості взаємодії з іншими об’єктами ігрової сцени.

Розроблений застосунок повинен виконувати наступні функції для гравця:

- пересування героя в тривимірному просторі;
- знаходити предмети та активувати об’єкти;
- реалізовувати механіки атмосфери, такі як страх темряви, ліхтарик;
- мати анімації взаємодії з об’єктами та тригери, що активують квести;
- надавати користувачу доступ до нових локацій ;
- реалізовувати механіки квестів;
- мати тригери та зони дії елементів управління для початку завдань та їх завершення, а також отримання предметів до інвентарю, і їх збереження;
- повідомляти користувача про наступні дії, використовуючи, безпосередньо, об’єкти сцени.

Важливим є створення тригерів та подій, що реагують на дії головного героя, та не порушують структуру та хронологію дій. Виходячи з цього, вхідними даними, які надає гравець використовуючи додаток будуть:

- команди, які надходять від користувача через клавіатуру, мишу, або інші пристрої введення(пересування героя, взаємодія з об’єктами, активація подій);

– поточний стан ігрових об’єктів, включаючи знайдені предметі, активовані механізми та завершені квести;

– збережені дані про прогрес гравця, такі як завершені локації, інформація про зібрані предмети;

– інформація про поточне ігрове середовище, а саме: розташування об’єктів, рівень освітлення;

– ресурсні файли гри: текстури, звуки, моделі, які завантажуються та використовуються в додатку.

задля кращого розуміння ігрового циклу, було розроблено діаграму взаємодії з ігровими сценами та ходом гри (рис. 1.12)

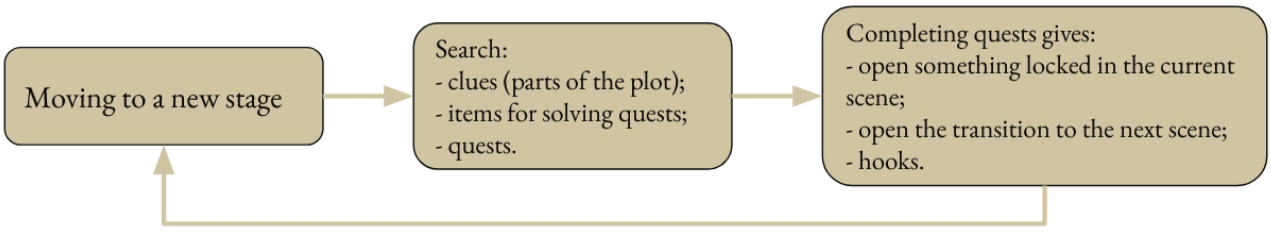

Рис. 1.12. Ігровий цикл

Виходячи з цієї схеми вихідною інформацією системи є дані про:

– прогрес гравця;

– завершені квести та активовані тригери;

– підібрані предмети та доступні квести.

Дана інформація відображатиметься на екрані, і являтиме собою відповідні іконки, позначки, а також додаткові предмети або об’єкти сцени з якими з’явиться можливість маніпулювати.

Існують умови, при яких припиняються розв’язання завдання автоматизованим способом. Такі умови в поточному проекті включають наступні події та ситуації:

– квести або складні ситуації, які неможливо розв’язати за допомогою автоматичних алгоритмів гри, та необхідна підказка для продовження ігрового процесу;

– у разі виникнення неочікуваних подій або змін, які не були передбачені, і вимагають втручання для подальшого розвитку дій.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Для досягнення задачі в програмному застосунку, повинно бути реалізоване наступне:

– можливість керувати головним героєм, пересуваючись у тривимірному просторі за допомогою клавіатури та миші ;

– здатність персонажа ходити, бігати та взаємодіяти з об’єктами;

– наявність механік, що відповідають жанру;

– інтуїтивний зовнішній вигляд інтерфейсу користувача;

– оптимізація ресурсів та управління пам’яттю;

– безпечні алгоритми для збереження продуктивності додатку;

– інтерактивне оточення, де кожен об’єкт може бути досліджений, а деякі з них можуть містити підказки або елементи для вирішення квестів;

– апаратна та програма сумісність з більшістю операційних систем та пристроїв;

– 3D графіка з упором на реалізм та деталізацією, що створює атмосферу пригодницької гри;

– звукове супроводження, що включає звуки середовища та ефекти, що посилюють емоції гравця;

1.5.2. Вимоги до інформаційної безпеки

Проаналізувавши електронну статтю, що присвячена питанню інформаційної безпеки [15], слід зазначити, що інформаційна безпека — це практика захисту інформації шляхом зменшення інформаційних ризиків. Зазвичай це передбачає запобігання або зменшення ймовірності несанкціонованого чи неналежного доступу до даних або незаконного використання, розкриття, порушення, видалення, пошкодження, модифікації, перевірки, запису або знецінення інформації. Перелік пунктів, що вживають в досягненні інформаційної безпеки включає:

– визначення всіх інформаційних ресурсів, пов’язаних активів, можливих загроз, вразливостей та їхніх потенційних наслідків;

– аналіз і оцінка для визначення рівня значущості;

– прийняття рішень, як поводитися з ризиками та впровадження заходів безпеки;

В розробці даного проекту, головною метою безпеки є уникнення неліцензійного використання продукту, та запобігання створення та розповсюдження піратських копій гри. Таким чином, для досягнення даної мети, необхідно реалізувати процес ліцензійних ключів, що включає:

– генерація ліцензійних ключів;

– валідація ліцензійного ключа;

– захист від повторної активації.

Також, однією з вимог безпеки є створення системи античіт, що дозволить відстежувати ті дії гравця, що неможливі при звичайному перебігу подій ігрового процесу, такими пунктами є:

– обмеження швидкості пересування персонажа та її відстеження;

– відстеження послідовності перебігу ігрового процесу.

1.5.3. Вимоги до складу та параметрів технічних засобів

В проекті передбачено, що користувач буде мати технічні засоби, що відповідають нижче зазначеним, які є рекомендованими для коректної роботи застосунку. Таким чином, для запуску та стабільного відображення, а також функціонування додатку, необхідно використовувати:

- Процесор Intel Core i5-2500K з тактовою частотою 3.3 ГГц, або еквівалентний процесор AMD.
- Оперативна пам’ять не менше 8GB.
- Відеокарта NVIDIA GeForce GTX 660 або AMD Radeon HD7950 з підтримкою DirectX11.
- 15 GB вільного місця.

Для швидкого доступу до ресурсів програмного додатку рекомендується використовувати SSD. Також, додаткові вимоги до апаратного забезпечення включають:

- Монітор з роздільною здатністю 1920x1080 або вище.
- Клавіатура, миша, геймпад(за необхідності).
- Аудіо система(навушники або динаміки).

Запуск ігрового додатку передбачає обов’язкову наявність периферійних пристроїв, що були зазначені вище з метою можливості маніпулювання ігровим процесом. Описана системна конфігурація є мінімальною для коректної роботи застосунку

1.5.4. Вимоги до інформаційної та програмної сумісності

Основною програмною вимогою сумісності є підтримка Unity операційними системами, тож перелік буде наступним:

- Windows 8 та вище.
- macOS – не нижче 10.13 High Sierra версії.

– Linux дистрибутиви, що підтримують Unity (Ubuntu 18.04 і вище).

Також, важливим програмним компонентом є наявність Graphics API, а саме DX10, DX11, або DX12 сумісні версії.

Коректна робота програмного застосунку підтримується лише на архітектурі x64 центрального процесору.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Ігровий застосунок в даному проекті орієнтований на ігрові механіки. Таким чином, гравець має вирішувати задачі та квести що надаються в ході ігрового процесу виходячи з правил та можливостей додатку. Тому, призначення додатку буде наступним:

- завантаження сцени;
- створення об’єктів з відповідними сценаріями;
- маніпулювання об’єктами і зміна їх параметрів;
- запуск і створення елементів анімацій згідно з сценарієм;
- надання візуальних ефектів та фізики, а саме: гравітація, зіткнення, рух об’єктів, перетин.
- реалізація звукового супроводження та атмосфери.

Експлуатаційне призначення проекту включає:

- надання квестів та задач;
- підказки до завдань та ігрового процесу;
- керування головним персонажем;
- вибір послідовності дій користувачем;
- динамічна зміна віртуального середовища;
- відображення інтерфейсу користувача;
- система ігрових механік, що визначають перебіг подій ігрового процесу;
- початок та завершення гри.

В цілому, додаток надає елементи управління віртуальним середовищем, та орієнтований на вирішення логічних задач в ході виконання квестів. Також, він вимагає додаткової уваги від користувача через застосування відповідних

механік жанру. Таким чином, дана поведінка системи може сприяти розвиненню таких якостей, як увага до деталей, і особливо вплинути на здатність планування та перебігу подій, розвиток уявлення і творчих навичок, а також формування логіки і створення досвіду в ігровій індустрії.

2.2. Опис застосованих математичних методів

В ході розробки були використані наступні математичні методи:

Перлін шум (Perlin Noise) [16]. Даний метод використано для генерації випадкового плавного значення, що моделює природне мерехтіння світла. Це використано в місцях ігрової мапи, де існує джерело світла – керасинова лампа. Алгоритм визначення складається з наступних кроків:

– Визначення n – вимірної сітки, де кожній точці сітки відповідає випадковий градієнтний вектор

– Знаходження положення відносно найближчих вузлів сітки для кожної точки

– Розрахунок градієнтних векторів для найближчих вузлів сітки

– Застосування лінійної інтерполяції та функцію згладжування

Цей алгоритм було розроблено у 1983 році Кеном Перліном, та є примітивом процедурних текстур, які належать до градієнтних шумів [17]. Він має застосування в комп'ютерній графіці, та робить її більш реалістичною.

Лінійна інтерполяція. Цей метод використовується для плавного переходу між мінімальною та максимальною інтенсивністю світла на ігровій сцені, що основана на значенні, яке виходить з значення шуму Перліна.

Формула розрахунку значення:

$$y = y_1 + \frac{(x - x_1) \cdot (y_2 - y_1)}{x_2 - x_1} \quad (2.1)$$

Де y – апроксимоване значення,

x – незалежна змінна, для якої виконується апроксимація,

x_1, y_1 – координати першої відомої точки,

x_2, y_2 – координати другої відомої точки.

Кватерніони (Quaternion) [18]. В проекті даний метод має використання для представлення обертання камери у тривимірному просторі. Функція викликається кожного разу при оновлені кадрів, та обчислюється обертання камери на основі вхідних значень чутливості миші. Камера прикріплена до головного герою, та слідує за його об’єктом.

Кватерніони використовуються для розрахунку поворотів у просторі, зокрема у тривимірній графіці. Кватерніони описані ірландським математиком Вільямом Роуен Гамільтоном у 1843 році, та широко застосовуються у наукових дослідженнях та інженерії [19].

Вище описані алгоритми та методи виконується в проекті у вигляді інструкцій з аргументами функції, що є базовим функціоналом рушія в колекції загальних математичних функцій.

2.3. Опис використаних технологій та мов програмування

2.3.1. Опис мови C# у Unity

В поточній кваліфікаційній роботі використовується рушій Unity. Для написання сценаріїв та взаємодії з об’єктами, окрім застосування Unity Editor [20], використовуються сценарії на мові програмування C#.

C# є об’єктно-орієнтованою мовою програмування з сучасним синтаксисом, який включає в себе різноманітні можливості, що, зокрема використовуються під час створення ігрових додатків. Дані можливості включають:

– Багатопоточність, яка надає можливості створювати додатки, які використовують паралельні потоки виконання з ціллю оптимізації застосунків;

– Інтеграція з .NET Framework розширює можливості, надаючи доступ до спектру бібліотек і функцій, що спрощують розробку.

– Об’єктно – орієнтовану парадигма програмування, що відповідає структурі об’єктів у Unity.

Unity має вбудований інструмент – Unity Editor, що дозволяє використовувати C# для програмування в цьому середовищі за допомогою скриптів та компонентів.

Сценарії підключаються до внутрішньої структури програми, створюючи клас, який успадковує вбудований клас під назвою MonoBehaviour [21]. Цей є шаблоном для створення нового типу компонента, який можна прикріпити до ігрових об’єктів. Кожен раз, коли додається сценарій до ігрового об’єкту, створюється новий екземпляр цього об'єкта, описаного у схемі класу.

Рушій використовує відкритий вихідний код платформи .NET для забезпечення сумісності програм, створених у Unity, з різноманітними апаратними конфігураціями.

Кожен раз при компіляції проекту проводиться пошук та видалення невикористаного коду в додатку. Таким чином, відбувається оптимізація розміру збірки, і допомагає зменшити розмір вихідного проекту.

Рушій підтримує різні платформи та серверні сценарії, але деякі системні бібліотеки .NET можуть вимагати специфічних реалізацій для правильної роботи на цій платформі. Немає гарантій продуктивності чи розподілу деяких системних бібліотек .NET у версіях Unity, і Unity не виправляє регресії продуктивності в цих бібліотеках [22].

Мова програмування C# має велику спільноту розробників. Відкрита платформа .NET, яка забезпечує сумісність програм з різними апаратними конфігураціями, а також має значні функціональні можливості, що дозволяє створювати технологічні рішення.

2.3.1.Опис Unity в 3D

Unity 3D пропонує багатий інструментарій для створення ігрових додатків. Графічні можливості рушія в цьому напрямку включають:

– відображення рельєфу;
– екранна просторова-залежна оклюзія (SSAO) [23];
– динамічні тіні з використанням мап тіней;
– ефекти рендеринга [24] до текстури;
– повноекранна постобробка;

Рушій підтримує різноманітні формати 3D моделювання, а саме вихідні файли з найбільш відомих програмних засобів для створення моделей, таких як Blender [25], ZBrush [26] або 3ds Max [27], та інші популярні програмні продукти.

В Unity Editor доступний Asset Store, який є багатим ресурсом для розробників. Даний магазин пропонує понад 7000 тис. різноманітних пакетів[28], серед яких є 3D – моделі, текстури, матеріали, системи частинок, музика та звукові ефекти.

Поточний проект використовує набір запропонованих платформою інструментів, в тому числі доступні ефекти та пакети.

2.4. Опис структури системи та алгоритмів її функціонування

2.4.1. Структура та алгоритми системи

З ціллю структурування інформації про компоненти, було обрано та використано метод декомпозиції структури. Цей процес дозволяє розглядати будь – яку систему як сукупність підсистем, які також можуть бути розділені на частини. В даному контексті, системами будуть як об’єкти сцени, ігрові механіки, а також контролери керування ігровими процесами (табл. 2.1).

Таблиця 2.1

Декомпозиція структури проекту

Ігровий додаток	Персонаж	Контролер кроків
		Головна камера
		Контролер персонажу
		Контролер ліхтарика

Продовж. табл. 2.1

	Механіки	Система термяви		
		Система інвентарю		
		Підбір предметів		
		Система дверей		
		Ігрова кімната №0	Система телепортів	
		Ігрова кімната №1	Головоломка «Бойлер»	Механіка вентилю
			Головоломка «Електричний щиток»	Система збору компонентів
		Ігрова кімната №2	Головоломка «Сейф»	Система цифрового блоку
			Головоломка «Картина»	Система збору елементів
			Головоломка «Годинник»	Система стрілок
	Менеджер камер	Перемикач камер	Камери сцени	
	Інтерфейс	Система вказівника	Контролер вказівника	Вказівник взаємдії з об'єктами
				Вказівник взаємодії з головоломками
	Звукове супроводження	Ембієнт	Звук дощу	
		Підбір предметів	Звук підбору	
		Взаємодія з дверима	Звук відкриття	
		Активація квестів	Перемикання вентилю	Звук перемикання
			Починку елементу «Електричний щиток»	Звук починки

Продовж. табл. 2.1

<table>
<tr><td rowspan="8"></td><td rowspan="2"></td><td rowspan="2"></td><td colspan="2">Звук кроків персонажу</td></tr>
<tr><td colspan="2">Звук механіки «темрява»</td></tr>
<tr><td rowspan="3">Анімації</td><td colspan="3">Відчинення дверей</td></tr>
<tr><td colspan="3">Підбір предметів</td></tr>
<tr><td>Активація головоломок</td><td colspan="2">Контролер стану положення об'єкту</td></tr>
<tr><td rowspan="3">Джерела Світла</td><td rowspan="3">Ліхтарики</td><td rowspan="2">Статичні джерела</td><td>Керасинові лампи</td></tr>
<tr><td>Елементи інтер'єру</td></tr>
<tr><td>Динамічні джерела</td><td>Переносний ліхтарик персонажу</td></tr>
<tr><td>Менеджер колізій</td><td>Контролер колізії об'єкту</td><td colspan="2">Об'єкти та параметри колізій</td></tr>
</table>

Додаток розбито на компоненти та системи, що виконують завдання. Нижче наведено опис компонентів:

– Менеджер колізій. Відповідає за стеження зіткнення об’єкту моделі персонажу з іншими об’єктами, або тригерами. При цьому, встановлюючи відповідні прапорці. Даний функціонал використовується іншими системами для перевірки можливості застосування функцій або підсистем.

– Контролер об’єкта гравця. Цей компонент визначає розташування та логіку переміщення моделі головного героя, а також взаємодіє з іншими елементами, що прив’язані до персонажу, а саме: ліхтарик, звукові об’єкти, і головна камера. Гравець має певну швидкість, з якою може пересуватися по локації, та інші обмеження. Також, цей елемент відповідає за вид головної камери і змінення параметрів її розташування в залежності від положення та повороту моделі гравця.

