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 *Гамзіна Олександра Сергійовича*
(ПІБ)

академічної групи *122-19-4*
(шифр)

напряму підготовки *122 Комп'ютерні науки*
(код і назва напряму підготовки)

на тему: *Створення 3D-моделей персонажів та оточення та їх впровадження у ігровий рівень на Unreal Engine за допомогою мови програмування C++*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*проф. Бердник М.Г*			
розділів:				
спеціальний	*проф. Бердник М.Г.*			
економічний	*проф. Вагонова О.Г.*			
Рецензент	*доц. Шедловський І.А*			
Нормоконтролер	*Доц.Гуліна І.Г.*			

Дніпро
2023

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

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

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

« » _____ 20 ____ року

ЗАВДАННЯ
на дипломний проект

бакалавра
(назва освітньо-кваліфікаційного рівня)

студенту *122-19-4* *Гамзіну Олександру Сергійовичу*
(група) (прізвище та ініціали)

Тема дипломного проекту *Створення 3D-моделей персонажів та оточення та їх впровадження у ігровий рівень на Unreal Engine за допомогою мови програмування C++*

затверджена наказом ректора ДВНЗ «НГУ» від *16 . 05 .2023 р.* № *350-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***	***12.05.2023 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПО й тривалості його розробки***	***27.05.2023 р.***

Завдання видав _____ *проф. Бердник М.Г*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Гамзін О.С.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2023 р.*

Термін подання дипломного проекту до ДЕК *12.06.2023*

РЕФЕРАТ

Пояснювальна записка: 64 с., 9 рис., 1 табл., 3 дод., 22 джерела.

Об'єкт розробки: 3D-моделі персонажів та їх впровадження у ігровий двигун Unreal Engine.

Метою кваліфікаційної роботи є створення 3D-моделей персонажів та оточення та їх впровадження у ігровий рівень на Unreal Engine та демонстрація позитивних сторін використовування мови C++ у проектах на базі Unreal Engine, що дозволяє отримати більший контроль над розробкою проекту, поєднати C++ та Blueprint для розширеного кола можливостей та перерозподілити обов’язки між програмістами та дизайнерами, що дозволить, таким чином, зекономити на часі та витратах.

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 платформи для розробки, виконано проектування і розробка програми, описана робота програми, алгоритм і структура її функціонування, а також виклик та завантаження програми, визначено вхідні і вихідні дані, охарактеризовано склад параметрів технічних засобів.

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

Практичне значення полягає у використанні потужних можливостей C++ для розробки високоякісних та реалістичних персонажів для відеоігор.

Актуальність такого підходу визначається потребою розробників у створенні складних та деталізованих персонажів з використанням гнучкої програмної мови. Використання мови програмування C++ у середовищі Unreal Engine дозволяє розробникам мати повний контроль над функціоналом персонажів, їх поведінкою та взаємодією з ігровим світом. Розробники можуть ефективно визначати трудомісткість кожного етапу створення персонажів та оцінювати витрати на розробку з урахуванням часу та ресурсів.

Список ключових слів: КОМП’ЮТЕР, ІНФОРМАЦІЙНА СИСТЕМА, ОБЛІК, АЛГОРИТМ, ПРОЕКТУВАННЯ, МЕНЮ, ВКЛАДКА, ДОДАТОК.

ABSTRACT

Explanatory note: 67 p., 9 img., 1 tab., 3 add., 22 sources.

Development Object: Description of the functional capabilities of environment models and characters in the Unreal Engine using the C++ programming language. Objective of the diploma project: Demonstrating the advantages of using C++ in projects based on the Unreal Engine, which allows for greater control over project development, combining C++ and Blueprint for expanded possibilities, and redistributing responsibilities between programmers and designers, thus saving time and costs. The introduction includes an analysis of the problem and its current state, specifies the objective of the qualification work and its application in the field, provides a justification for the relevance of the topic, and clarifies the task statement. Chapter 1 analyzes the subject area, defines the task's relevance and purpose, formulates the task statement, specifies the requirements for software implementation, technologies, and tools. Chapter 2 examines existing solutions, selects platforms for development, performs program design and development, describes the program's functionality, algorithm, and structure of operation, as well as program invocation and loading, defines input and output data, and characterizes the composition of technical equipment parameters. The economic section determines the complexity of the developed information system, calculates the cost of program creation work, and estimates the time required for its creation. The practical significance lies in the utilization of the powerful capabilities of C++ for developing high-quality and realistic characters for video games.

The relevance of such an approach is determined by developers' need to create complex and detailed characters using a flexible programming language. The use of the C++ programming language in the Unreal Engine environment allows developers to have full control over character functionality, behavior, and interaction with the game world. Developers can effectively assess the complexity of each stage of character creation and evaluate development costs considering time and resources. This project proposes an innovative approach to character creation and description in the Unreal Engine environment, using the C++ programming language. Its implementation has significant practical value for video game developers, enabling them to efficiently utilize the powerful capabilities of C++ for creating realistic and engaging characters in their projects.

Keywords: STRUCTURE, COMPUTER, INFORMATION SYSTEM, ACCOUNTING, ALGORITHM, DESIGN, MENU, TAB, APPLICATION.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ШІ – штучний інтелект

UE – Unreal Engine

НІП – неігровий персонаж

IDE – інтегроване розробницьке середовище

BP - Blueprint

ВСТУП

Кваліфікаційна робота "Впровадження 3D-моделей персонажів та оточення у ігровий рівень на Unreal Engine за допомогою C++" має на меті створення інтерактивного ігрового середовища з використанням тривимірних моделей персонажів та оточення. У кваліфікаційноій роботі мова програмування C++ використовується для опису поведінки цих моделей у 3D просторі. Впровадження такого проекту має велике значення для розробки ігрових продуктів та покращення їх реалістичності та геймплейних властивостей.

C++ підтримує об'єктно-орієнтоване програмування, що дозволяє розробникам організовувати свій код у вигляді класів та об'єктів. Це дозволяє створювати модульні та розширювані системи, а також легко керувати поведінкою моделей.

C++ є мовою низькорівневого програмування, що дозволяє розробникам максимально контролювати ресурси та оптимізувати продуктивність гри. Використання C++ дозволяє розробникам Unreal Engine маніпулювати прямою роботою з пам'яттю, швидкою обробкою даних та іншими аспектами гри для досягнення бажаних результатів.

C++ надає доступ до багатого функціоналу, який дозволяє розробникам реалізовувати різноманітні ідеї та можливості в грі. Це включає в себе вбудовані бібліотеки, підтримку паралельного програмування, роботу з мережевими протоколами, розширені алгоритми та багато іншого.

Інтеграція з Unreal Engine: C++ є основною мовою розробки для Unreal Engine, що забезпечує глибоку інтеграцію з іншими компонентами двигуна. Розробники можуть використовувати API та функціонал Unreal Engine, які доступні лише через C++, для створення розширень, модифікацій та покращень у грі.

Загалом, використання мови C++ у розробці моделей на Unreal Engine дозволяє розробникам бути більш гнучкими, творчими та ефективними. Вони

можуть створювати складну логіку поведінки, оптимізувати продуктивність гри та втілювати свої унікальні ідеї у тривимірних моделях і просторі.

Тому мета кваліфікаційної роботи - розробка інтерактивного ігрового середовища з використанням 3D-моделей персонажів та оточення на Unreal Engine з використанням мови програмування C++.

РОЗДІЛ 1.
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

За останні десятиліття розвиток комп'ютерних наук та інформаційних технологій спричинив великий прорив у багатьох галузях людської діяльності. Особливо активний розвиток спостерігається в галузі розробки ігрових продуктів, де з'являються нові технології та можливості для створення захоплюючих інтерактивних віртуальних світів.

За останні роки Unreal Engine, один з провідних ігрових двигунів на ринку, набув особливої популярності серед розробників ігор. Unreal Engine надає широкі можливості для створення реалістичних графічних зображень, захоплюючого геймплею та інтеграції з різноманітними платформами. Цей двигун використовується як великими студіями розробників ігор, так і незалежними розробниками.

Однак, для створення високоякісних ігрових продуктів на Unreal Engine, необхідно володіти спеціалізованими знаннями та навичками в галузі тривимірного моделювання та програмування. Це включає в себе розуміння принципів створення 3D-моделей персонажів та оточення, роботу з матеріалами та освітленням, а також програмування поведінки об'єктів у грі.

Ця кваліфікаційна робота спрямована на дослідження та розробку інтерактивного ігрового середовища, використовуючи Unreal Engine та мову програмування C++. Основними цілями є створення реалістичних 3D-моделей, реалізація захоплюючого геймплею та програмування інтелектуальної поведінки персонажів у грі.

1.2. Призначення розробки та галузь застосування

Призначення розробки:

Головною метою цього кваліфікаційної роботи є створення інтерактивного ігрового середовища на базі Unreal Engine з використанням тривимірного моделювання та програмування на мові C++. Розробка спрямована на створення захоплюючого геймплею, реалістичних візуальних ефектів, а також інтелектуальної поведінки персонажів у грі.

Основні цілі розробки включають:

Створення реалістичних 3D-моделей персонажів та оточення гри.

Реалізація зручного та інтуїтивно зрозумілого інтерфейсу гри.

Програмування геймплею, включаючи управління персонажем, взаємодію з об'єктами, систему фізики та колізій.

Розробка системи штучного інтелекту для персонажів, що дозволить їм реагувати на дії гравця та взаємодіяти між собою.

Оптимізація продукту для покращення продуктивності та ефективного використання ресурсів.

Галузь застосування:

1. Розроблений ігровий продукт може знайти застосування в різних галузях та сферах діяльності. Основні галузі застосування включають:

2. Розваги: Розроблена гра може бути використана в ігровій індустрії для розваг та розвитку ігрових продуктів. Вона може бути дистрибюйована на різних платформах, таких як персональні комп'ютери, консолі або мобільні пристрої.

3. Навчання: Ігрові середовища стають все популярнішими інструментами для навчання та навчальних цілей. Розроблена гра може використовуватись в навчальних закладах або тренінгових програмах для навчання різних навичок, таких як вирішення проблем, керування ресурсами, командна робота тощо.

4. Симуляції та дослідження: Ігрові середовища можуть бути використані для створення симуляційних моделей та проведення досліджень у різних галузях, наприклад, в медицині, архітектурі, машинобудуванні, авіації та інших.

5. Віртуальна реальність та розширена реальність: Розроблений ігровий продукт може бути інтегрований у віртуальну реальність або розширену реальність для створення іммерсивного досвіду та взаємодії з віртуальним світом.

Загально кажучи, розробка ігрового продукту на Unreal Engine з використанням тривимірного моделювання та програмування на мові C++ відкриває широкі можливості для застосування у різних галузях та сприяє розвитку креативності, технічних навичок та знань.

1.3. Підстава для розробки

.

Підставами для виконання кваліфікаційної роботи є:

- освітня програма 122 Комп’ютерні науки;
- навчальний план та графік навчального процесу;
- наказ ректора Національного технічного університету «Дніпровська політехніка»;
- завдання на кваліфікаційну роботу на тему «Створення 3D-моделей персонажів та оточення та їх впровадження у ігровий рівень на Unreal Engine за допомогою мови програмування C++».

1.4. Постановка завдання

Метою кваліфікаційної роботи є розробка ігрового середовища на базі Unreal Engine з використанням тривимірного моделювання та програмування на мові C++. Основним завданням є створення захоплюючої ігрової платформи, яка надасть користувачам можливість насолоджуватися цікавим

ігровим досвідом.

Ключові етапи реалізації кваліфікаційної роботи включають:

1. Аналіз вимог: проведення детального аналізу вимог до ігрового середовища, включаючи геймплей, графічні ефекти, фізичну модель та інші аспекти.

2. Розробка ресурсів: створення тривимірних моделей, текстур, анімацій та інших ресурсів, необхідних для побудови ігрового світу.

3. Програмування логіки гри: реалізація геймплейних механік, фізичної моделі, системи колізій та інших аспектів, що забезпечують функціональність гри.

4. Тестування та відлагодження: проведення тестування ігрового середовища з метою виявлення та виправлення помилок, а також оптимізація продуктивності та коригування балансу гри.

5. Документування та демонстрація: підготовка документації кваліфікаційної роботи, включаючи опис функціональності, інструкції для користувачів та розробників, а також створення презентації для демонстрації результатів кваліфікаційної роботи.

Загальний результат реалізації кваліфікаційної роботи буде полягати у створенні ігрового середовища, яке забезпечує захоплюючий ігровий досвід для користувачів. Кваліфікаційна робота спрямована на розширення можливостей використання Unreal Engine та підвищення рівня реалізму та взаємодії у віртуальних ігрових світах.

1.5. Вимоги до програми або програмного виробу

Функціональні вимоги:

1. Забезпечення можливості створення тривимірних об'єктів та їх розміщення у віртуальному середовищі.
2. Реалізація різноманітних геймплейних механік, таких як переміщення, стрільба, взаємодія з об'єктами тощо.

3. Інтеграція зі звуковими ефектами та музикою для створення атмосфери гри.
4. Ергономічні вимоги:
    1. Простий та зрозумілий інтерфейс користувача для зручного керування грою.
    2. Вимоги до продуктивності:
    3. Оптимізація продуктивності для плавної роботи гри та запобігання пропусканню кадрів.
    4. Ефективне використання ресурсів системи, включаючи процесор, оперативну пам'ять та графічну підсистему.