– Менеджер камер. Містить елементи камер та відповідний менеджер колізій, з яким кооперується. Зона відповідальності стосується головоломок, та перемикання камер при взаємодії з тригерами. Кожна камера містить унікальний вказівник миші для управління та параметри освітлення (рис. 2.1)

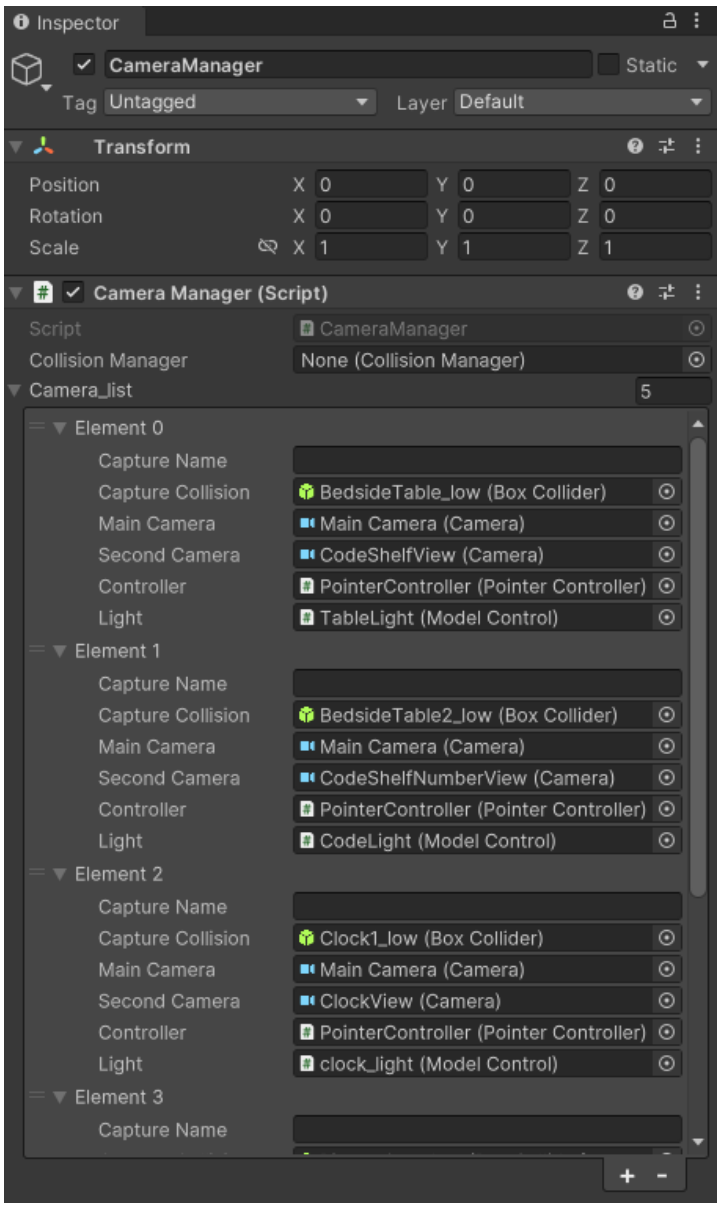

Рис. 2.1. Менеджер камер застосунку

– Контролери анімацій. Кожен об’єкт сцени, що анімується, має контролер, який відповідає за змінення положення об’єкта на сцені під час активації відповідних тригерів. Наприклад, тригерами можуть бути:

– наведення миші на об’єкт;

– натискання кнопки миші або клавіатури;

– перетин певної ігрової зони.

Дані тригери можуть поєднуватися разом з іншими. Таким чином, пропозиція гри до взаємодії може відбутися при одночасному знаходженні моделі

героя в певній області, при наявності декількох прапорців в системі, та одночасного наведення і натискання кнопка миші або клавіатури.

2.4.2. Опис ігрових механік і послідовності завдань

Додаток має механіки, що є основною та невід'ємною частиною ігрового процесу. В цілому, ігрові механіки – це набір правил і систем, що визначають ігровий процес та взаємодію гравця з застосунком. Вони включають в себе всі дії, які може виконувати головний герой, а також способи, якими гра реагує на ці дії. Механіки формують основні аспекти гри, одним з котрих й найважливішим є взаємодія з віртуальним середовищем. Основні компоненти механік включають:

- правила гри, які визначають, що гравець може робити, або що не може;
- цілі та задачі, які повинні виконуватися під час прогресу;
- система взаємодії з об'єктами сцени та середовищем;
- елементи випадковості, які впливають на результат дій.

Таким чином, є необхідним створення механік, які дозволять взаємодіяти з ігровим процесом. В поточному проекті, існує наступний перелік:

- механіка «темрява». Дане правило обмежує гравця в часі, який може бути витрачений на проходження головоломок. Воно діє тільки в тому випадку, якщо користувач знаходиться в темряві поза статичними джерелами світла. Блок – схема алгоритму дії зазначена на наступній діаграмі (рис 2.2):

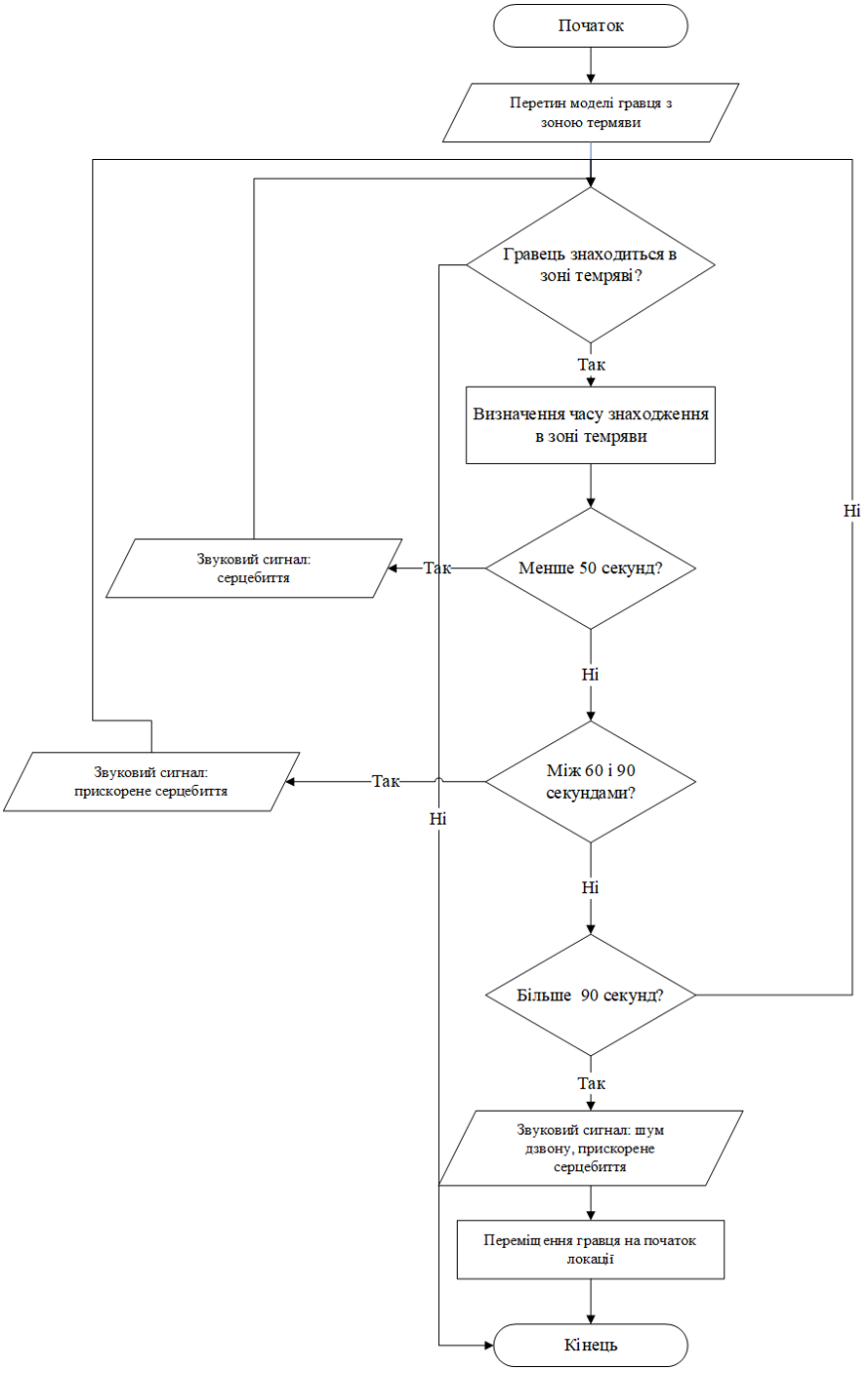

Рис. 2.2. Алгоритм механіки «Темрява»

Таким чином, під час знаходження поза статичними джерелами світла, гравець отримує звукові ефекти. У разі їх ігнорування, користувач переміщується на початок локації. З ціллю запобігання даних ефектів, гравець повинен знаходити та активувати нові джерела світла, які діють як безпечні зони та контрольні точки.

– Система інвентарю. Ця система зберігання, що має інформацію про предмети в поточному інвентарі гравця. Вона не має графічного відображення, а об’єкти показуються один раз після їх підбору, у разі необхідності.

– Кодовий замок. Являє собою відгадування коду на одному з квестів ігрової локації. Послідовність цього завдання зображена на рис. 2.3. Механіка розв’язування полягаю в наступному:

– В момент взаємодії з головоломкою вид камери перемикається на камеру, що прив’язана до цієї головоломки;

– Під час натискання на кнопки цифрового блоку з’являються звукові сигнали. Таким чином, необхідно ввести вірну комбінацію;

– При некоректному введені значення, з’являється відповідний звуковий сигнал.

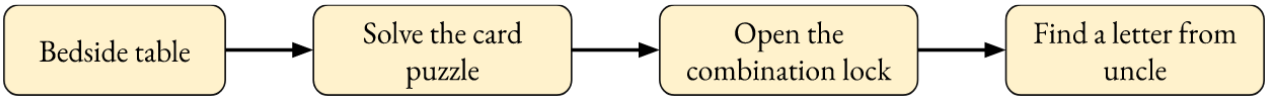

Рис. 2.3. Послідовність завдання з кодовим замком.

Розглянемо послідовність дій усіх квестів:

– Головоломка «Картина» (рис. 2.4):

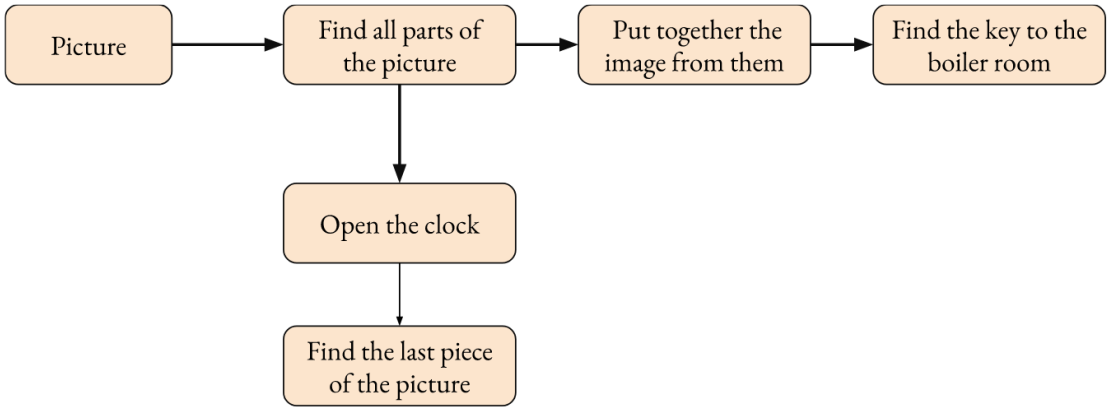

Рис. 2.4. Послідовність дій з картиною

– Головоломка «Електричний щиток» (рис. 2.5):

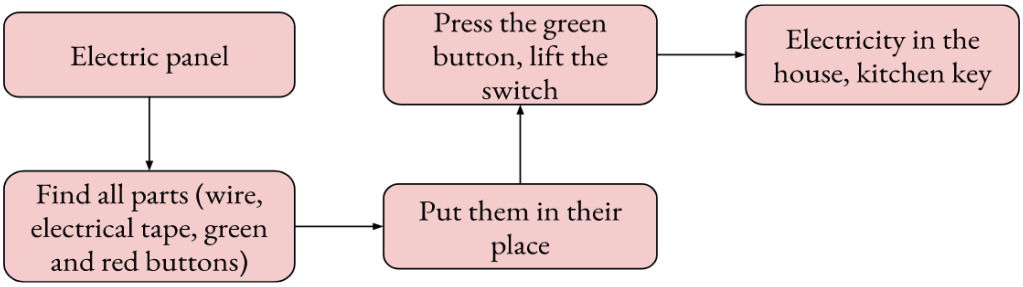

Рис. 2.5. Послідовність дій з «Електричний щиток»

– Квест «Бойлер» (рис. 2.6):

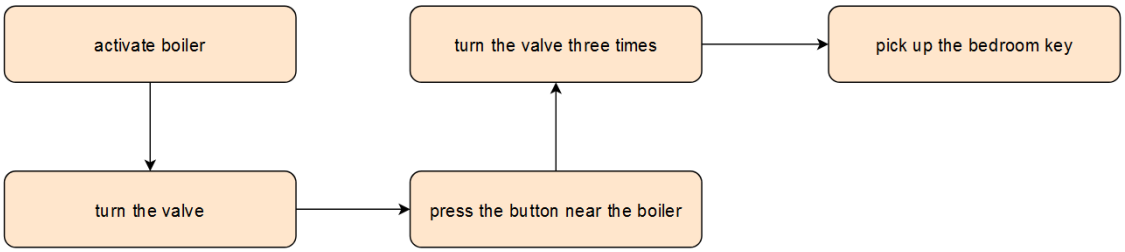

Рис. 2.6. Послідовність дій з «Бойлер»

– Квест «Годинник» (рис 2.7):

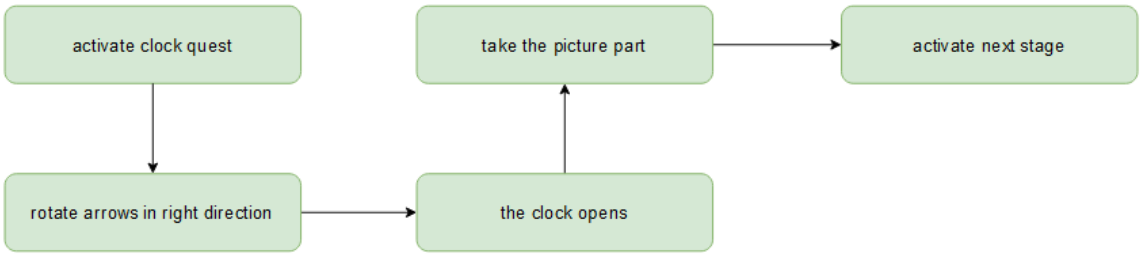

Рис. 2.7. Послідовність дій з «Годинник»

2.4.3. Структура ігрових компонентів

На рис. 2.8 – 2.15 представлено UML – діаграму класів компонентів структури програмного застосунку. Даний підхід дозволяє візуалізувати структуру програмної системи, класи, їхні атрибути та взаємозв’язки між ними.

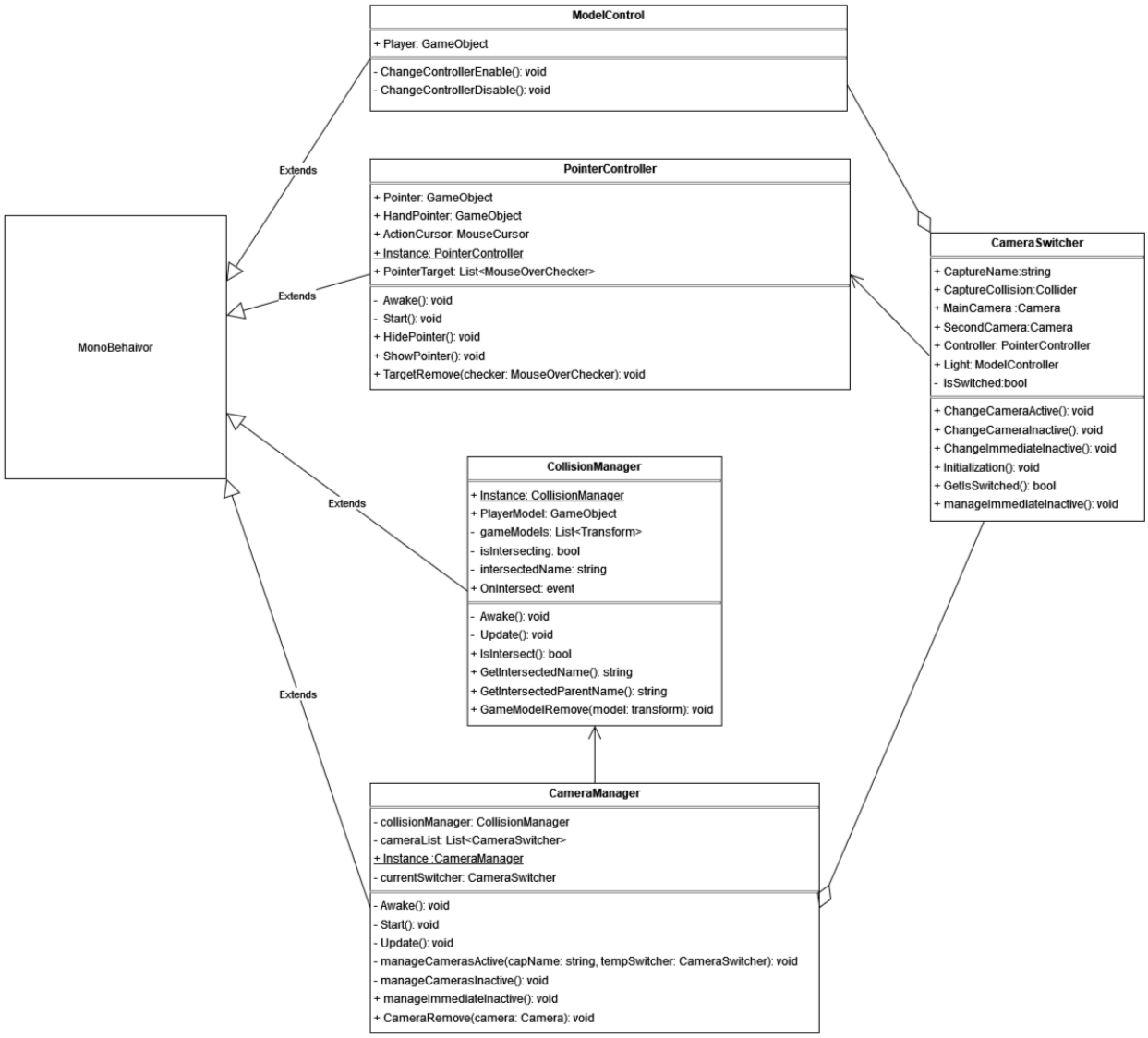

Рис 2.8. Діаграма класів глобальних скриптів

Рис. 2.8 представляє діаграму класів глобальних скриптів. Класи мають наступні взаємозв’язки:

Ці класи описують принцип взаємодії з функцією перемикання камер в грі, та зв’язки з іншими класами додатку, такими як менеджер колізій, що відповідає

за перетин ігрових територій та ініціювання тригерів. Окрім цього, описані класи контролера вказівника і моделі персонажа взаємопов’язані під час виконання алгоритмів застосунку.