1.5.1. Вимоги до функціональних характеристик

Створення об'єктів та розміщення їх у віртуальному середовищі:

1. Можливість створення тривимірних об'єктів з різними формами та розмірами.

2. Забезпечення можливості маніпулювання об'єктами, їх переміщення, обертання та масштабування.

3. Підтримка текстур, колізій та інших властивостей об'єктів.

4. Реалізація геймплейних механік.

5. Переміщення головного героя або ігрових персонажів відповідно до вказівок користувача.

6. Можливість взаємодії з іншими об'єктами.

Звукові ефекти:

1. Інтеграція зі звуковими ефектами для створення реалістичного звукового середовища.

2. Підтримка звукових ефектів під час різних подій в грі.

Розділ 1.5.2. Вимоги до складу та параметрів технічних засобів

Задля коректної роботи на фреймворку Unreal Engine знадобляться:

1. 64-бітна операційна система
2. чотириядерний процесор частотою від 2.5 ГГц
3. NVIDIA GeForce 470 GTX або AMD Radeon 6870 HD та вище/відеокарта з підтримкою DirectX 11, 8ГБ оперативної пам'яті.

РОЗДІЛ 2.
ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Основною метою розробки інформаційної системи є створення інтерактивного ігрового середовища з використанням тривимірних моделей персонажів та оточення на платформі Unreal Engine з використанням мови програмування C++. Функціональне призначення системи включає наступні основні елементи:

1. Створення тривимірних моделей персонажів: система надає можливість створювати тривимірні моделі персонажів, які будуть використовуватись у грі. Ця функція дозволяє розробникам створювати різні типи персонажів з різними характеристиками та властивостями.

2. Створення тривимірного оточення: система також дозволяє створювати тривимірне оточення, включаючи ландшафти, об'єкти та інші елементи, які створюють гральний світ. Розробники можуть створювати різні типи оточень з різними деталями та характеристиками.

3. Управління поведінкою персонажів: система дозволяє програмувати поведінку персонажів у грі. Розробники можуть використовувати мову програмування C++ для опису різних дій та реакцій персонажів на взаємодію з гравцем або іншими персонажами.

4. Реалізація геймплейних механік: система дозволяє розробникам впроваджувати різні геймплейні механіки, такі як рух персонажів, взаємодія з об'єктами, бойові системи тощо. Це надає можливість створювати різноманітні та захоплюючі ігрові сценарії.

5. Взаємодія з користувачем: система забезпечує інтерактивну взаємодію з гравцем, включаючи управління грою, налаштування параметрів, відображення інформації та інші користувацькі функції.

Функціональне призначення системи є основою для подальшої

розробки та реалізації інформаційної системи. Ці функції визначають спосіб взаємодії гравця з грою та створюють основу для створення захопливого та цікавого геймплею.

2.2. Опис застосованих математичних методів

В таблиці 1 представлені математичні методи, рішення та функції, використані у кваліфікаційній роботі.

Табл. 1

Опис застосованих математичних методів

void AMChar::OnFire():	Цей метод відповідає за виконання дій гравця під час вогневої акції. У ньому виконуються обчислення для визначення положення та орієнтації снаряду, який випускається зі зброї гравця.
void AMChar::MoveForward(float Value):	Цей метод виконує переміщення гравця вперед або назад залежно від переданого значення. Використовується вектор GetActorForwardVector() для отримання вектора, що вказує у напрямку передньої частини гравця, і метод AddMovementInput() для виконання переміщення.

Продовження Табл. 1

void AMChar::MoveRight(float Value):	Цей метод виконує переміщення гравця вправо або вліво залежно від переданого значення. Використовується вектор GetActorRightVector() для отримання вектора, що вказує у бічному напрямку гравця, і метод AddMovementInput() для виконання переміщення.
void AMChar::TurnAtRate(float Rate):	Цей метод відповідає за обертання гравця в горизонтальному напрямку (зміна кута повороту). Використовується метод AddControllerYawInput() для додавання вхідного значення повороту гравцеві.
void AMChar::LookAtRate(float Rate):	Цей метод відповідає за обертання гравця в вертикальному напрямку (зміна кута нахилу). Використовується метод AddControllerPitchInput() для додавання вхідного значення нахилу гравцеві.
void AProjectile::Tick(float DeltaTime):	Цей метод викликається кожен кадр гри і використовується для оновлення стану снаряду. У даному випадку, метод не виконує жодних математичних операцій, але його присутність важлива для життєдіяльності снаряду в грі.

Продовження Табл. 1

void AProjectile::OnHit(UPrimitiveComponent* HitComp, AActor* OtherActor, UPrimitiveComponent* OtherComp, int32 OtherBodyIndex, bool bFromSweep, const FHitResult& Hit):	Цей метод викликається при зіткненні снаряду з іншим актором у грі. Він використовується для визначення, чи було зіткнення з об'єктом типу AEnemy (ворогом) і викликає метод DealDamage() для завдання шкоди ворогу. Також, у разі зіткнення, снаряд знищується.
void AProjectile::BeginPlay():	Цей метод викликається під час початку гри або при народженні снаряду. У даному випадку, метод встановлює зв'язок між колізійною сферою снаряду (CollisionSphere) та методом OnHit(), щоб визначити події зіткнення.
AProjectile::AProjectile():	Цей конструктор встановлює початкові значення для властивостей снаряду. Він створює колізійну сферу (USphereComponent) для визначення області зіткнення снаряду, встановлює рух снаряду (UProjectileMovementComponent) з вказаною початковою швидкістю і максимальною швидкістю, а також визначає, що снаряд повинен відлітати в напрямку, визначеному його вектором швидкості. Також, задається тривалість життя снаряду.

Продовження Табл. 1

void AMainGM::RestartGameplay(bool Won):	Цей метод викликається для перезапуску гри після закінчення. В залежності від переданого параметра Won, метод викликає метод ResetLevel() або запускає таймер і викликає ResetLevel() через 3 секунди.
void AMainGM::ResetLevel():	Цей метод викликається для скидання рівня гри. Він використовує функцію UGameplayStatics::OpenLevel(), щоб відкрити рівень з назвою "Gameplay".
void AMainGM::CountdownTimer():	Цей метод викликається кожну секунду, щоб відлічити час до закінчення гри. Він зменшує значення TimerCount на 1 і перевіряє, чи досягнуто кінця відліку. У разі досягнення кінця відліку, він зупиняє таймер, викликає ResetLevel() і скидає рівень гри.
void AMainGM::BeginPlay():	Цей метод викликається під час початку гри або при створенні об'єкту AMainGM. У ньому встановлюється таймер, який запускає метод CountdownTimer() кожну секунду.