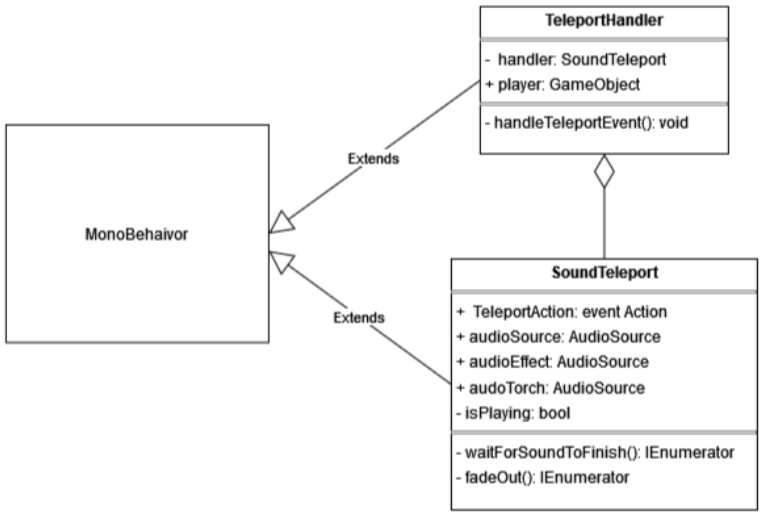

Рис 2.9. Система телепортів

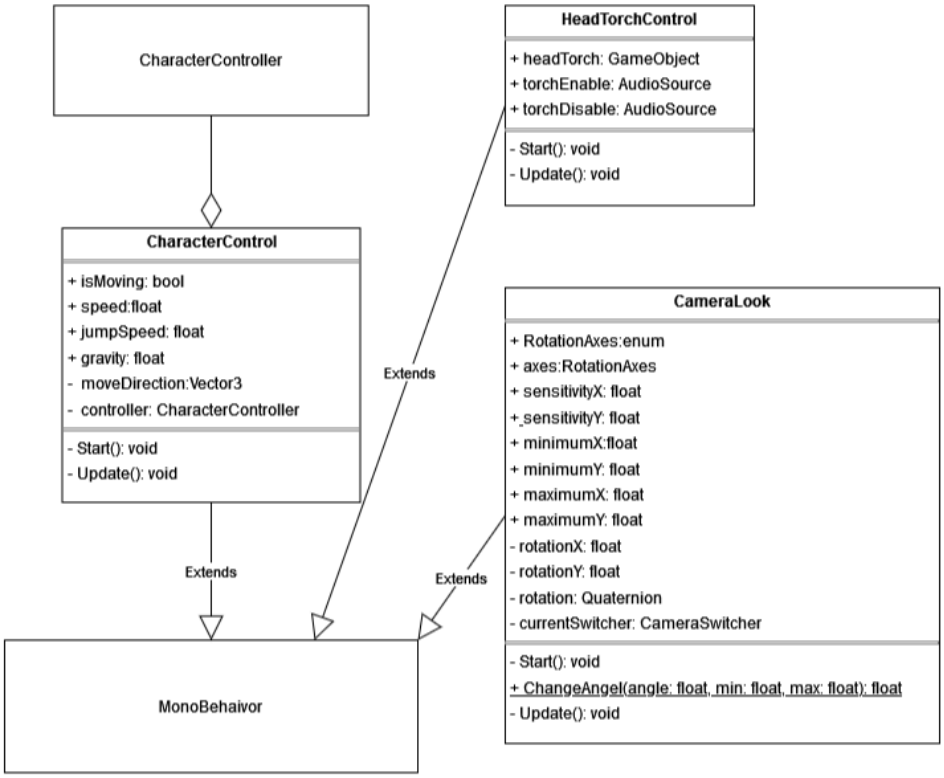

Рис 2.10. Контроль персонажу та його елементів

Класи, що представлені на діаграмі рис. 2.10, керують, переважно, рухом і обертанням персонажу. Методи виконують пересування в просторі, та оберт і контроль камери. Окрім цього, до об'єкту моделі гравця прикріплені додаткові елементи, такі як ліхтарик.

На рис. 2.11 зображено основні моделі, відповідальні за квестову систему ігрової кімнати. Інтерфейс IClickable дозволяє досягти поліфоморфізму, спростити та зменшити кількість класів.

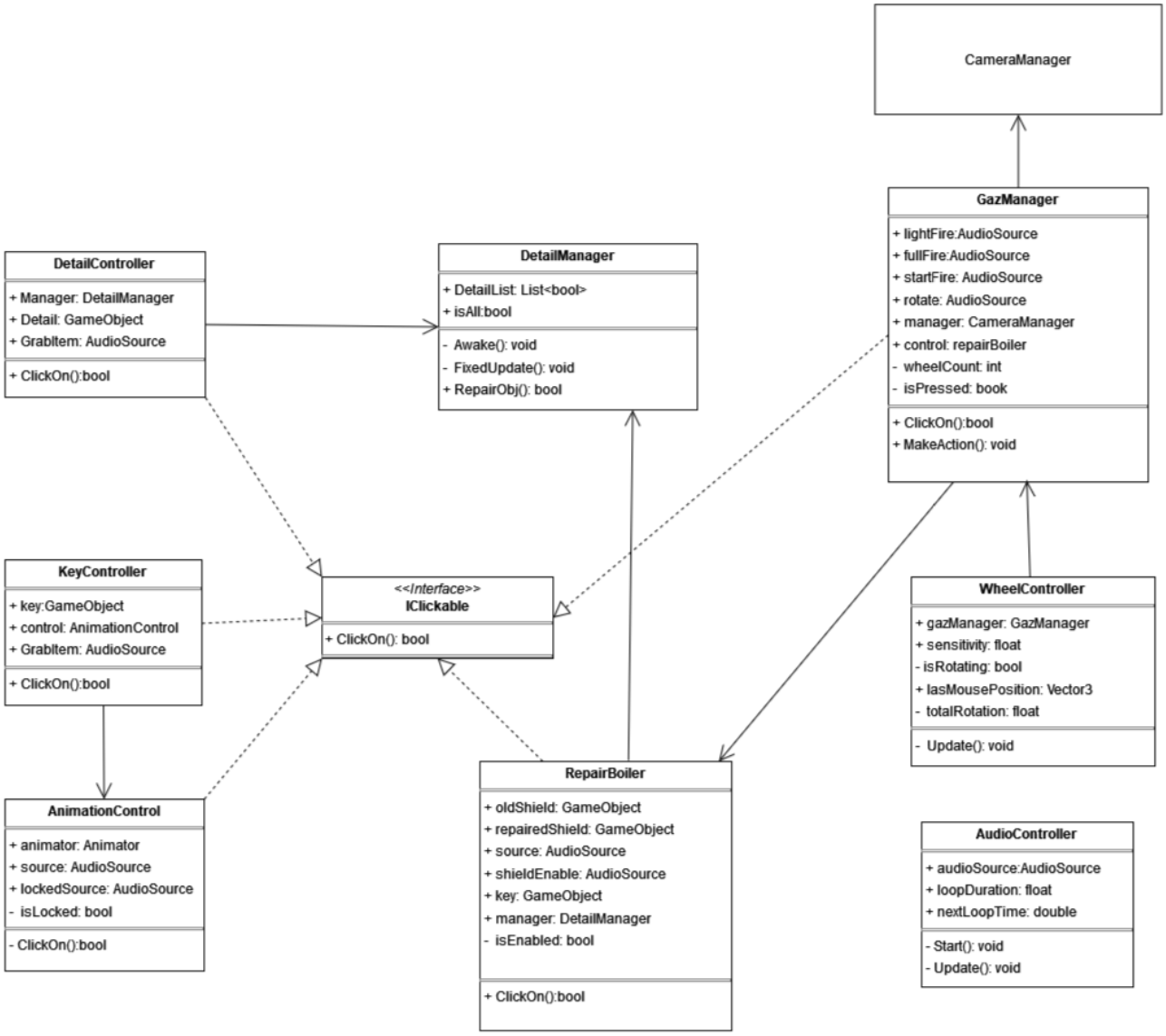

Рис 2.11. Система механік квесту «Бойлер»

Контролери керують світлом і станом персонажу (рис. 2.12), його позицією. Цей функціонал має зону відповідальності за механіку «темрява».

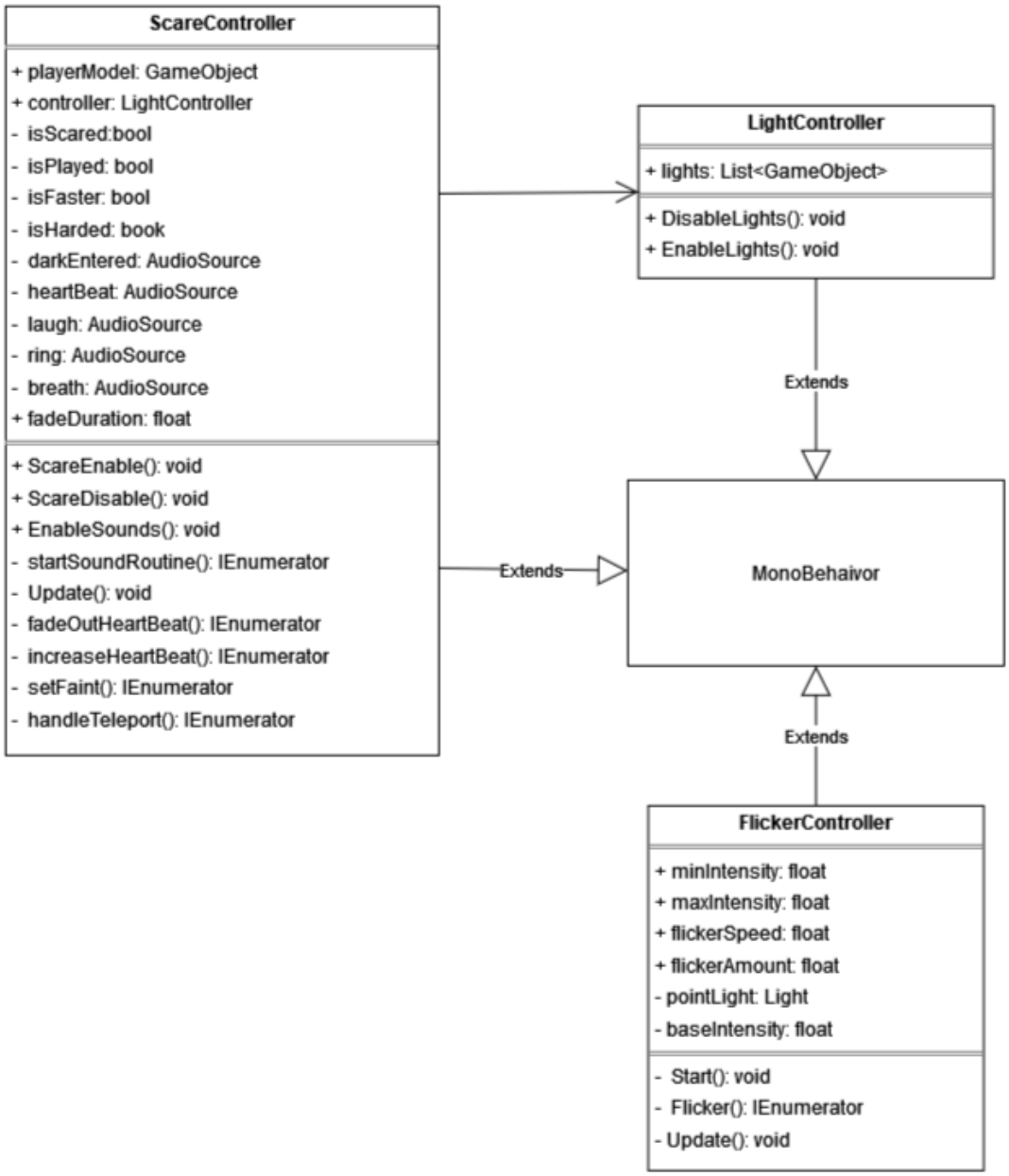

Рис 2.12. Модель механіки «темрява»

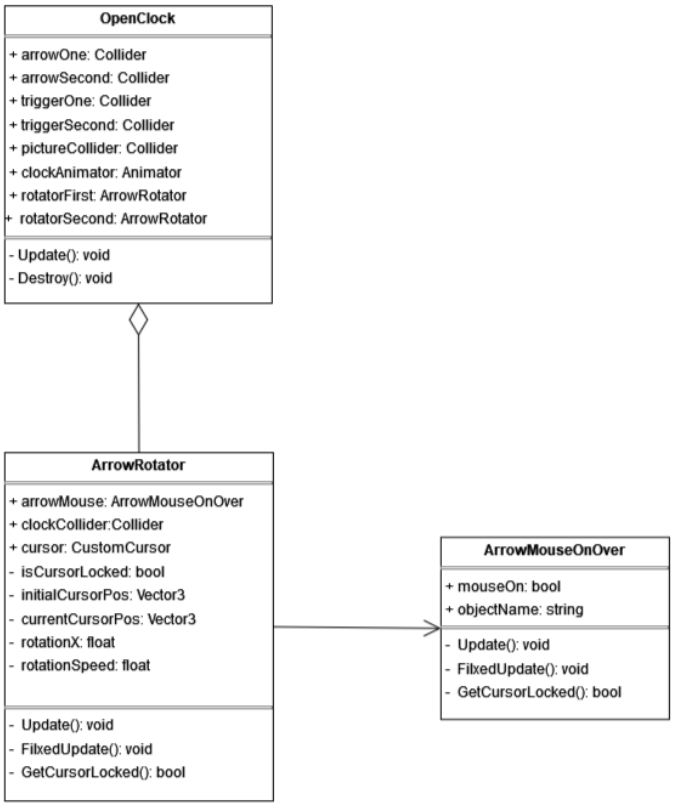

Рис 2.13. Діаграма класів системи «годинник»

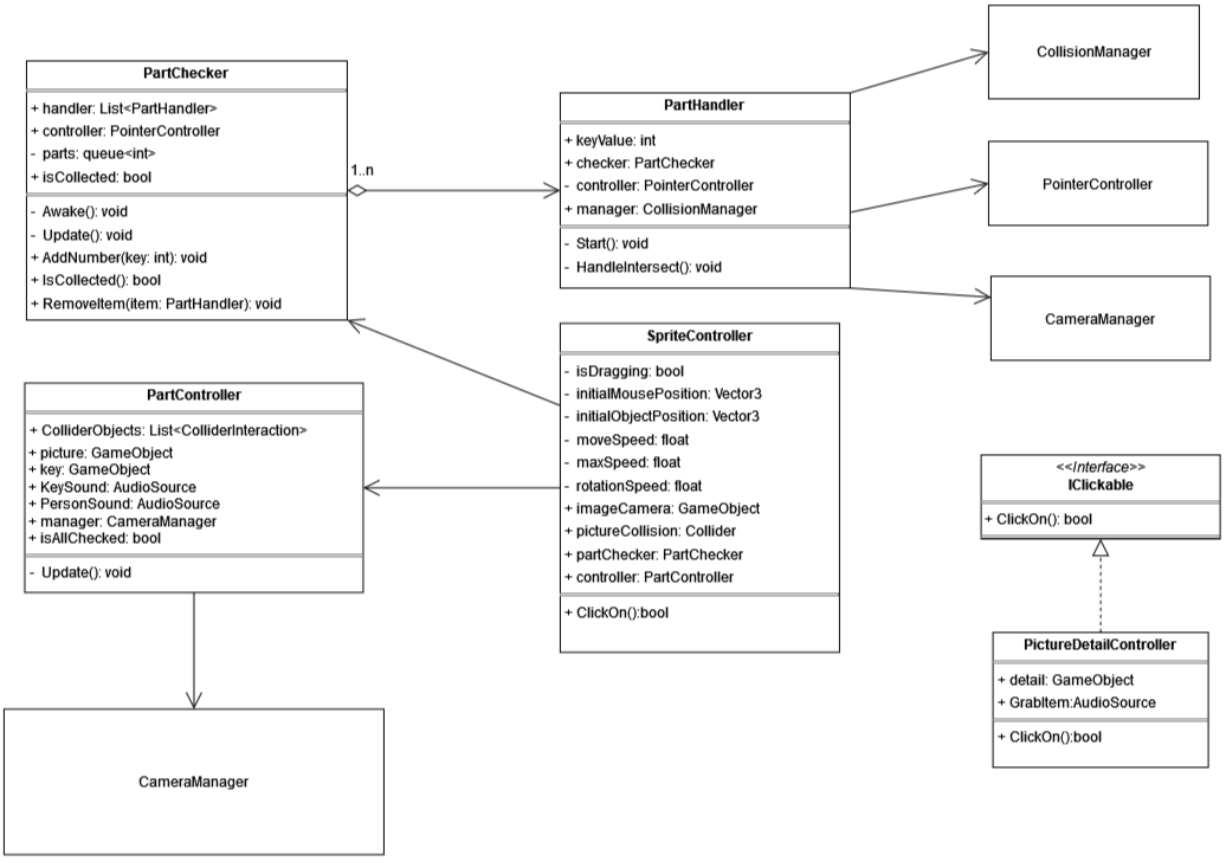

Рис. 2.14. Діаграма квест – системи «картина»

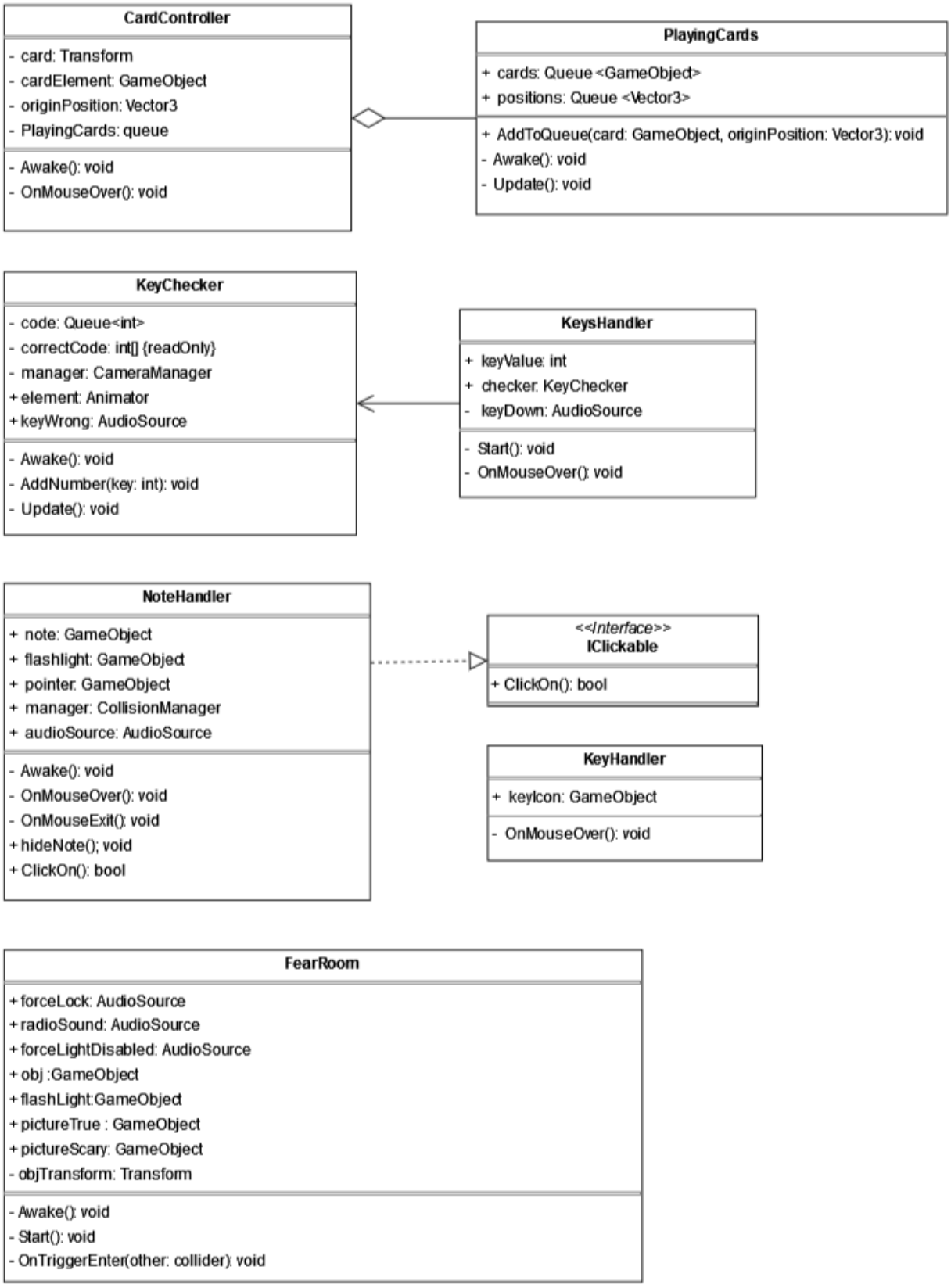

Рис. 2.15. Діаграма класів об’єктів ігрової кімнати

Детальне представлення призначення та взаємозв’язків між класами надано в табл. 2.1.

Таблиця 2.1.

Таблиця призначення та зв’язків класів програми

Назва	Призначення	Зв’язки
CharacterControl	Керує пересуванням персонажу, використовуючи компоненти CharacterController	Агрегація з «CharacterController»
HeadTorchControl	Керує ліхтариком та його супровідними звуковими ефектами	
CameraLook	Обертання камери за допомогою миші для огляду сцени, з будь – якої точки зору	
ModelControl	Перемикання активності між гравцем і і поточною моделлю	Агрегація з «CameraSwitcher»
CameraSwitcher	Перемикання активності між двома камерами на основі вхідних данніх користувача, а також управління курсором	Асоциація з «PointerController» та «CameraManager»
PointerController	Відповідає за керування покажчиком, забезпечує показ або приховування покажчика та керування списком цілей	
CameraManager	Забезпечую контроль камерами у грі, забезпечуючи їх активацію та деактивацію на основі умов взаємодії з об’єктами та стану колізій.	Асоциація з «CollisionManager»

Продовж. табл. 2.1

CollisionManager	Керує колізіями між ігровими моделями та моделлю гравця, визначає факт перетину об’єктів, зберігає ім’я поточного перетнутного об’єкта і викликає події під час перетину.	
SoundTeleport	Відтворення звукових ефектів при перетині певних зон, та управління перемиканням стану об’єктів. Ініціює подію орбоки аудіоефекту телепортації	Агрегація з «TeleportHandler»
TeleportHandler	Обробка події телепортації, виклик звукових ефектів телепортування; вимкнення можливості керування персонажем; зміна позиції гравця на зазначені координати; відновлення можливості керування персонажем	
ScareController	Реалізує механіку «темрява»; змінює звукові та візуальні ефекти впродовж 90 секунд; вмикає та вимикає стан реагування на темряву	Асоциація з «LightController»
LightController	Вимкнення та увімкнення світлових джерел	
FlickerController	Ефект мерехтіння світла. Налаштовує параметри мінімальної та максимальної інтенсивності світла, швидкості мерехтіння та її амплітуду; змінює інтенсивність світла в залежності від шуму	

Продовж. табл. 2.1

DetailController	Управління деталями ігрової кімнати, реагує на кліки користувача; додає деталі до списку деталй; відтворює звук захоплення предмета; прихвоує курсор відносно ігрової сцени	Асоциація з «DetailManager» та реалізація «IClickable» інтерфейсу
DetailManager	Контроль та перевірка стану деталей; вказує, чи всі деталі зібрані	
RepairBoiler	Відповідає за відновлення котла; перевіряє, чи всі деталі відновлені, і чи активований «Електрощиток»	Реалізує інтерфейс «IClickable»
AnimationControl	Знімає блокування з елементу анімацій; приховує вказівник; приховує ігровий об’єкт	Реалізує інтерфейс «IClickable»
KeyController	Керує використанням ключа у грі, обробляє натискання на об’єкт	Асоциація з «AnimationControl», реалізує інтерфейс «IClickable»
GazManager	Здійснює керування увімкнення джерела вогню, зміни його рівня; викликає звукові ефекти зміни положення вентилю	Асоциація з «RepairBoiler», реалізує інтерфейс «IClickable»
WheelController	Обертання об’єкта вентилю, враховуючи чутливість руху миші	Асоциація з «GazManager»
AudioController	Автоматичне відтворення звукових доріжок в регулярних інтервалах	
OpenClock	Відкриття годинника у грі. Визначає перетин двох стрілок з відповідними тригерами та активує анімацію відкриття годинника; вимикає колайдер годинника	

Продовж. табл. 2.1

ArrowRotator	Обертання стрілок годинника; визначає, чи заблокований курсор над стрілкою, та обертає її відповідно до руху курсора	Агрегація з «OpenClock», асоциація з «ArrowMouseOnOver»
ArrowMouseOnOver	Взаємодія з мишкою для стрілки годинника. Викликається, коли миша наведена на об’єкт	
PlayingCards	Управління колодою грального карт; керування чергою карт та додавання разом з початковими позиціями; перевірка відповідності двох карт одночасно	Агрегація з «CardController»
CardController	Обробка натискань на картки; підйом картки та додавання в чергу при натсканні	
KeysHandler	Відтворення звуку натискання клавіш механіки «сейф»; додавання значення натиснутої кнопки	Асоциація з KeyChecker
KeysChecker	Зберігання введеної послідовності кнопок; перевірка коректності послідовності введених значень; відтворення анимації відкриття ігрового об’єкта «сейф»	
NoteHandler	Обробка натискання на ігрові нотатки; перевірка натискання клавіші миші та імени об’єкта; відображення та приховування нотатків	Реалізує інтерфейс «IClickable»
FearRoomController	Відтворення звукових та візуальних ефектів; запуск аниміцій та зміна положення об’єктів	

Продовж. табл. 2.1

PictureDetailController	Підбір за знищення ігрового елементу; відтворення звукового супроводження	Реалізує інтерфейс «IClickable»
PartController	Перевірка частини мають перетин; виклик обробки подій, якщо всі частини зібрани	Асоциація з «CameraManager»
SpriteController	Управління спрайтами, перевірка та активування; перетягування спрайту та обертання	Асоциація з «PartChecker»
PartChecker	Додавання частин до черги та перевірка на зібраність; видалення частин	Асоциація з «PartHandler»
PartHandler	Обробка взаємодії з частинками об’єктів; виявлення взаємодії	Агрегація з «PartChecker». Асоциація з «ColissionManager», «PointerController» та «CameraManager»

2.4.4. Файлова структура

Застосунок складається ігрового додатку складається з декілької частин:

– Assets. Основна папка для зберігання всього вмісту проекту, включаючи скрипти, текстури, моделі, звуки, префабми, та інші ресурси.

– Library. Містить інформацію про зібрані файли та імпортовані ресуси.

– ProjectSettings. Зберігає файли конфігурації проекту: налаштування редактора, якості, вхідних даних.

Цей перелік директорій містить основі елементи застосунку, але, також, ще наявні такі папки як Logs, Obj, Temp та Packages (рис 2.16).

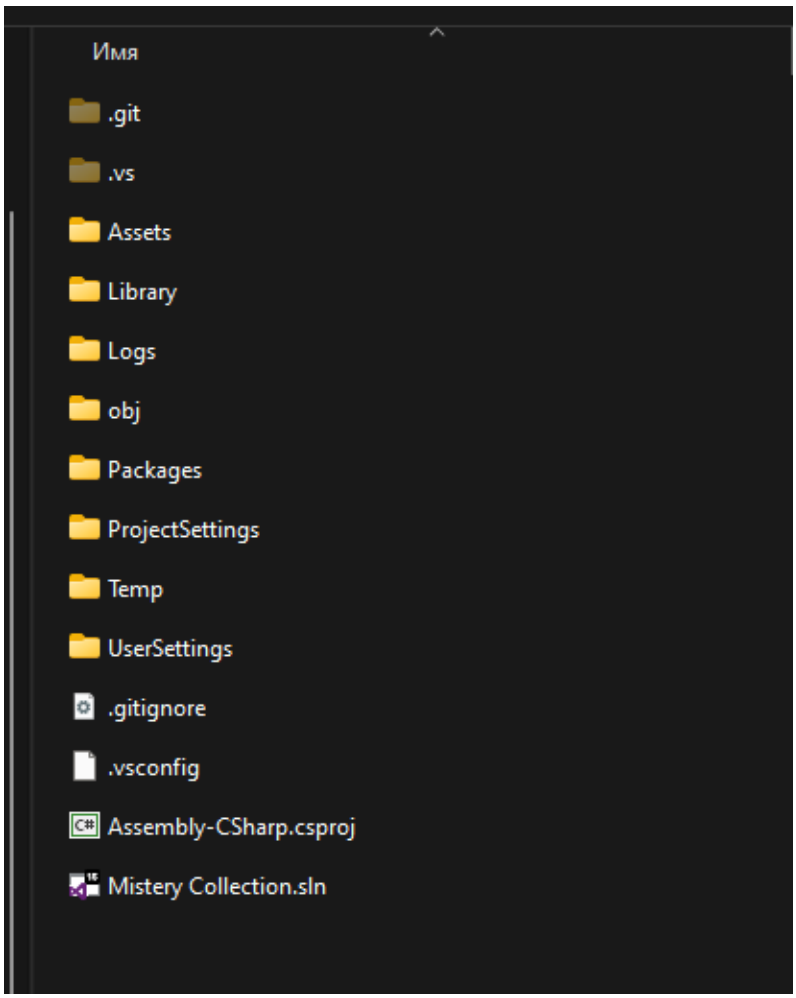

Рис 2.16. Загальний вигляд папки проекту

На рис. 2.17 зображено загальну структуру директорій Assets. В поточному проекті, кожен елемент додатку розибитй на відповідні частини та підчастини. Наприклад, «GloabalScripts» зберігає усі глобальні скрипти застосунку, зокрема менеджер камер та колізій, керування кукрсором та освітленням. Тоді як «Sounds» містить усі звукові дорожки, що застосовані в додатку.

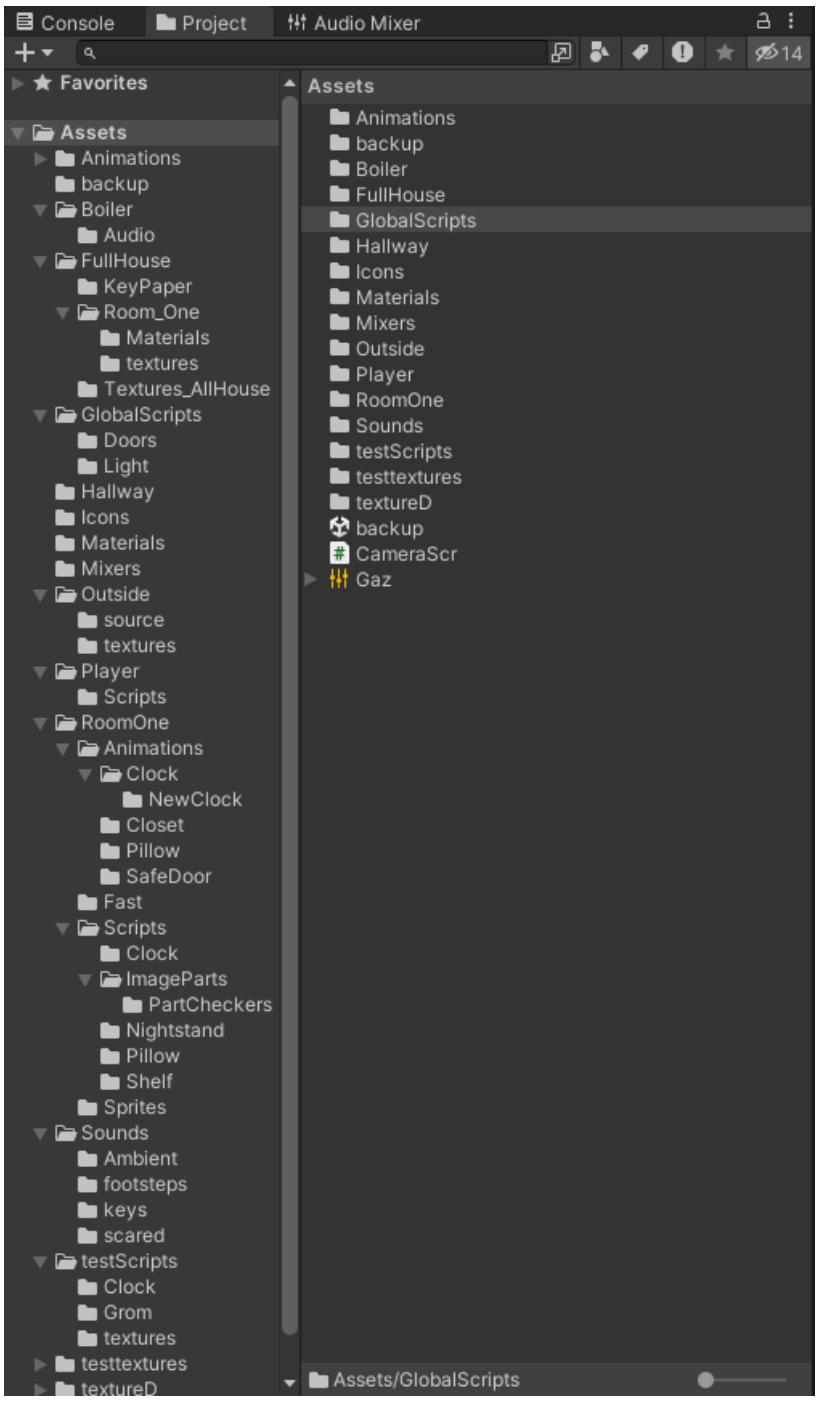

Рис. 2.17. Файлова структура «Assets»