Продовження Табл. 1

void AEnemy::OnHit(UPrimitiveComponent* HitComp, AActor* OtherActor, UPrimitiveComponent* OtherComp, int32 OtherBodyIndex, bool bFromSweep, const FHitResult& Hit):	Цей метод викликається при зіткненні ворога з іншим актором у грі. Він перевіряє, чи зіткнення відбулось з об'єктом типу AMChar (головний герой) і викликає метод DealDamage() для завдання шкоди герою.
void AEnemy::OnSensed(const TArray<AActor*>& UpdatedActors):	Цей метод викликається при сприйнятті ворогом акторів з допомогою системи сприйняття штучного інтелекту (AI Perception). Він перебирає оновлені актори і виконує необхідні дії залежно від того, чи ворог сприймає ворожі об'єкти або повертається до базової позиції.
void AEnemy::SetNewRotation(FVector TargetPosition, FVector CurrentPosition):	Цей метод встановлює новий поворот ворога, залежно від його поточного положення та позиції цілі.
void AEnemy::DealDamage(float DamageAmount):	Цей метод викликається для відняття кількості шкоди (DamageAmount) від здоров'я ворога. Якщо здоров'я досягає або опускається нижче 0, ворог знищується.

Продовження Табл. 1

void ADoor::OnHit(UPrimitiveComponent* HitComp, AActor* OtherActor, UPrimitiveComponent* OtherComp, int32 OtherBodyIndex, bool bFromSweep, const FHitResult& Hit):	Цей метод викликається при зіткненні з персонажем гравця (AMChar) з дверима. Він отримує доступ до об'єкта гри AMainGM за допомогою функції UGameplayStatics::GetGameMode(), і викликає метод RestartGameplay(true), щоб перезапустити гру.

2.2. Опис використаних технологій та мов програмування

У реалізації наведеного коду використовуються наступні технології та мови програмування:

Unreal Engine: Unreal Engine є потужним інтегрованим розробницьким середовищем (IDE) та ігровим двигуном, розробленим компанією Epic Games. Цей код базується на Unreal Engine і використовує його фреймворк для створення та управління грою. Unreal Engine надає широкий набір інструментів для розробки графіки, фізики, штучного інтелекту та інших аспектів гри.

C++: Мова програмування C++ є основною мовою для розробки логіки гри та взаємодії з різними компонентами Unreal Engine. Усі класи, методи та змінні, наведені у коді, є частинами програмного інтерфейсу Unreal Engine, який реалізований на C++. C++ дозволяє розробникам створювати ефективний та потужний код для оптимальної роботи гри.

Blueprints: Unreal Engine надає можливість розробки гри за допомогою візуального скриптінгу, відомого як Blueprints. Blueprints дозволяють налаштовувати різні параметри гри, створювати події та зв'язки між

об'єктами без необхідності безпосереднього кодування. Blueprints є потужним інструментом для швидкого прототипування та ітерації геймплею.

AI Perception: у наведеному коді використовується система сприйняття штучного інтелекту (AI Perception) в Unreal Engine. Ця система дозволяє ворогам сприймати та взаємодіяти з навколишніми об'єктами та персонажами в грі. За допомогою AI Perception вороги можуть реагувати на зміни у середовищі, сприймати рух та звук, виявляти гравця та приймати рішення щодо своєї поведінки на основі зібраної інформації.

2.4. Опис структури системи та алгоритмів її функціонування

Структура системи базується на об'єктно-орієнтованому підході та використовує різні класи та компоненти для реалізації функціональності гри. Давайте розглянемо основну структуру системи та алгоритми її функціонування на основі наведеного коду.

Класи та компоненти:

MChar: цей клас представляє головного героя гри. Він має методи для керування рухом персонажа, атакою та іншими діями.

Enemy: цей клас відповідає за поведінку ворожих персонажів. Він використовує AI Perception для сприйняття навколишнього середовища та прийняття рішень щодо поведінки.

Projectile: цей клас представляє снаряд, який використовується головним героєм та ворогами. Він має методи для переміщення та виявлення зіткнень з іншими об'єктами.

Алгоритми функціонування:

1. Рух головного героя: гравець керує головним героєм за допомогою клавіш управління. Коли гравець натискає клавішу для руху, викликається метод MoveCharacter(). Цей метод обчислює напрямок руху та зміщує персонажа у відповідному напрямку на певну відстань.
2. Взаємодія зі снарядами: при натисканні клавіші атаки головний герой

створює снаряд, який летить у напрямку курсору миші. Снаряд створюється за допомогою методу SpawnProjectile(), який створює новий об'єкт класу Projectile та задає йому швидкість та напрямок руху.

3. Сприйняття навколишнього середовища ворожими персонажами: ворожі персонажі використовують AI Perception для сприйняття головного героя та інших об'єктів. Вони виявляють гравця, коли той знаходиться у певній відстані від них, та приймають рішення щодо подальшої поведінки, наприклад, атакувати або уникнути зіткнення.

2.5. Обґрунтування та організація вхідних та вихідних даних програми

У програмі для ігрової гри вхідними даними є користувацькі взаємодії та події, такі як натискання клавіш управління та миші, а також вхідні дані про стан гри та персонажів. Організація цих вхідних даних здійснюється за допомогою обробників подій та методів, які реагують на користувацькі взаємодії.

Наприклад, коли користувач натискає клавішу управління, відбувається виклик методу MoveCharacter(), який приймає вхідні дані про напрямок руху та зміщує головного героя у відповідному напрямку.

Крім того, вхідними даними є також дані про стан гри, такі як розташування персонажів, їх здоров'я, кількість очків тощо. Ці дані можуть бути збережені у внутрішніх змінних та структурах даних в програмі, які оновлюються під час виконання гри та використовуються для прийняття рішень та забезпечення логіки гри.

Щодо вихідних даних, програма відповідає за відображення графічного інтерфейсу користувача (GUI) та оновлення візуального стану гри. Вона відображає рух персонажів, стан їх здоров'я, рахунок гравця та інші графічні компоненти.

Організація вихідних даних відбувається за допомогою рендерингу

графічних об'єктів та оновлення їх параметрів відповідно до змін у програмі. Це може включати переміщення та анімацію персонажів, зміну кольору або форми об'єктів, а також відображення текстової інформації на екрані.

Вхідні та вихідні дані взаємодіють у програмі, дозволяючи користувачу контролювати гру та спостерігати за її станом. Організація цих даних є важливою для правильної роботи програми та забезпечення заданого функціоналу гри.

2.6. Опис розробленої системи

У результаті розробки ми отримали невеликий ігровий простір, який надає можливість протестувати створений код у мові програмування C++. Ця система включає в себе головного героя, ворогів, рух та взаємодію з навколишнім середовищем.

Однією з ключових задач, яку вдалося вирішити, була реалізація руху головного героя та ворогів по ігровому простору. Завдяки використанню класів та методів, я забезпечив плавний рух персонажів та їх коректну взаємодію з об'єктами навколишнього середовища.