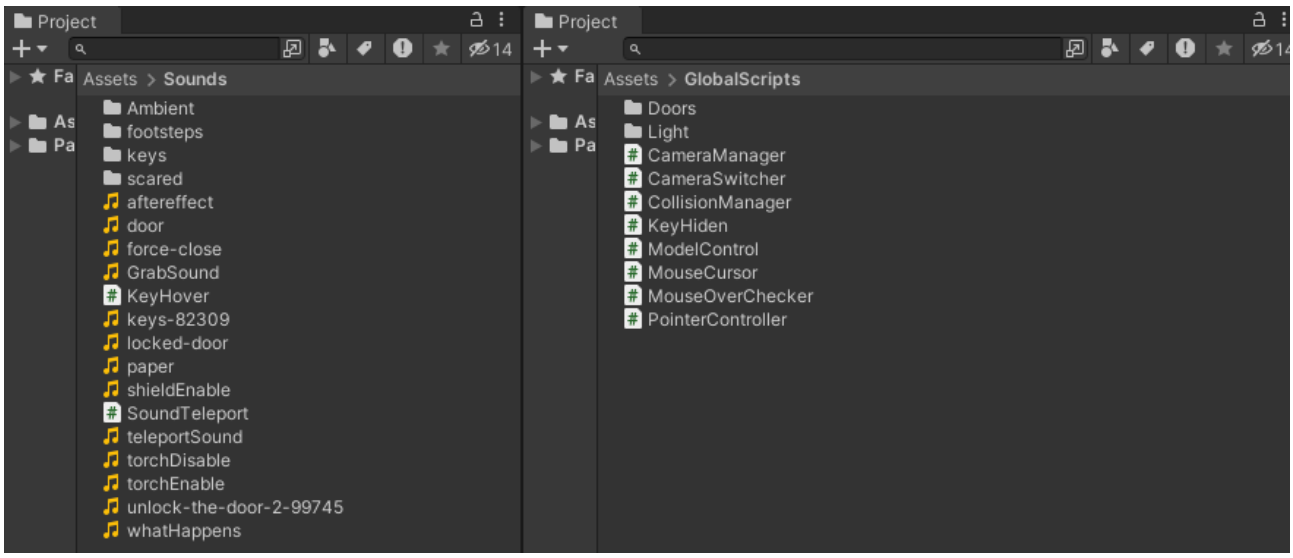

Рис. 2.17. Звукові ефекти та глобальні скрипти

2.4.5. Ієрархія ігрових об’єктів

Ігрові об’єкти сцени розташовані ієрархічно, і мають як дочірні об’єкти, так і об’єкти предків (рис 2.18). Кожен об’єкт має параметри, що зв’язані зі скриптом, або матеріалом, чи переліком властивостей об’єкта доступних в Unity. Основна частина, зокрема глобальні скрипти та скрипти персонажа відокремлені і знаходяться в самому початку ієрархії для швидкого доступу та редагування. Об’єкти сцени, що представляють декорації віртуального середовища, розміщені в окремому компоненті (рис 2.19).

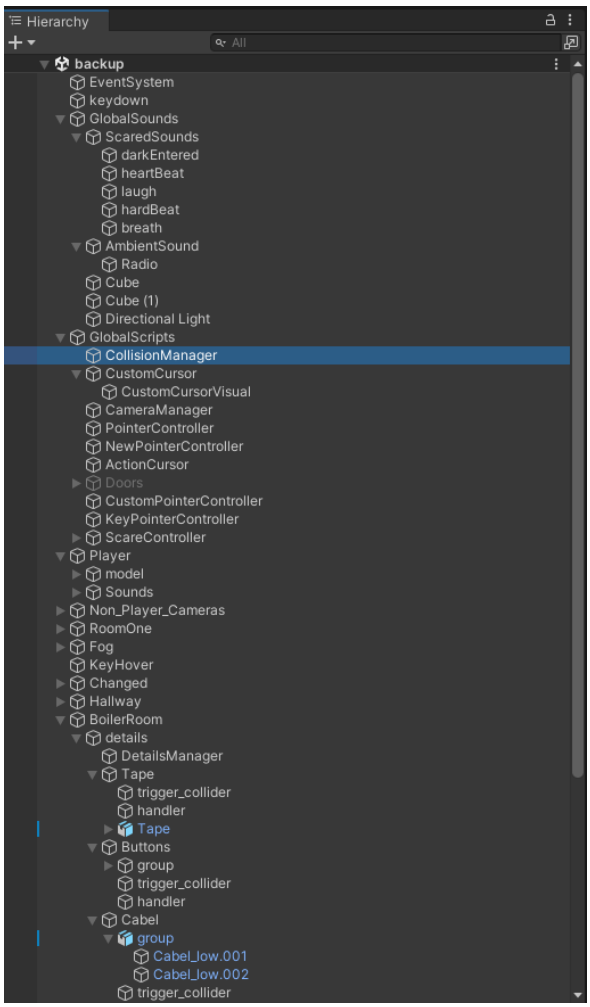

Рис. 2.18. Скорочений список ігрових об’єктів

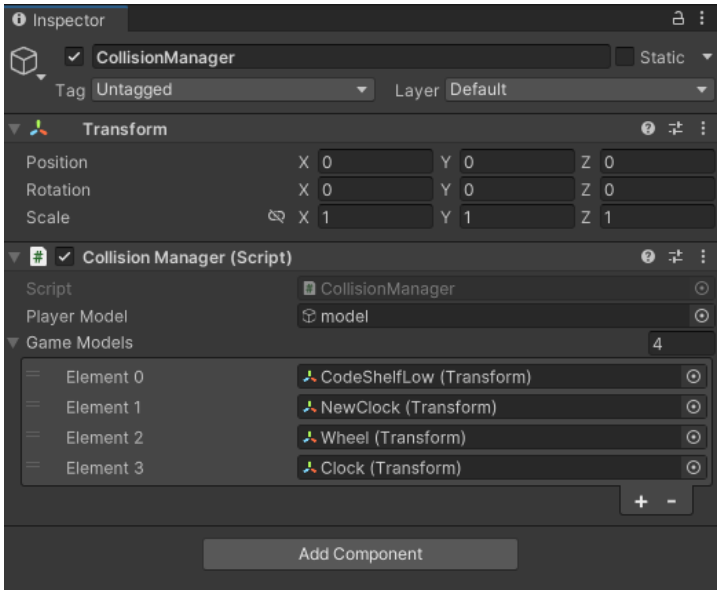

Рис. 2.19. Параметри менеджеру коллізій

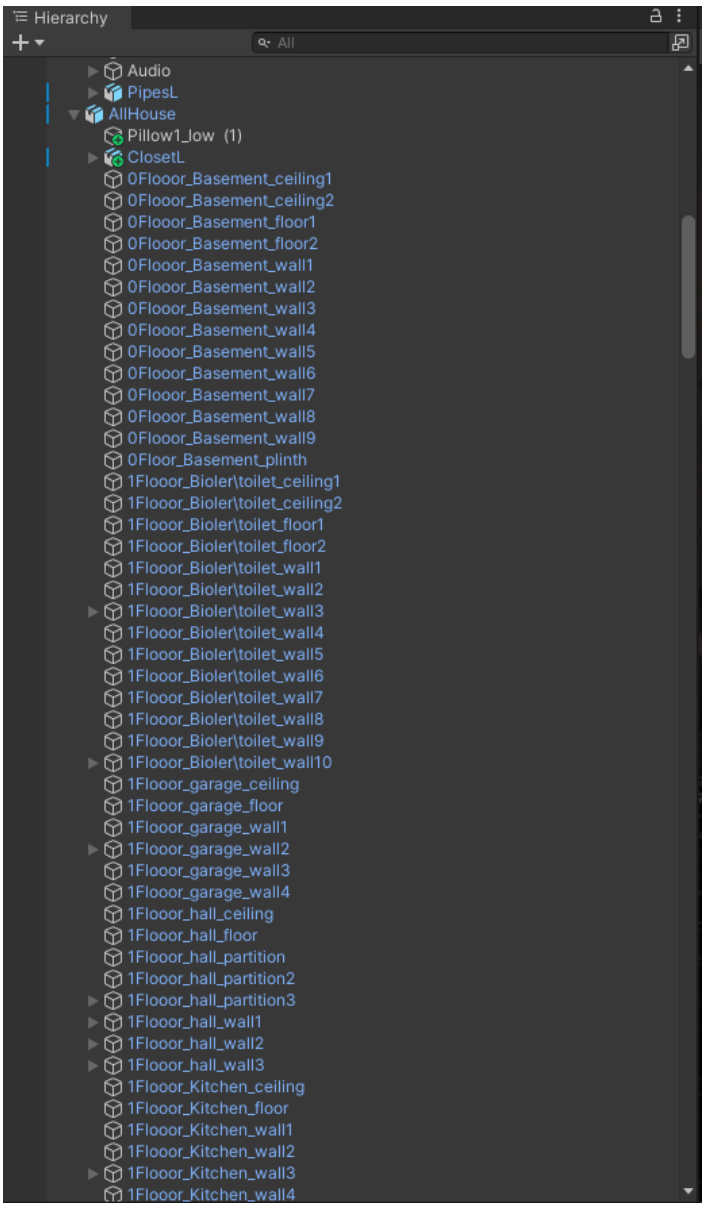

Рис. 2.20. Скорочене представлення декоративних об’єктів сцени

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Вхідні дані забезпечуються через два основних джерела: пристрої введення (мишка та клавіатура) та інтерактивні елементи в ігровому середовищі. Таким чином, перефірійні пристрої мають наступні та цілі збору даних:

– Мишка. Використовується для огляду навколишнього середовища, взаємодії з об’єктами. Рух змінює напрямок камери (вид з першої особи), а кнопки використовуються, а кнопки миши – використовуються для взаємодії з об’єктами.

– Клавіатура. Здійснює пересування персонажа при натисканні кнопок `W`, `A`, `S`, `D`, відповідно вперед, вліво, назад і вправо.

Організація вхідних даних здійснюється за допомогою інструментів вводу Unity, безспосередньо, в відповідальних за цю зону класах програми.

Вихідні дані представлені у виляді візуальної та аудіовізуальної інформації:

– звукові ефекти для сигналізації про важливі події (звуки кроків, натискання на ігрові елементи, або підбір предметів);

– текстові повідомлення на екрані, що представляються в вигляді ігрових записок з текстом, або текстовими зображеннями на текстурах;

– візуальні ефекти, такі як VFX [29] та системи частинок при активації тригерів, а також анімації ігрових об’єктів.

Звукові файли заздалегідь обробляються в звукових редакторах, і зберігаються в «Assets/Audio», звідки активуються за допомогою скриптів та компонентів Unity – AudioSource.

Візуальні повідомлення і UI-елементи зберігаються і налаштовуються через систему Canvas [30] в Unity

Візуальні дані, такі як шейдери та матеріали, анімації і ефекти створюються в окремих файлах, та використовуються під час виконання додатку.

2.6. Опис розробленої системи

2.6.1. Використані технічні засоби

Для розробки і тестування ігрового додатку було використано персональний компьютер з наступною конфігурацією:

– ЦП: AMD Ryzen 7 3700x 3.6GHz;

– ОЗП: 16Гб;

– диск SSD: Kingstion A400 480Gb;

– відеокарта NVIDIA Geforce RTX 2060 Super 8Gb GDDR6;

- операційна система: 64 – разрядна Windows 11;
- дисплей:
  - частота оновлення 165Гц;
  - роздільна здатність: 1920x1080;
  - формат кольору: RGB;
  - співвідношення сторін: 16:9
- стандартна клавіатура;
- стандартна миша.

Розроблений додаток тестувався, переважно, з використанням представленої вище конфігурації. Ця конфігурація є рекомендованою для уникнення можливих помилок під час експлуатації.

2.6.2. Використані програмні засоби

Під час розробки ігрового застосунку використано наступні програмні засоби:

- Visual Studio;
- Adobe Audition;
- Adobe Photoshop;
- система Git.
- Github.

Visual Studio [31] – середовище розробки програмного забезпечення, що надається компанією Microsoft. Застосовується для створення додатків на таких мовах програмування як C#, C++, TypeScript, JavaScript. Має пакетний менеджер NuGet [32], який дозволяє встановлювати бібліотеки в ході розробки.

В контексті розробки застосунків на рушії Unity, Visual Studio використовується для написання та редагування сценаріїв. Інтеграція середовища та Unity дозволяє налагоджувати ігрові скрипти безпосередньо з IDE, використовуючи точки зупину, та інші можливості Visual Studio.

Adobe Audition [33] – програмне забезпечення призначене для обробки, редагування та створення аудіо – ресурсів. В даній роботі цей інструмент використовується для створення аудіодоріжок, які реалізуються в коді в якості звукових ефектів застосунку.

Adobe Photoshop [34] – програмний додаток, який дозволяє створювати та редагувати растрові зображення, та ефекти. Під час розробки був використаний для створення 2D зображень віртуального середовища застосунку.

Git [35] – система контролю версій, що застосовується для управління версіями коду, зокрема в ігрових застосунках. Надає можливість відстежувати історію змін, використовувати разгалуження для розробки функцій або виправлення помилок, що знижує ризик впливу на основну гілку коду. Окрім цього, млжливо виконувати відкати або злиття файлів.

В поточній кваліфікаційній роботі, є важливим інструментом керування та відстеження потенційно можливих помилок під час розробки застосунку.

GitHub [36] є сервісом для хостингу репозиторіїв Git, дозволяє зберігати файли проекту, забезпечувати доступ з будь – якого комп'ютера, що має доступ до віддаленного репозиторія. Використаний для зберігання та відстеження коду під час розробки.

2.6.3. Виклик та завантаження програми

Зібраний додаток Unity може бути запущений з носія даних, таких, як жорсткий диск, зовнішній USB – накопичувач. Додаток встановлюється на жорсткий диск, та викликається через ярлик.

Оперативна пам'ять (RAM) використовується для зберігання даних, необхідних для виконання застосунку в режимі реального часу, та в кожен момент часу залежить від кількості об'єктів на поточній сцені та текстур, а також симуляції фізики та анімацій.

Для запуску та коректного використання програми необхідно мати:

– Visual C++ Redistributable;

– DirectX 11;

Слід зазначити, що для успішного запуску застосунку необхідна операційна система, сумісна з додатками, які розроблені на рушії Unity. Наприклад, додаток може бути встановлений та запущений на Windows, macOS, Linux та інших платформах, які підтримують Unity.

2.6.4. Опис інтерфейсу користувача

Розроблене віртуальне середовище містить перелік ігрових кімнат, а саме: коридор, бойлерна, спальна кімната та стартова кімната.

Початкова кімната (рис. 2.21) є безпечною зоною для гравця, де існує джерело світла. Користувач може вертатися до цієї кімнати, щоб обнулити процес механіки «темрява» та продовжити гру.

До персонажа прикріплений ліхтарик, який створює промені світла навколо.

Рис. 2.21. Початкова кімната

Користувач може взаємодіяти з об’єктом «двері» та «записка» в цій ігровій кімнаті, при чому з’являється іконка взаємодії з об’єктами (рис 2.22– 2.23).

Рис 2.22. Взаємодія з запискою

Рис 2.23. Іконка взаємодії з дверима

Записка містить наративну інформацію до локацій гри, а також підказку що герой повинен виконувати для проходження квестів (рис. 2.24). Після з’являється записка перед головною камерою. Цей елемент можливо згорнути натисканням будь – якої кнопки миші, при чому записка пропаде з шухляди.

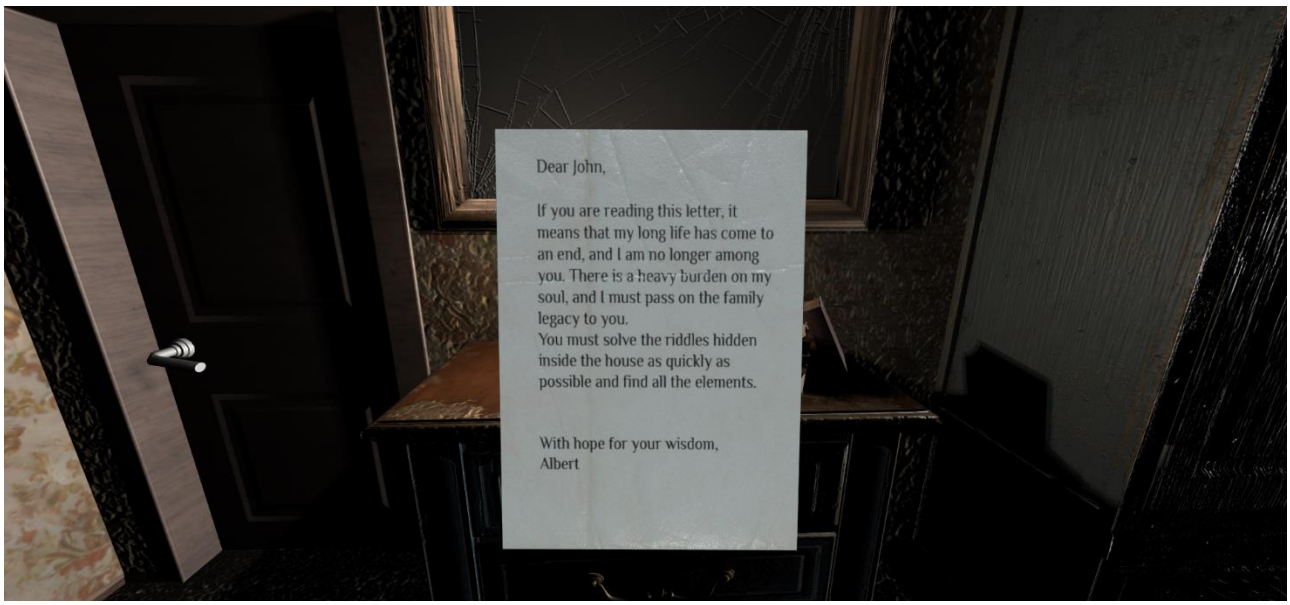

Рис 2.24. Ігрова підказка

Зв’язком між основними ігровими кімнатами є коридор, що представлений на рис. 2.25. Недоступні для користувача місця закриті темрявою (рис. 2.26). Так, при потраплянні в цю зону модель персонажа потрапляє до стартової кімнати.

Рис 2.25. Локація «Коридор»

Рис 2.26. Недоступна зона

Іншою кімнатою є «Болйерна». Як зображено на рис. 2.27 – 2.29, необхідно вирішити головоломку з об’єктом «вентиль». При цьому, зображуються елементи керування об’єктом.

Окрім зазначеного, кімната містить предмети необхідні для починки «Електрощит» (рис. 2.30).

Рис. 2.27. Бойлерна, елемент керування

Рис. 2.28. Вид камери при активації завдання

Рис. 2.29. Додатковий елемент управління об'єктом

При наявності усього переліку предметів в інвентарі персонажу, що були забрані в кімнаті «Бойлерна» (рис. 2.31), та з умовою виконанного квесту «Бойлер», стає доступною активація об’єкту «Електрощит» (рис. 2.32). Це є тригером для появи компоненту «ключ» для доступу до наступної ігрової кімнати (рис. 2.33).

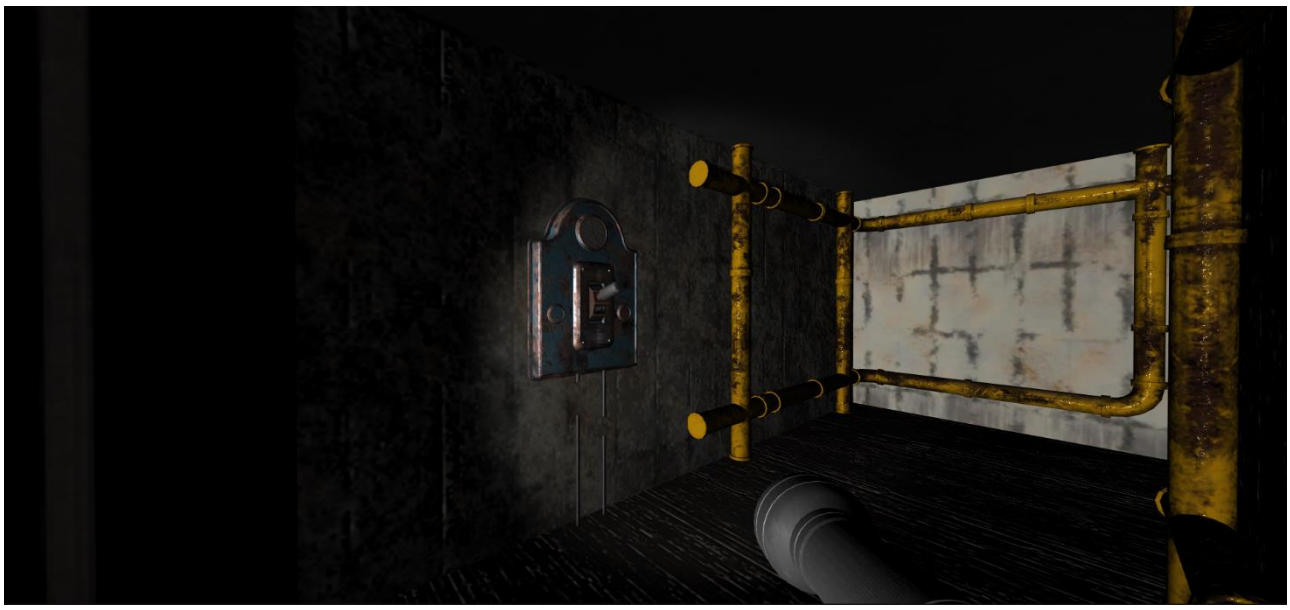

Рис. 2.30. «Електрощит» в неактивному стані

Рис. 2.31. Предмет «Ізолента»

Рис. 2.32. Активований «Електрощит»

Рис. 2.33. Компонент «Ключ»

«Спальна кімната» представляє основні ігрові механіки, такі як «картина» та «сейф». Для активації «Картина» необхідно зібрати усі частини, що розташовані в різних куточках кімнати (рис. 2.35 – 2.37). Цей квест тісно пов’язаний з механікою «годинник» (рис. 2.38) та «сейф» (рис. 2.39) і є частиною його вирішення.

Рис. 2.34. Ігрова кімната – «Спальна»

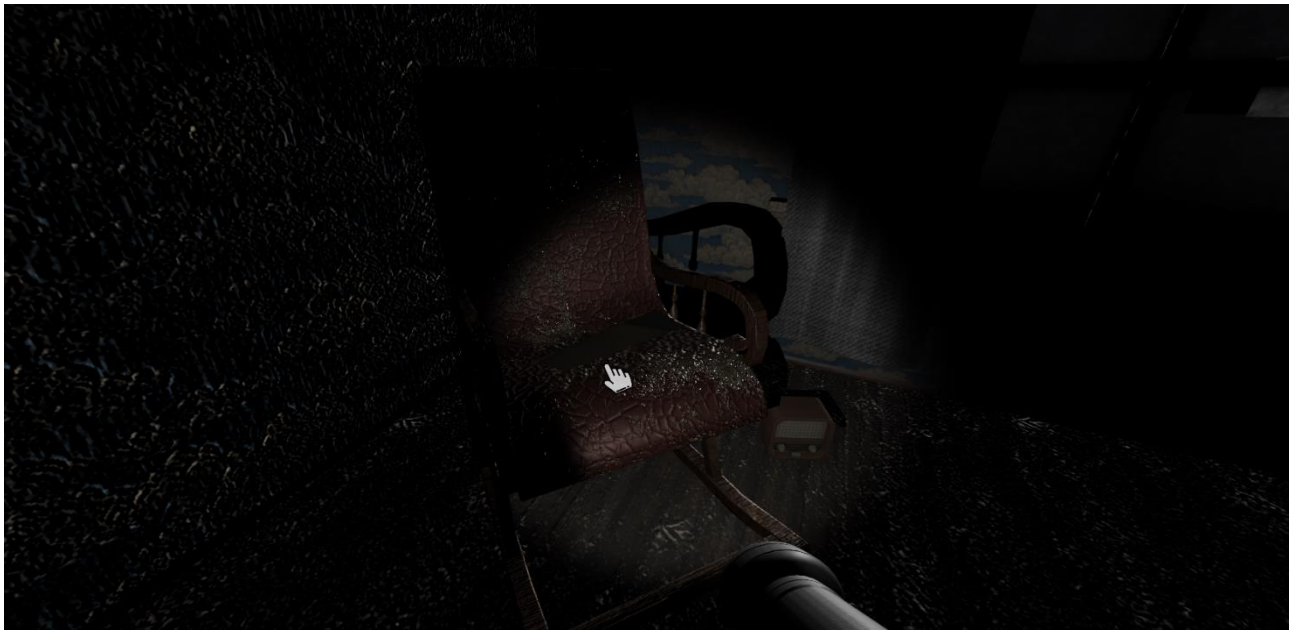

Рис. 2.35. Перший елемент картини

Рис. 2.36. Другий елемент картини

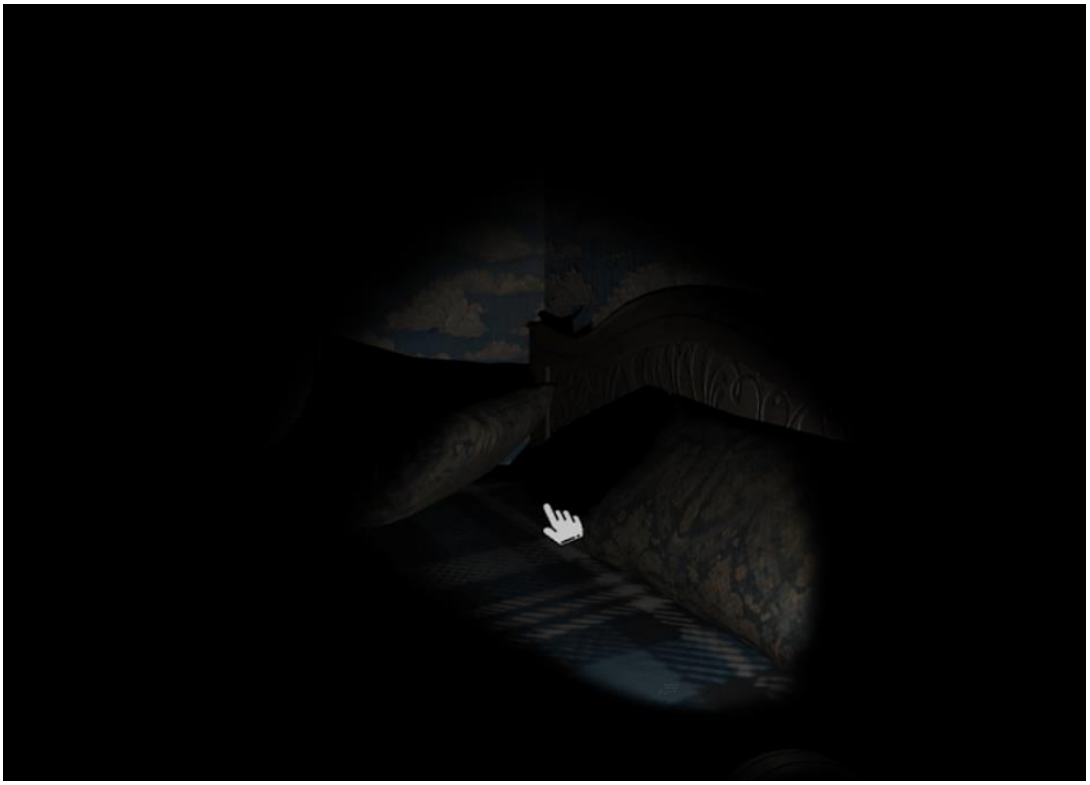

Рис. 2.37. Третій елемент картини

Так, як показано на рис. 2.38, годинник є відкриттим. Для досягнення цього стану об’єкту, необхідно розташувати стрілки в порядку часу, який зазначено в листі – підказці після відкриття об’єкту «сейф».

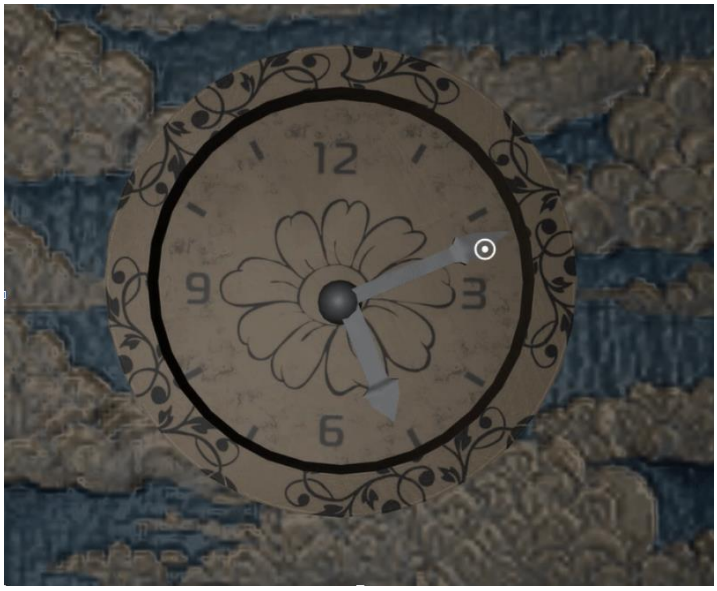

Рис. 2.38. Об’єкт «Годинник»

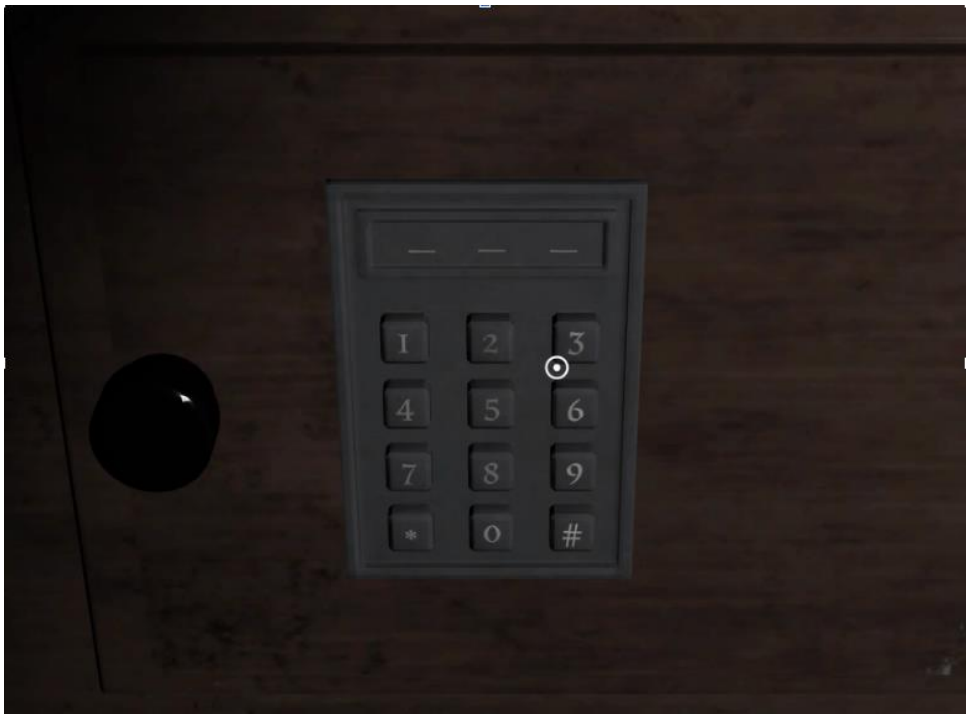

Рис. 2.39. Вид головоломки «Сейф»

Щоб користувач міг дізнатися порядок введення трьохзначного коду, необхідно виконати головломку «гральні карти», що зображено на рис. 2.40. Останні елементи, що залишаться при виборі наявних пар, будуть ключем до розв’язку завдання.

Рис. 2.40. Вид головоломки «Гральні карти»

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Визначення трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 2500;
2. коефіцієнт складності програми – 1,8;
3. коефіцієнт корекції програми в ході її розробки – 0,1;
4. годинна заробітна плата програміста – 504,84 грн/год;

Виходячи з статистичних даних, що були взяті з сайту jooble [37], станом на 29 травня 2024 середня заробітна плата позиції Unity Developer становить 1996,91$, що при курсі валют долара до гривні (40,45 грн), згідно Національного банку України [38] прирівнюється до 80775 грн. Тоді, середня заробітна плата програміста в даній галузі становитиме 504,84 грн/год.

5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,2;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,2;
7. вартість машино-години ЕОМ – 1,59 грн/год.

Розрахунок вартості витрат проведена з урахуванням часу затраченого на написання програми, що тривала впродовж 9 місяців. Ціна за електроенергію в Україні становить 2,64 грн кВт/год з урахуванням ПДВ [39]. Виходячи зі споживання електроенергії пристроєм, на якому велася розробка, а саме 125Вт на годину, витрати під час роботи на електроенергію складають 0,125*2,64=0,33. Додатково, підключення до інтернет – провайдеру «Київстар» проводиться за умовами тарифного плану «Ваш Оптимум» [40], щомісячна оплата якого складає 250 грн. Таким чином, вартість використання погодинно становить 0,35 грн. Обслуговування персонального комп’ютера вартістю 40000 грн, гарантована

тривалість роботи якого, складає 5 років. Тобто, погодинне значення використання складатиме 0,91 грн. Отже, загальна вартість ЕОМ дорівнює 1,59 грн.

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 людино-годин);

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 програмному забезпеченні, 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 (1,8);

p - коефіцієнт корекції програми в ході її розробки (0,1).

Звідси умовне число операторів в програмі:

$$Q = 1{,}8 \cdot 2500 \cdot (1 + 0{,}1) = 4950$$

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

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

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. При стажі роботи від 3 до 5 років він складає 1,2.

Приймемо збільшення витрат праці внаслідок недостатнього опису завдання не більше 50% (B = 1.2). З урахуванням коефіцієнта кваліфікації k = 1,2, отримуємо витрати праці на вивчення опису завдання:

$$t_u = (4950 \cdot 1{,}2) / (75 \cdot 1{,}2) = 66 \text{ людино-годин}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20\ldots25) \cdot k}, \text{ людино-годин,} \qquad (3.4)$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Підставивши відповідні значення в формулу (3.4), отримаємо:

$$t_a = 4950 / (20 \cdot 1{,}2) = 206{,}25 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20\ldots25) \cdot k}, \text{ людино-годин,} \qquad (3.5)$$

$$t_n = 4950 / (25 \cdot 1{,}2) = 165 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5) \cdot k}, \text{ людино-годин,} \qquad (3.6)$$

$$t_{отл} = 4950 / (5 \cdot 1{,}2) = 825 \text{ людино-годин.}$$

– за умови комплексного налагодження завдання:

$$t^k_{отл} = 1{,}5 \cdot t_{отл}, \text{ людино-годин,} \qquad (3.7)$$

$$t^k_{отл} = 1{,}5 \cdot 825 = 1237{,}5 \text{ людино-годин.}$$

Витрати праці на підготовку документації визначаються за формулою:

$$t_д = t_{др} + t_{до}, \text{ людино-годин,} \qquad (3.8)$$

де $t_{др}$ - трудомісткість підготовки матеріалів і рукопису:

$$t_{др} = \frac{Q}{(15..20) \cdot k}, \text{ людино-годин,} \qquad (3.9)$$

$t_{до}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0{,}75 \cdot t_{др}, \text{ людино-годин,} \qquad (3.10)$$

Підставляючи відповідні значення, отримаємо:

$$t_{др} = 4950 \ / (18 \cdot 1{,}2) = 229{,}17 \text{ людино-годин.}$$

$$t_{до} = 0{,}75 \cdot 229{,}17 = 171{,}88 \text{ людино-годин.}$$

$$t_{д} = 229{,}17 + 171{,}88 = 401{,}1 \text{ людино-годин.}$$

Повертаючись до формули (3.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

$$t = 50 + 66 + 206{,}25 + 165 + 825 + 401{,}1 = 1\,713{,}35 \text{ людино-годин.}$$

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ $К_{ПО}$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

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

Заробітна плата виконавців 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 60 грн / год, отримуємо:

$$З_{ЗП} = 1\ 713{,}35 \cdot 504{,}84 = 864\ 967{,}614 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

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

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

$C_{мч}$ - вартість машино-години ЕОМ, грн/год (0,68).