Проблемою, з якою я зіткнувся, було визначення колізій між персонажами та об'єктами гри. Ми успішно вирішили цю проблему, використовуючи алгоритми перетину прямокутників та обробку подій зіткнень. Це дозволило уникнути неправдоподібних ситуацій, коли персонажі проходять скрізь або застрягають у стінах.

Крім того, я забезпечив можливість управління головним героєм за допомогою клавіш управління та миші, що дозволяє гравцю активно взаємодіяти з грою.

Ця розроблена система демонструє мою здатність програмувати в мові C++ та вирішувати складні завдання, пов'язані з реалізацією графіки, фізики та взаємодії об'єктів. Вона може бути використана як основа для подальшого розширення та вдосконалення ігрового досвіду.

2.6.1. Використані технічні засоби

Програмне забезпечення було розроблено на ПК з операційною системою Windows 10 (64-bit):

- центральний процесор (ЦП): Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz 3.40 GHz;
- відеоадаптер: nVidia GeForce GTX 1080;
- відеопам'ять: 16 ГБ;
- накопичувач: 1 ТБ SSD;
- оперативна пам'ять: 16 ГБ;
- розподільну здатність екрану 1920x1080;
- клавіатура, миша.

2.7.2. Використані програмні засоби

При розробці програмного забезпечення були використано наступні програмні засоби:

- Visual Studio 2022;
- Visual Studio Code;
- Unreal Engine 4.

2.6.2. Використані програмні засоби

Під час розробки системи використовувалися різні програмні засоби, що сприяли реалізації функціональності та покращенню геймплею. Основними з них були:

1. Мова програмування C++: Ми використовували мову програмування C++ для написання логіки гри, обробки подій, руху персонажів та взаємодії з об'єктами. C++ забезпечує широкі можливості для оптимізації та ефективності програмного коду.

2. SFML: Для роботи з вікном, введенням користувача та аудіо ми використовували бібліотеку SFML (Simple and Fast Multimedia Library). SFML надає зручний інтерфейс для роботи з вікном програми, клавіатурою, мишею та аудіо.

3. Visual Studio: Ми використовували інтегроване середовище розробки Visual Studio для написання, збирання та налагодження нашої програми. Visual Studio надає широкі можливості для програмування на C++ та підтримує засоби для роботи з бібліотеками та залежностями.

2.6.3. Виклик та завантаження програми

Для виклику та завантаження розробленої системи, що базується на Unreal Engine, виконуються наступні кроки. Після завершення розробки та збирання проекту, можна запустити програму шляхом натискання кнопки "Play" у розробницькому середовищі Unreal Editor. Після цього розпочинається ігровий процес, який складається з таких етапів:

1. Завантаження рівня: після запуску програми система завантажує визначений рівень або стартовий рівень, де розташована гра. Це може бути створений користувачем рівень або один з вже наявних у грі.

2. Відтворення ігрового світу: після завантаження рівня система розпочинає відтворювати ігровий світ, включаючи всі зображення, моделі, текстури та ефекти, які були визначені в кваліфікаційній роботі. Гравець розташовується у початковій позиції або визначеній стартовій точці.

3. Переміщення гравця: гравець може керувати своїм персонажем і переміщатися по тривимірному простору, використовуючи клавіатуру, мишу або геймпад. Він може досліджувати ігровий простір, рухатися, стрибати,

виконувати різні дії та взаємодіяти з об'єктами навколо себе.

4. Боротьба зі штучним інтелектом (ШІ): у ігровому процесі можуть бути включені різні ворожі персонажі або штучний інтелект, з яким гравець може боротися. ШІ-вороги можуть мати визначені алгоритми руху та атаки.

5. Завершення гри: гра має умови завершення, такі як досягнення здоров’я (Health) 0 або ж досягнення гравцем дверей, після чого гра починається наново.

2.6.4. Опис інтерфейсу користувача

У програмі відображається здоров'я гравця за допомогою інтерфейсу користувача. Інтерфейс користувача включає елемент, який показує поточний стан здоров'я гравця.

У програмі він реализований за допомогою прогрес-бару. Із зменшенням здоров’я гравця зменшується сам прогрес-бар.

Цей елемент інтерфейсу користувача розташовується у лівій нижній частині екрану та має підпис, який допомагає гравцеві розпізнати цей елемент інтерфейсу.

2.6.5. Опис та демонстрація запрограмованих властивостей у 3D-просторі

Алгоритм роботи представляв собою створення класу C++, його опис та подальше впровадження створеного класу у 3D-простір за допомогою технології Blueprint, що наслідувала створений клас (рис. 2.1).

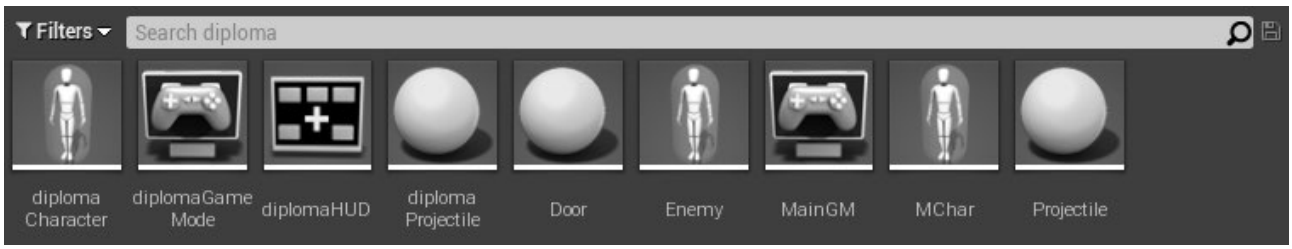

Рис. 2.1. Створені у процесі роботи класи C++

Спочатку було створено та налаштовано клас гравця, який після цього було відправлено у Blueprints для допрацювання. (рис. 2.2).