Підставивши в формулу (3.12) відповідні значення, визначимо вартість необхідного для налагодження машинного часу:

$$З_{мв} = 825 \cdot 1{,}59 = 1311{,}75 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{ПО} = 1311{,}75 + 864967{,}614 = 866\ 279{,}364 \text{ грн.}$$

Очікуваний період створення ПЗ:

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

де B_k - число виконавців (дорівнює 1);

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні F_p=176 годин);

t – загальна трудомісткість, людино годин()

Звідси витрати на створення програмного продукту:

$$T = 1\ 713{,}35 / 1 \cdot 176 \approx 9{,}73 \text{ міс.}$$

Висновок: для розробки ігрового застосунку з використанням рушія Unity загальна трудомісткість становить 1 713,35 людино-годин, тому очікуваний період створення програмного додатку становитиме, приблизно, 9,73 місяців, а сумарні витрати на створення - 866 279,364грн.

ВИСНОВКИ

Згідно завданню кваліфікаційної роботи та поставленої мети, розроблено ігровий застосунок для інтерактивного навчання.

Розглянуті рушії Unreal та Unity пропонують інструментарій для створення сценаріїв виконання компонентів застосунку, включаючи засоби для роботи з графічними елементами. Однак, Unreal Engine націлений на створення масштабних продуктів, і потребує більшу кількість апаратних ресурсів, зокрема уваги до оптимізації компонентів. Так, Unity Engine забезпечує продуктивність та доступність на менш потужних пристроях.

Проведений аналіз існуючих рішень на ринку ігрової індустрії показує, що найбільшим попитом користуються додатки жанру пригод, зокрема його піджанри. Кожне з розглянутих рішень пропонує унікальні ігрові механіки, проте мають спільні риси, такі як дослідження гравцем віртуального середовища та вирішення ігрових завдань. Однак, на відміну від «Myst» та «Nancy Drew», «Alan Wake» пропонує елементи жанру горор, які відсутні в двох інших розглянутих застосунках. Таким чином, поєднання жанру пригод з елементами горору сприятиме залученню користувачів до ігрового процесу.

Практичне значення полягає у створенні тривимірного проекту, який дозволить керувати ігровим процесом, та взаємодіяти з ігровими об’єктами, а також забезпечить реалізацію ігрових механік. Отже, користувачу надається можливість:

- проходження та виконання віртуальних завдань;
- керування персонажем;
- вибір послідовності дій та прийняття рішень;
- взаємодія з ігровими об’єктами та компонентами;
- ігрові механіки, що визначають перебіг подій ігрового процесу.

Графічний інтерфейс має мінімальну кількість елементів, що переважно включає іконки взаємодії з об’єктами сцени. Аудіоефекти інформують про найважливіші події в ході ігрового процесу.

Розрахунки економічного розділу показують, що трудомісткість проекту становить 1713,35 людино-годин, а очікуваний період створення застосунку становить 9,73 місяці. Витрати мають значення - 866 279,364 грн.

В подальшому розвитку застосунку планується розвиток ігрових механік, та збільшення загальної тривалості гри.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Головна сторінка Unreal Engine. / URL: https://www.unrealengine.com/en-US/. Дата звернення: 25.05.2024.
2. Unity Engine / URL: https://unity.com/. Дата звернення 25.05.2024.
3. Документація Unreal Engine / URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/blueprints-visual-scripting-in-unreal-engine. Дата звернення: 25.05.2024.
4. Unity architecture / URL: https://docs.unity3d.com/Manual/unity-architecture.html. Дата звернення: 13.05.2024.
5. Unity core modules / URL: https://docs.unity3d.com/ScriptReference/UnityEngine.CoreModule.html. Дата звернення: 13.05.2024.
6. Sivalaya G, Mounika B, Sailasya G, Kumar NS. Implementation of Augmented Reality Application using Unity Engine Deprived of Prefab, Volume 83, Page Number: 20079 – 20082, March - April 2020, ISSN: 0193-4120, Department of Computer Science and Engineering, GITAM Institute of Technology, Visakhapatnam. Published by The Mattingley Publishing Co., Inc.
7. Šmíd A. Comparison of Unity and Unreal Engine. Faculty of Electrical Engineering, Department of Computer Graphics and Interaction, Field of Study: STM, Web and Multimedia, May 2020.
8. Qaffas AA. An Operational Study of Video Games' Genres. iJIM – Vol. 14, No. 15, 2020. University of Jeddah, Jeddah, Saudi Arabia. DOI: 10.3991/ijim.v14i15.16691.
9. Collins G, Winardy B, Septiana E. Role, play, and games: Comparison between role-playing games and role-play in education. Social Sciences & Humanities Open 8 (2023) 100527. Faculty of Psychology, University of Indonesia, Jl. Lkr. Kampus Raya Jl, Prof. DR. R Slamet Iman Santoso, Pondok Cina, Kecamatan Beji, Kota, Depok, 16424, Indonesia.

10. Tinwell A, Grimshaw M. Survival horror games - an uncanny modality. 2019.

11. Myst / URL: https://store.steampowered.com/app/1255560/Myst/. Дата звернення: 15.05.2024.

12. Alan Wake / URL: https://store.steampowered.com/app/108710/Alan_Wake/. Дата звернення: 15.05.2024.

13. Nancy Drew / URL: https://store.steampowered.com/app/31800/Nancy_Drew_Danger_by_Design/?l=ukrainian. Дата звернення: 23.05.2024.

14. ESRB Raiting. Офіційна сторінка / URL: https://www.esrb.org/. Дата звернення: 15.05.2024.

15. Information security / URL: https://en.wikipedia.org/wiki/Information_security. Дата звернення: 23.05.2024.

16. Perlin noise / URL: https://en.wikipedia.org/wiki/Perlin_noise. Дата звернення: 23.05.2024.

17. NVIDIA Developer, Implementing Improved Perlin Noise / URL: https://developer.nvidia.com/gpugems/gpugems2/part-iii-high-quality-rendering/chapter-26-implementing-improved-perlin-noise. Дата звернення 23.05.2024.

18. Jia YB. Quaternions. Com S 477/577 Notes, Dec 8, 2022.

19. Griffin S. Quaternions: Theory and Applications. Publication Date: March 2019. ISBN: 978-1-53610-768-5.

20. Unity Editor / URL: https://docs.unity3d.com/ScriptReference/Editor. Дата звернення: 23.05.2024.

21. Parviainen N. Dependency Injection in Unity3D. Bachelor’s thesis, March 2017. Degree Programme in Software Engineering, Technology, Communication, and Transport.

22. Chen J, Price E. Game Development with Unity for .NET Developers: The ultimate guide to creating games with Unity and Microsoft Game Stack. Packt Publishing Ltd, 2022. ISBN 1801075077, 9781801075077. 584 pages.

23. Hoang TD, Low KL. Efficient screen-space approach to high-quality multiscale ambient occlusion. Computer Science Department, National University of Singapore, Singapore. 27 September 2019, Volume 28, pages 289–304. DOI: /10.1007/s00371-011-0639-y.

24. Akenine-Möller T, Haines E, Hoffman N. Real-Time Rendering. New York, 2019. ISBN 9781315365459.

25. Blender / URL: https://www.blender.org/. Дата звернення : 23.05.2024.

26. ZBrush / URL: https://www.maxon.net/en/zbrush/. Дата звернення: 23.05.2024.

27. 3ds Max / URL: https://www.autodesk.com/products/3ds-max/overview?term=1-YEAR&tab=subscription/. Дата звернення 23.05.2024.

28. Asset Store / URL: https://assetstore.unity.com/. Дата звернення 23.05.2024.

29. Akenine-Möller T, Haines E, Hoffman N. Real-Time Rendering. New York, 2019. ISBN 9781315365459.

30. Jackson S. Mastering Unity 2D Game Development. Community experience distilled. Packt Publishing Ltd, 2019. ISBN 1849697353, 9781849697354. 474 pages.

31. Visual Studio / URL: https://visualstudio.microsoft.com/. Дата звернення: 26.05.2024.

32. NuGet / URL: https://www.nuget.org/. Дата звернення: 26.05.2024.

33. Adobe Audition / URL: https://www.adobe.com/ua/products/. Дата звернення 26.05.2024.

34. Adobe Photoshop / URL: https://www.adobe.com/ua/products/photoshop-express.html. Дата звернення: 26.05.2024.

35. Skoulikari A. Learning Git. O'Reilly Media, Inc., 2023. ISBN 1098133870, 9781098133870. 320 pages.

36. GitHub Essentials. Packt Publishing Ltd., September 2015. Production reference: 1280915. ISBN 978-1-78355-371-6. Published by Packt Publishing Ltd., Livery Place, 35 Livery Street, Birmingham B3 2PB, UK.

37. Unity Developer: середня зарплата в Україні / URL: https://ua.jooble.org/salary/unity-developer#hourly. Дата звернення: 26.05.2024.

38. Національний банк України / URL: https://bank.gov.ua/. Дата звернення: 26.05.2024.

39. Офіційний сайт YASNO / URL: https://yasno.com.ua/b2c-tariffs. Дата звернення 10.05.2024.

40. Тарифи Київстар / URL: https://kyivstar.ua/archive/tariffs/vash-optimum. Дата звернення: 10.05.2024.

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

CameraManager.cs

```
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.CompilerServices;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.UI;
public class CameraManager : MonoBehaviour
{
    [SerializeField] private CollisionManager collisionManager;
    [SerializeField] private List<CameraSwitcher> camera_list;
    public static CameraManager Instance { get; private set; }
    private CameraSwitcher currentSwitcher = null;
    private void Awake()
    {
        collisionManager = GameObject.Find("CollisionManager").GetComponent<CollisionManager>();

        if(Instance == null) Instance = this;
    }
    private void Start()
    {
        foreach (var cameraSwitcher in camera_list)
        {
            cameraSwitcher.Initialization();
        }
    }
    private void Update()
    {
        if (camera_list != null && camera_list.Count > 0)
        {
            string capName = null;
            CameraSwitcher tempSwitcher = null;
            manageCamerasInactive();
            manageCamerasActive(capName, tempSwitcher);
        }
```

```
    }
    private void manageCamerasActive(string capName, CameraSwitcher tempSwitcher)
    {
        if (MouseOverChecker.objectName != null)
        {
            tempSwitcher = camera_list.Find(camera => ((capName = MouseOverChecker.objectName) != null) &&
camera.captureName == capName);
            currentSwitcher = tempSwitcher;

            if (currentSwitcher != null && collisionManager.IsIntersect() &&
GameObject.Find(capName).transform.IsChildOf(GameObject.Find(collisionManager.GetIntersectedName()).transfor
m))
            {
                currentSwitcher.Controller.ShowPointer();
                currentSwitcher.ChangeCameraActive();
            }
        }
    }
    private void manageCamerasInactive()
    {
        if (!MouseOverChecker.mouseOn || !collisionManager.IsIntersect())
        {
            camera_list.FirstOrDefault().Controller.HidePointer();
            if (currentSwitcher != null && currentSwitcher.GetIsSwitched())
            {
                currentSwitcher.ChangeCameraInactive();

                if (!currentSwitcher.GetIsSwitched()) currentSwitcher = null;
            }
        }
    }
    public void manageImmediateInactive()
    {
        if (!MouseOverChecker.mouseOn || !collisionManager.IsIntersect())
        {
            camera_list.FirstOrDefault().Controller.HidePointer();
            if (currentSwitcher != null && currentSwitcher.GetIsSwitched())
            {
                currentSwitcher.ChangeImmediateCameraInactive();
                if (!currentSwitcher.GetIsSwitched()) currentSwitcher = null;
            }
```

```
        }
    }
    public void CameraRemove(Camera camera)
    {
        camera_list.Remove(camera_list.Find(match => match.secondCamera == camera));
    }
}
```

CameraSwitcher.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEditor.UI;
using UnityEngine;
[Serializable]
public class CameraSwitcher
{   public string captureName;
    public Collider CaptureCollision;
    public Camera mainCamera, secondCamera;
    public PointerController Controller;
    public ModelControl light;
    private bool isSwitched;
    public void ChangeCameraActive()
    {
        if (Input.GetKeyDown(KeyCode.Mouse0))
        {
            if (mainCamera.gameObject.activeSelf)
            {
                isSwitched = true;
                light.ChangeControllerEnable();
                Controller.ActionCursor.CursorEnable();
                if (CaptureCollision != null)
                    CaptureCollision.enabled = false;
                mainCamera.gameObject.SetActive(false);
                secondCamera.gameObject.SetActive(true);
            }
        }
    }
    public void ChangeCameraInactive()
    {       if (Input.GetKeyDown(KeyCode.Mouse1))
        {           if (!mainCamera.gameObject.activeSelf)
```

```
        {               isSwitched = false;
            light.ChangeControllerDisable();
            Controller.ActionCursor.CursorDisable();
            if (CaptureCollision != null)
                CaptureCollision.enabled = true;
            mainCamera.gameObject.SetActive(true);
            secondCamera.gameObject.SetActive(false);
        }
    }
}
public void ChangeImmediateCameraInactive()
{
    if (!mainCamera.gameObject.activeSelf)
    {
        isSwitched = false;
        light.ChangeControllerDisable();
        Controller.ActionCursor.CursorDisable();
        if (CaptureCollision != null)
            CaptureCollision.enabled = true;
        mainCamera.gameObject.SetActive(true);
        secondCamera.gameObject.SetActive(false);
    }
}
public void Initialization()
{
    secondCamera.gameObject.SetActive(false);
    captureName = CaptureCollision.gameObject.name;
}
public bool GetIsSwitched()
{
    return isSwitched;
}
}
```

CollisionManager.cs

```
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Reflection;
using UnityEngine;
public class CollisionManager : MonoBehaviour
```

```
{
    public static CollisionManager Instance { get; private set; }
    public GameObject PlayerModel;
    [SerializeField] private List<Transform> gameModels;
    private bool isIntersecting;
    private string intersected_name;
    public event System.Action OnIntersect;
    // Start is called before the first frame update
    private void Awake()
    {
        intersected_name = null;
        if (Instance == null) Instance = this;
    }
    private void Update()
    {
        if (gameModels != null && PlayerModel != null)
        {
            isIntersecting = false;
            intersected_name = null;
            foreach (Transform model in gameModels)
            {
                if (model != null)
                {
                    if
(model.Find("trigger_collider").GetComponent<Collider>().bounds.Intersects(PlayerModel.GetComponent<Collider>(
).bounds))
                    {
                        isIntersecting = true;
                        intersected_name = model.name;
                        if(OnIntersect != null)
                            OnIntersect.Invoke();
                        //Debug.Log("CollisionManager:" + intersected_name);
                        break;
                    }
                }
            }
        }
    }
    public bool IsIntersect()
    {
        return isIntersecting;
```

```
    }
    public string GetIntersectedName()
    {
        return intersected_name;
    }
    public string GetIntesectedParentName()
    {
        return GameObject.Find(intersected_name).transform.parent.name;
    }
    public void gameModelRemove(Transform model)
    {
        gameModels.Remove(model);
    }
}
```

ModelControl.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class ModelControl : MonoBehaviour
{
    public GameObject player;
    // Start is called before the first frame update
        public void ChangeControllerEnable()
    {
        if (player != null)
        {
            player.SetActive(false);
            this.gameObject.SetActive(true);
        }
    }
    public void ChangeControllerDisable()
    {
        if (player != null)
        {
            player.SetActive(true);
            this.gameObject.SetActive(false);
        }
    }
}
```

PointerController.cs

```
using Palmmedia.ReportGenerator.Core.Reporting.Builders;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using Unity.VisualScripting;
using UnityEngine;
public class PointerController : MonoBehaviour
{
    public GameObject pointer, hand_pointer;
    public MouseCursor ActionCursor;
    public static PointerController Instance { get; private set; }
    public List<MouseOverChecker> pointerTarget;
    // Start is called before the first frame update
    private void Awake()
    {
        if (Instance == null) Instance = this;
    }
    void Start()
    {
        pointer.SetActive(false);
    }
    public void HidePointer()
    {
        pointer.SetActive(false);
    }
    public void ShowPointer()
    {
        pointer.SetActive(true);
    }
    public void TargetRemove(MouseOverChecker checker)
    {
        pointerTarget.Remove(checker);
    }
}
```