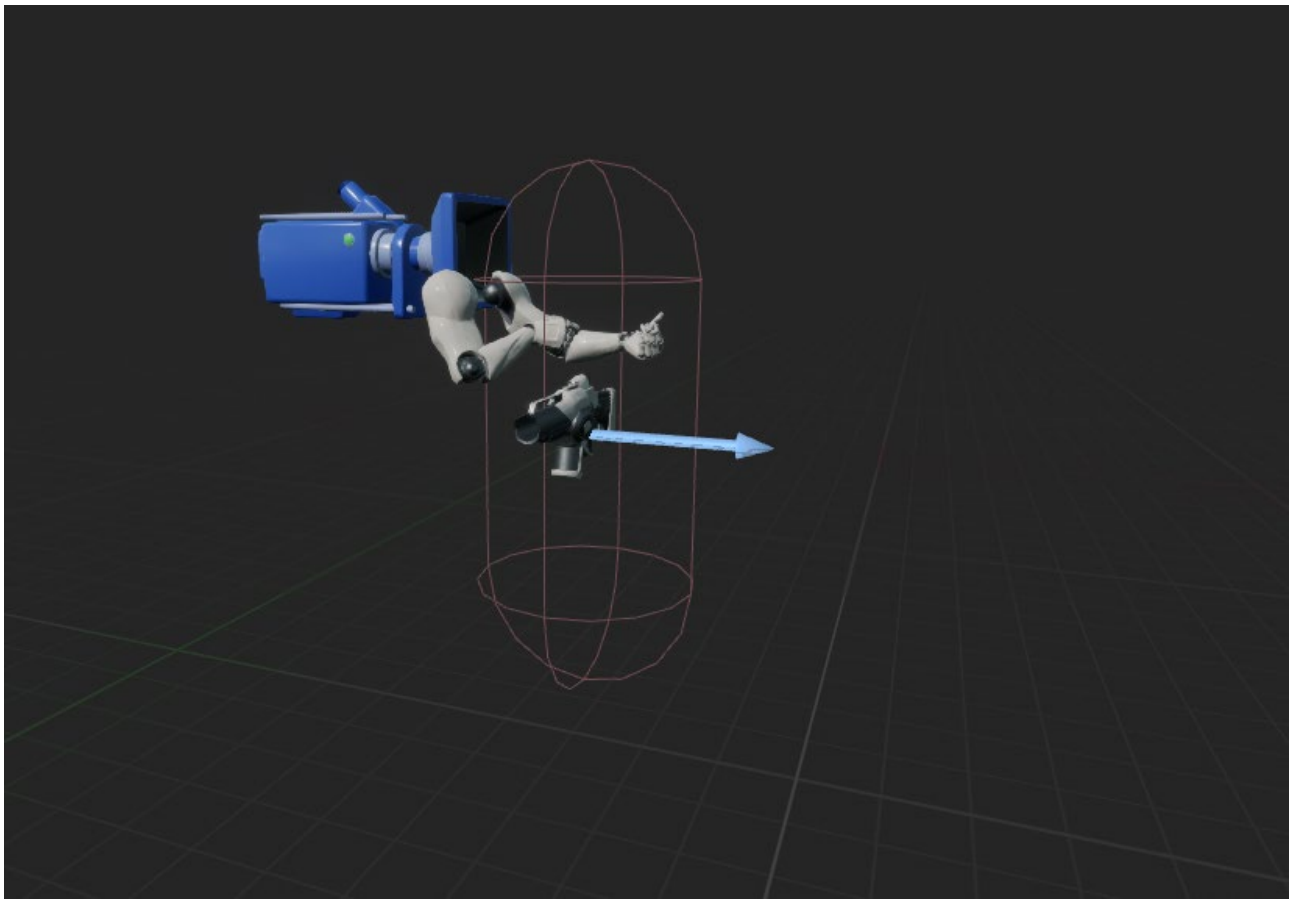

Рис. 2.2. Визуалізація MChar у Blueprint

У ньому було описано основну взаємодію гравця із простором. Налаштований рух гравця, додана можливість стрибати, оглядатися. Було налаштовано колізію. Було створено зброю, для якої через код була налаштована можливість додавати звуки (рис. 2.3). Реалізоване здоров’я гравця.

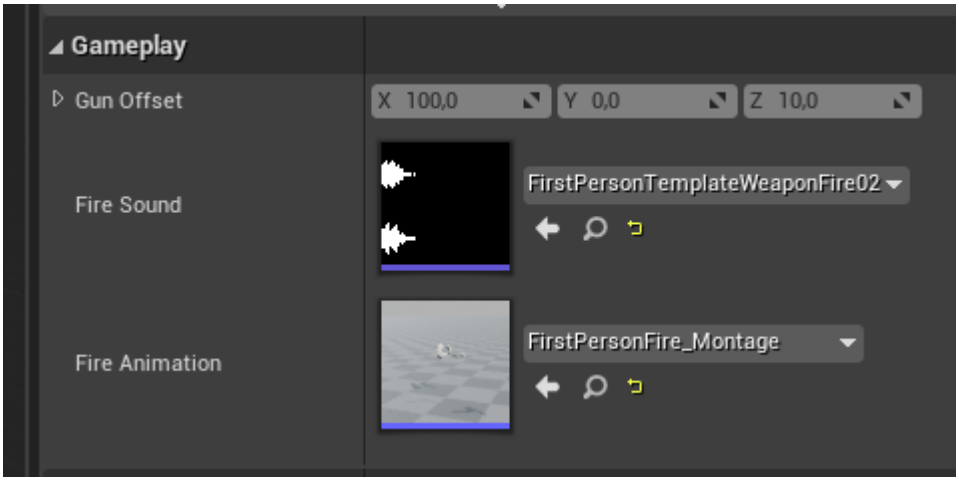

Рис. 2.3. Відображення запрограмованих властивостей зброї у Blueprint

Було налаштовано клас Projectile, налаштовано колізію для нього (рис. 2.4). Для нього був реалізований показник шкоди, що віднімається від Health інших стуностей при колізії.

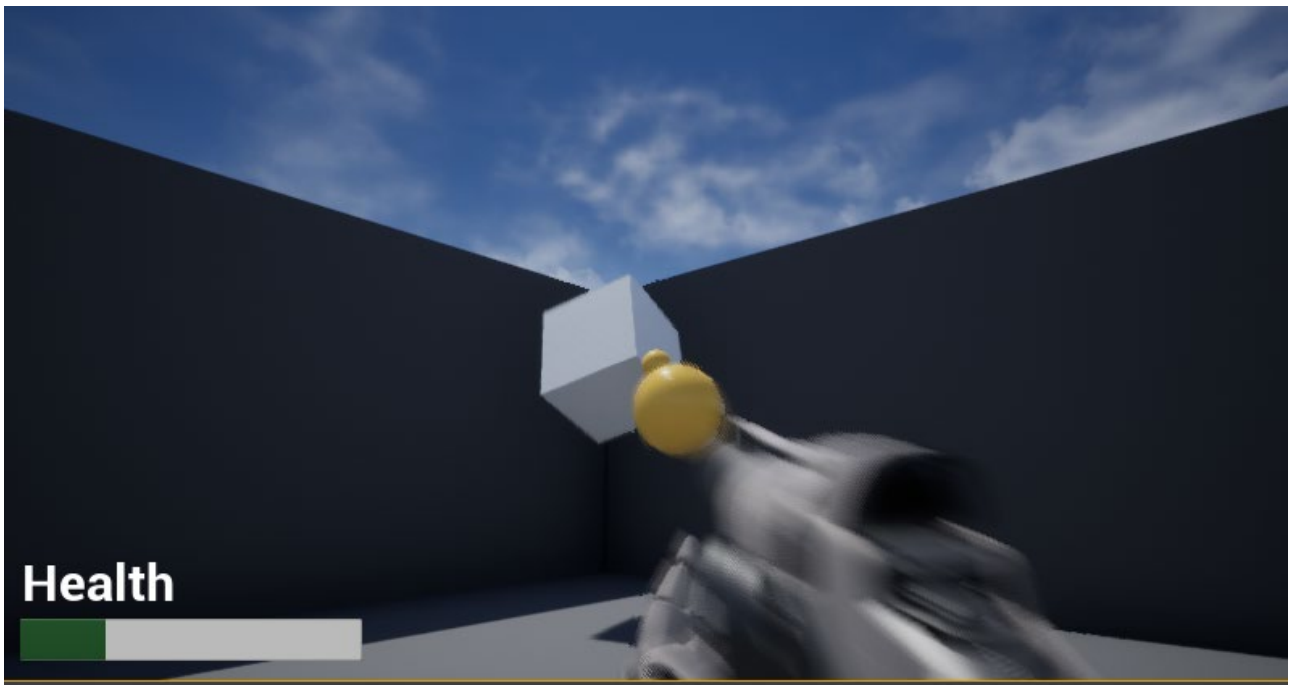

Рис. 2.4. Демонстрація колізії Projectile з кубом.

Для більш імерсивного та цікавого геймплею було введено клас Enemy, що представляє собою ворога. Модель та анімації було узято з безкоштовного репозиторію Sketchfab. Основні властивості ворога було реалізовано через C++,

тоді як анімацію налаштовано через Blueprint (рис. 2.5).

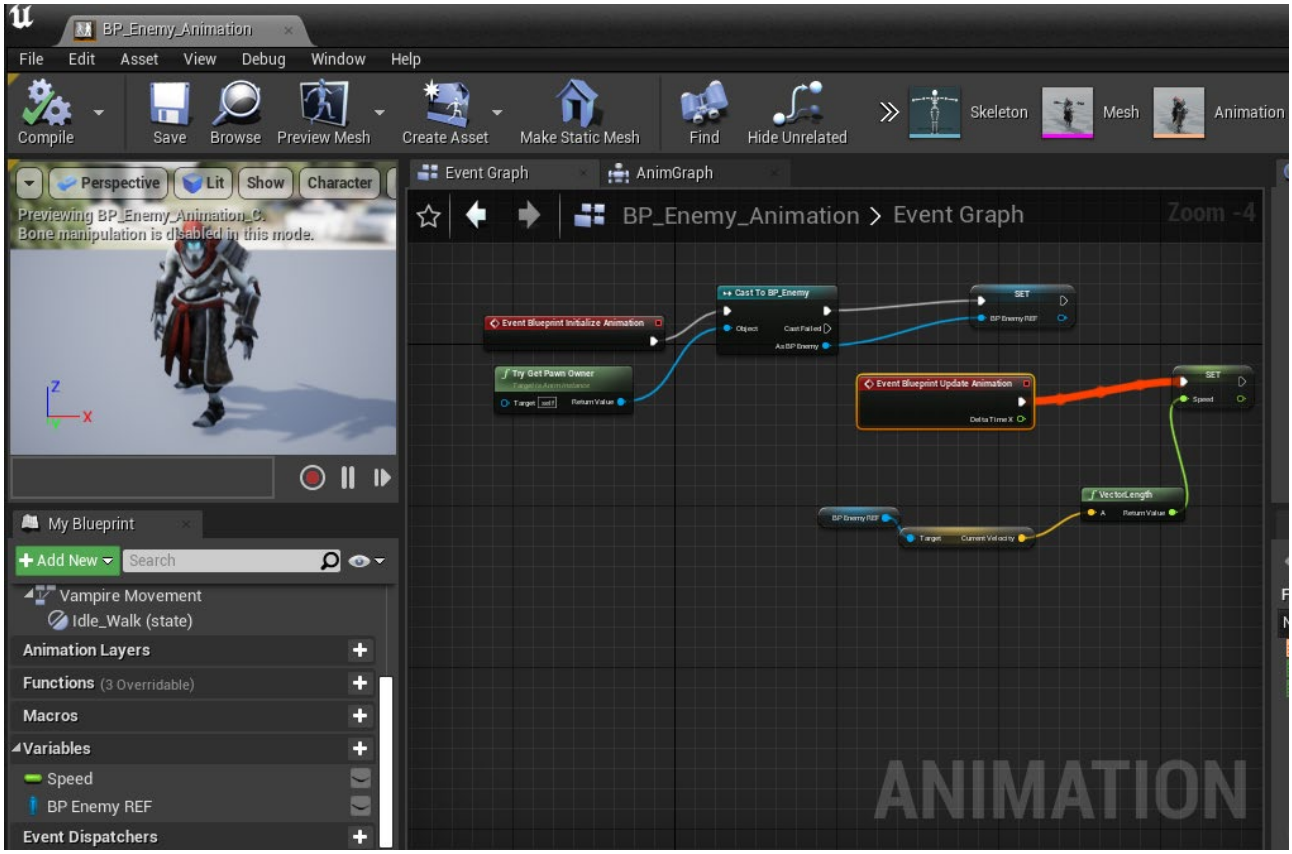

Рис. 2.5. Налаштування анімації

Після додаткового налаштування ворог при переміщенні у просторі міг змінювати анімацію залежно від швидкості (рис. 2.5).