CameraLook.cs

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

```
public class CameraLook : MonoBehaviour
{
   // Start is called before the first frame update
   [AddComponentMenu("Camera-Control/CameraLook")]
   public enum RotationAxes { MouseXY = 0, MouseX = 1, MouseY = 2};
   public RotationAxes axes = RotationAxes.MouseXY;
   public float sensitivityX = 2, sensitivityY = 2;
   public float minimumX = -360, maximumX = 360,
            minimumY = -360, maximumY = 360;
   private float rotationX = 0, rotationY = 0;
   Quaternion rotation;
   void Start()
   {
      if (GetComponent<Rigidbody>()) GetComponent<Rigidbody>().freezeRotation = true;
      rotation = transform.localRotation;
   }
   public static float ChangeAngel(float angle, float min, float max)
   {
      if(angle < -360F) angle += 360;
      if (angle > 360F) angle -= 360;

      return Mathf.Clamp(angle, min, max);
   }
   // Update is called once per frame
   void Update()
   {
      if(axes == RotationAxes.MouseXY)
      {
         rotationX = Input.GetAxis("Mouse X") * sensitivityX;
         rotationY = Input.GetAxis("Mouse Y") * sensitivityY;
         rotationX = ChangeAngel(rotationX, minimumX, maximumX);
         rotationY = ChangeAngel(rotationY, minimumY, maximumY);
         Quaternion xQuartenion = Quaternion.AngleAxis(rotationX, Vector3.up);
         Quaternion yQuartenion = Quaternion.AngleAxis(-rotationY, Vector3.right);
         transform.localRotation= rotation * xQuartenion * yQuartenion;
      }
      else if(axes == RotationAxes.MouseX){
         rotationX += Input.GetAxis("Mouse X") * sensitivityX;
         rotationX = ChangeAngel(rotationX, minimumX, maximumX);
         Quaternion xQuartenion = Quaternion.AngleAxis(rotationX, Vector3.up);
```

```
            transform.localRotation = rotation * xQuartenion;
        }
        else if (axes == RotationAxes.MouseY)
        {
            rotationY += Input.GetAxis("Mouse Y") * sensitivityY;
            rotationY = ChangeAngel(rotationY, minimumY, maximumY);
            Quaternion yQuartenion = Quaternion.AngleAxis(-rotationY, Vector3.right);
            transform.localRotation = rotation * yQuartenion;
        }
    }
}
```

CharacterControl.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEditor.UI;
using UnityEngine;
public class CharacterControl : MonoBehaviour
{
    // Start is called before the first frame update
    public bool isMoving { get; private set; } = false; // Ôëàã äëÿ îòñëåæèâàíèÿ äâèæåíèÿ
    public float speed = 4.0f,
                 jumpSpeed = 8.0f,
                 gravity = 20.0f;
    private Vector3 moveDirection = Vector3.zero;
    private CharacterController controller;
    void Start()
    {
        controller = GetComponent<CharacterController>();
    }
    // Update is called once per frame
    void Update()
    {
        if (controller.isGrounded)
        {
            isMoving = (Input.GetAxis("Horizontal") != 0 || Input.GetAxis("Vertical") != 0);
            moveDirection = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical"));
            moveDirection = transform.TransformDirection(moveDirection);
            moveDirection *= speed;
        }
        moveDirection.y -= gravity * Time.deltaTime;
```

```
        controller.Move(moveDirection * Time.deltaTime);
    }
}
```

HeadTorchControl.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class HeadTorchControl : MonoBehaviour
{
    public GameObject headTorch;
    public AudioSource torchEnable, torchDisable;
    // Start is called before the first frame update
    private void Start()
    {
        headTorch.SetActive(false);
    }
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.F) && headTorch.activeSelf)
        {
            headTorch.SetActive(false);
            torchDisable.Play();
        }
        else if (Input.GetKeyDown(KeyCode.F) && !headTorch.activeSelf)
        {
            headTorch.SetActive(true);
            torchEnable.Play();
        }
    }
}
```

TeleportHandler.cs

```
using System.Collections;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using UnityEngine;
public class TeleportHandler : MonoBehaviour
{
    private SoundTeleport handler;
    public GameObject player;
```

```
    private void Start()
    {
        handler = GetComponent<SoundTeleport>();
        handler.TeleportAction += handleTeleportEvent;
    }
    private void handleTeleportEvent()
    {
        Debug.Log("Whyy");
        Vector3 vec = new Vector3(-6.416f, -0.683f, -3.397f);
        player.GetComponent<CharacterControl>().enabled = false;
        player.GetComponent<CharacterController>().enabled = false;
        player.transform.localPosition = vec;
        player.GetComponent<CharacterControl>().enabled = true;
        player.GetComponent<CharacterController>().enabled = true;
    }
}
```

DetailManager.cs

```
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;
public class DetailManager : MonoBehaviour
{
    public List<bool> dlist;
    public bool isAll { get; private set; }

    private void Awake()
    {
        dlist = new List<bool>();
    }
    private void FixedUpdate()
    {
        if (!isAll && dlist.Count == 3)
            isAll = true;
    }
    public bool RepairObj()
    {
        if (isAll && dlist.Count == 3)
            return true;
        else return false;
```

```
    }
}
```

DetailController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class DetailController : MonoBehaviour, IClickable
{
    public DetailManager manager;
    public GameObject detail;
    public AudioSource GrabItem;
    public bool ClickOn()
    {
        manager.dlist.Add(true);
        Destroy(detail);
        GrabItem.Play();
        NewPointerController.HidePointer();
        return true;
    }
}
```

GazManager.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class GazManager : MonoBehaviour, IClickable
{
    public AudioSource lightFire, fullFire, startFire, rotate;
    public CameraManager manager;
    public RepairBoiler control;
    private int wheelCount = 0;
    private bool isPressed = false;
    public void MakeAction()
    {
        if (wheelCount < 1)
            wheelCount++;
        else if (isPressed)
            wheelCount++;
        switch(wheelCount) {
            case 1:
```

```
            startFire.Play();
            rotate.Play();
            manager.manageImmediateInactive();
            break;
        case 3:
            fullFire.Play();
            lightFire.Stop();
            control.isEnabled = true;
            break;
        default:
            break;
    }
  }
  public bool ClickOn()
  {
    startFire.Stop();
    lightFire.Play();
    NewPointerController.HidePointer();
    return isPressed = true;
  }
}
```

RepaitBoiler.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class RepairBoiler : MonoBehaviour, IClickable
{
  public GameObject oldShield, repairedShield;
  public AudioSource source, shieldEnable;
  public GameObject key;
  public DetailManager manager;
  public bool isEnabled;
  // Update is called once per frame
  public bool ClickOn()
  {
    if (manager.isAll && isEnabled)
    {
      shieldEnable.Play();
      oldShield.SetActive(false);
      repairedShield.SetActive(true);
```

```
            source.Play();
            key.SetActive(true);
            NewPointerController.HidePointer();
            return true;
        }
        return false;
    }
}
```

WheelController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class WheelController : MonoBehaviour
{
    public GazManager gazManager;
    public float sensitivity = 1f;

    private bool isRotating = false;
    private Vector3 lastMousePosition;
    [SerializeField] private float totalRotation = 0f;
    private void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            isRotating = true;
            lastMousePosition = Input.mousePosition;
        }
        if (Input.GetMouseButtonUp(0))
        {
            isRotating = false;
        }
        if (isRotating)
        {
            Vector3 mouseDelta = Input.mousePosition - lastMousePosition;
            float rotationZ =  Mathf.Clamp(mouseDelta.x * sensitivity, 0, 2);
            totalRotation += rotationZ;
            transform.Rotate(Vector3.forward, rotationZ);
            if(totalRotation >= 360)
            {
                totalRotation = 0f;
```

```
            gazManager.MakeAction();
        }
        lastMousePosition = Input.mousePosition;
    }
  }
}
```

KeyController.cs

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

public class KeyController : MonoBehaviour, IClickable
{
  public GameObject key;
  public AnimationControl control;
  public bool ClickOn()
  {
    control.isLocked = false;

    key.SetActive(false);
    NewPointerController.HidePointer();

    return true;
  }
}
```

PlayingCards.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class PlayingCards : MonoBehaviour
{
 public Queue<GameObject> cards;
  public Queue<Vector3> positions;
  private void Awake()
  {
    cards = new Queue<GameObject>();
    positions = new Queue<Vector3>();
  }
  public void AddToQueue(GameObject card, Vector3 originPosition){
```

```
        cards.Enqueue(card);
        positions.Enqueue(originPosition);
    }
    private void Update()
    {
        if(cards.Count == 2) {
            GameObject firstItem = cards.Dequeue(),
                       secondItem = cards.Dequeue();
            cards.Clear();
            Vector3 firstItemPosition = positions.Dequeue(),
                    secondItemPosition = positions.Dequeue();
            cards.Clear();
            if (firstItem.transform.parent.gameObject != secondItem.transform.parent.gameObject || firstItem == secondItem)
{
                firstItem.transform.position= firstItemPosition;
                secondItem.transform.position= secondItemPosition;

            }
            else
            {
                Destroy(firstItem.transform.parent.gameObject);
                Destroy(secondItem.transform.parent.gameObject);
            }
        }
    }
}
```

KeyChecker.cs

```
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using Unity.VisualScripting.Antlr3.Runtime.Tree;
using UnityEngine;
public class KeyChecker : MonoBehaviour
{
    private Queue<int> code;
    private readonly int[] correctCode = { 6, 3, 7 };
    public CameraManager manager;
    public Animator element;
    public AudioSource KeyWrong;
    private void Awake()
```

```
    {
        code = new Queue<int>();
    }
    public void AddNumber(int key)
    {
        code.Enqueue(key);
    }
    private void Update()
    {
        if(code.Count == 3) {
            for(int i = 0; i < correctCode.Length; i++)
            {
                if(code.Dequeue() != correctCode[i]) {
                    KeyWrong.Play();
                    code.Clear();
                    return;
                }
            }
            element.enabled = true;
        }
    }
}
```

KeyHandler.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class KeyHandler : MonoBehaviour
{
    public GameObject keyIcon;
    private void OnMouseOver()
    {
        if (Input.GetKeyDown(KeyCode.Mouse0))
        {
            keyIcon.SetActive(true);
            this.gameObject.GetComponent<MeshRenderer>().enabled = false;
        }
    }

}
```

CardController.cs

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

public class ClickOnCard : MonoBehaviour
{
   private Transform card;
   private GameObject cardElement;
   private Vector3 originPosition;

   public PlayingCards queue;

  // private float speed = 1f;
  // private bool clickedOn = false;

   private void Awake()
   {
      card = this.gameObject.transform.parent;
      cardElement = card.gameObject;

      originPosition = card.gameObject.transform.position;
   }

   private void OnMouseOver()
   {
      if (Input.GetKeyDown(KeyCode.Mouse0) /*&&!clickedOn*/)
      {
         //clickedOn = true;
         card.transform.position = new Vector3(0, 0.1f, 0);
         queue.AddToQueue(cardElement, originPosition);
      }
   }
 /* private void FixedUpdate()
   {
      if(clickedOn && card.transform.position.y < 0.1f)
      {
         card.transform.position = new Vector3(0, card.transform.position.y + speed * Time.deltaTime, 0);
      }
```

```
        if(card.transform.position.y > 0.1f) clickedOn = false;
    }*/

}
```

OpelClock.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class OpenClock : MonoBehaviour
{
    public Collider arrow1, arrow2;
    public Collider trigger1, trigger2, pictureCollider;
    public Animator clock_animator;
    [SerializeField] private ArrowRotator rotator, rotator2;
    private void Update()
    {
        if (arrow1.bounds.Intersects(trigger1.bounds) && arrow2.bounds.Intersects(trigger2.bounds) &&
!rotator.GetCursorLocked() && !rotator2.GetCursorLocked())
        {
            clock_animator.enabled = true;
            pictureCollider.enabled = false;
            Destroy();
        }
    }

    private void Destroy()
    {
        CameraManager cameraManager = GameObject.Find("CameraManager").GetComponent<CameraManager>();
        CollisionManager collisionManager =
GameObject.Find("CollisionManager").GetComponent<CollisionManager>();
        PointerController controller = GameObject.Find("PointerController").GetComponent<PointerController>();
        cameraManager.manageImmediateInactive();

cameraManager.CameraRemove(GameObject.Find("Non_Player_Cameras").transform.Find("ClockView").GetCompon
ent<Camera>());

        controller.pointerTarget.Remove(GameObject.Find("Clock1_low").GetComponent<MouseOverChecker>());
        collisionManager.gameModelRemove(GameObject.Find("NewClock").transform);
        Destroy(gameObject);
```

```
    }
}
```

ScareController.cs

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

public class ScareController : MonoBehaviour
{
    public GameObject playerModel;

    public LightController controller;

    [SerializeField] private bool isScared, isPlayed = false,
                                  isFaster = false, isHarded = false;

    [SerializeField] private AudioSource darkEntered, heartBeat, laugh, ring, breath;
    public float fadeDuration = 3.5f;

    public void scareEnable(){
        isScared = true;
        enableSounds();
    }
    public void scareDisable(){
        isScared = false;
    }

    private void enableSounds(){
        StartCoroutine(startSoundRoutine());
    }

    IEnumerator startSoundRoutine(){
        yield return new WaitForSeconds(2f);

        if (isScared)
        {
            darkEntered.Play();

        }
```

```
    while (darkEntered.isPlaying)
    {
        yield return null;
    }

    heartBeat.Play();

    float sTime = Time.time;
    while (heartBeat.isPlaying)
    {
        if ((Time.time - sTime) >= 35.0f && !isPlayed)
        {
            laugh.Play();
            isPlayed = true;

            StartCoroutine(IncreaseHeartBeat());
        }

        if((Time.time - sTime) >= 70.0f && !isFaster)
        {
            isFaster = true;
            StartCoroutine(IncreaseHeartBeat());
        }
        if((Time.time - sTime) >= 75.0f && isPlayed && !isHarded)
        {
            isHarded = true;
            StartCoroutine(SetFaint());
        }
        yield return null;
    }
}
private void Update()
{
    if (!isScared && heartBeat.isPlaying) {
        StartCoroutine(FadeOutHeartBeat());
    }
}
private IEnumerator FadeOutHeartBeat()
{
    float startVolume = heartBeat.volume;
```

```
    float startTime = Time.time;
    while (Time.time < startTime + fadeDuration)
    {
        heartBeat.volume = Mathf.Lerp(startVolume, 0, (Time.time - startTime) / fadeDuration);
        yield return null;
    }
    heartBeat.Stop();
    heartBeat.pitch = 1;
    heartBeat.volume = 1;
    isPlayed = false; isFaster = false; isHarded = false;
}
private IEnumerator IncreaseHeartBeat()
{
    float startVolume = heartBeat.pitch;
    float targetVolume = startVolume * 1.65f; // Óâåëè÷èâàåì ãðîìêîñòü íà ïîëîâèíó
    float startTime = Time.time;
    while (Time.time < startTime + fadeDuration)
    {
        float normalizedTime = (Time.time - startTime) / fadeDuration;
        heartBeat.pitch = Mathf.Lerp(startVolume, targetVolume, normalizedTime);
        yield return null;
    }
    heartBeat.volume = targetVolume; // Óñòàíîâèòü ãðîìêîñòü íà öåëåâîå çíà÷åíèå
}
private IEnumerator SetFaint()
{
    float sTime = Time.time;
    ring.Play();
    heartBeat.Stop();
    while (ring.isPlaying)
    {
        yield return null;
        if (Time.time - sTime >= 7)
            break;
    }
    heartBeat.pitch = 1;
    heartBeat.volume = 1;
    StartCoroutine(handleTeleport());
}
private IEnumerator handleTeleport()
{
```

```
        Debug.Log("Whyy");
        controller.DisableLights();
        yield return new WaitForSeconds(3.0f);
        Vector3 vec = new Vector3(-2.243082f, -0.6967797f, -2.455013f);
        playerModel.GetComponent<CharacterControl>().enabled = false;
        playerModel.GetComponent<CharacterController>().enabled = false;
        playerModel.transform.localPosition = vec;
       playerModel.transform.rotation = Quaternion.identity;
        playerModel.GetComponent<CharacterControl>().enabled = true;
        playerModel.GetComponent<CharacterController>().enabled = true;
        isPlayed = false; isFaster = false; isHarded = false;

        controller.EnableLights();
        breath.Play();
    }}
```

FlickeringLight.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class FlickeringLight : MonoBehaviour
{
    public float minIntensity = 1.0f; // Мінімальна інтенсивність світла
    public float maxIntensity = 2.0f; // Максимальна інтенсивність світла
    public float flickerSpeed = 1.0f; // Швидкість мерехтіння
    public float flickerAmount = 0.1f; // Амплитуда мерехтіння

    private Light pointLight;
    private float baseIntensity;

    void Start()
    {
        pointLight = GetComponent<Light>();
        baseIntensity = pointLight.intensity;

        StartCoroutine(Flicker());
    }

    IEnumerator Flicker()
    {
        while (true)
```

```
        {
            // Шум перліна для моделювання мерехтіння
            float noise = Mathf.PerlinNoise(Time.time * flickerSpeed, 0.0f) * flickerAmount;

            // Застосування мерехтіння до інтенсивності світла
            pointLight.intensity = Mathf.Lerp(minIntensity, maxIntensity, noise) * baseIntensity;

            yield return null;
        }
    }
}
```

ДОДАТОК Б

ВІДГУК

Керівника економічного розділу

на кваліфікаційну роботу бакалавра на тему:

«Розробка ігрового додатку на рушії Unity з використанням мови програмування C#»

Студента групи 122-20-4 Дубіни Єгора Сергійовича

Керівник економічного розділу

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

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файла	Опис
Пояснювальні документи	
Кваліфікаційна робота_Дубіна.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота_Дубіна.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Diplom.zip	Архів. Містить коди програми і відкомпільовану програму
Презентація	
Презентація_Дубіна.pptx	Презентація кваліфікаційної роботи