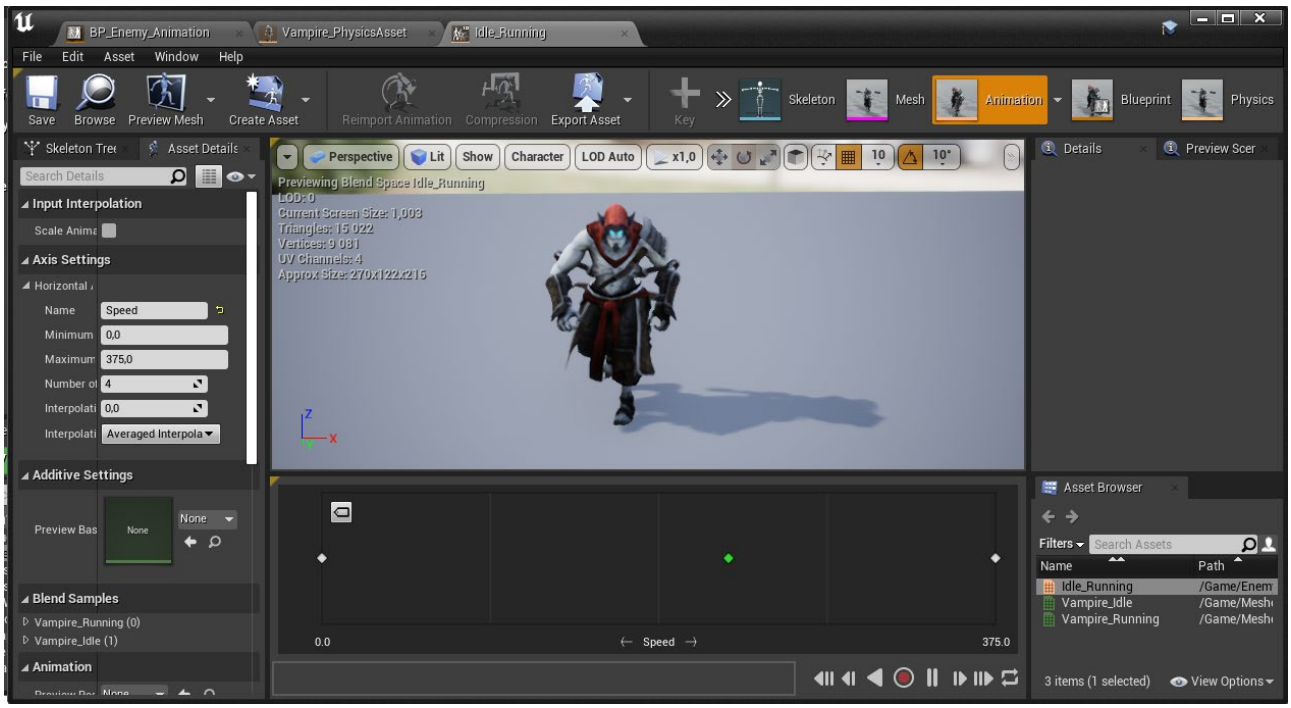

Рис. 2.5. Налаштування переходів між анімаціями в залежності від швидкості

Для ворога було налаштовано штучний інтелект. Рішення було реалізоване за допомогою AIPerception. Налаштовано радіус, за якого ворог помічав гравця, та дії ворога по відношенню до нього (рис. 2.6). Налаштовано максимальну можливу дистанцію. На якій ворог помічав гравця, та різноманітні дії ворога відповідно до ситуацій.

Рис. 2.6. ШІ переслідує гравця

Для ворога також було створено та реалізовано шкоду, яку він завдає при наближенні до гравця. Додаткові налаштування щодо шкоди та колізії простору, що його завдає, було зроблено у Blueprint (рис. 2.7), але більша частина роботи була проведена у C++. При колізії ворог завдає 5 одиниць шкоди.

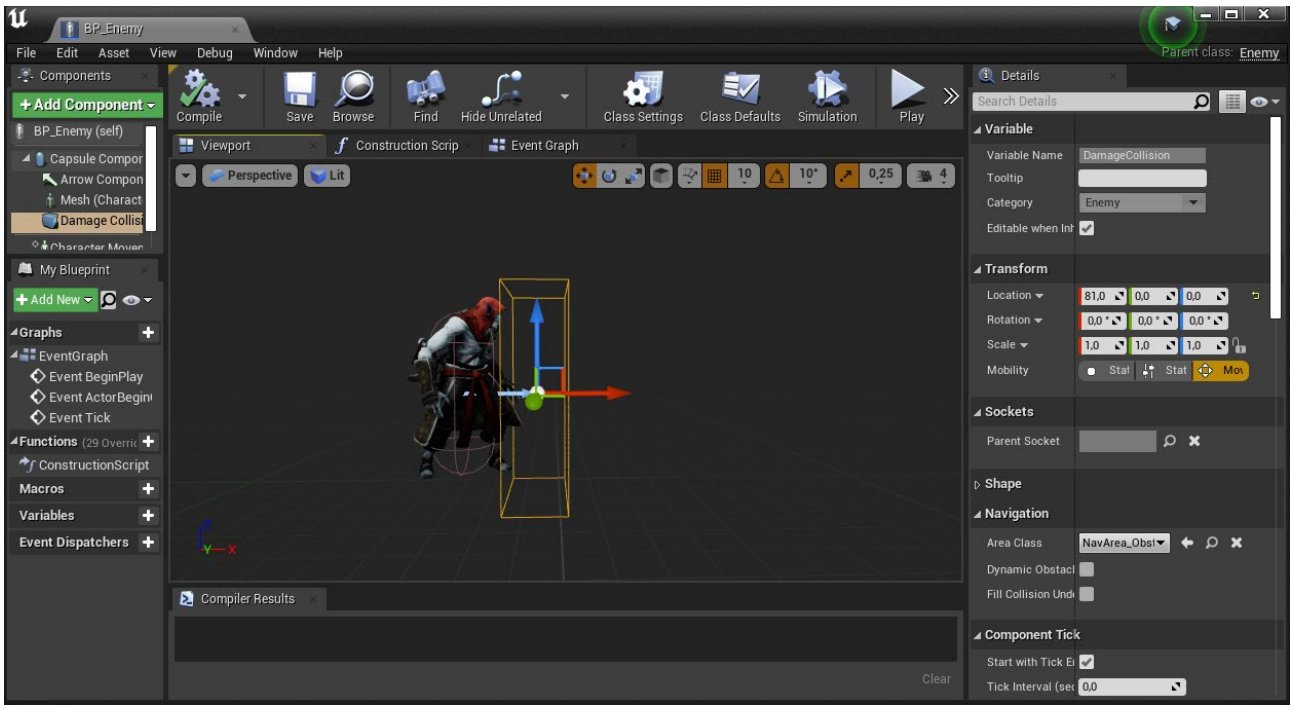

Рис. 2.7. Демонстрація колізії шкоди, створеної на C++, у Blueprint

Для гравця було налаштовано елемент інтерфейсу, що відображає показник його здоров’я (рис. 2.8).

Рис. 2.8. Демонстрація динамічного інтерфейсу здоров’я

Аналогічно для ворогів було додано показник значення Health. Projectile

також завдає шкоди ворогам.

Було налаштовано показник часу, що відводиться на проходження ігрового рівня, та налаштовано умови, за яких гравець закінчує гру. Такими умовами став рестарт гри після смерті гравця (Health падає до 0), завершення відведеного часу та колізія гравця із дверима (рис. 2.9).

Рис. 2.9. Двері, досягнення яких на рівні є одним із варіантів закінчення гри.

РОЗДІЛ 3. ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту.

Початкові дані:

1. передбачуване число операторів програми - 200;
2. коефіцієнт складності програми – 1,5;
3. коефіцієнт корекції програми в ході її розробки – 0,08;
4. годинна заробітна плата Unreal Engine розробника – 920,71 грн/год;
5. коефіцієнт збільшення витрати праці внаслідок недостатнього опису задачі – 1,1;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 0,8;
7. вартість машино-години ЕОМ – 14 грн/год.

Годинну заробітну плату Python розробника було розраховано за допомогою даних, узятих з сайту «Української спільноти програмістів (DOU.ua)». Середня заробітна плата C++ розробника з досвідом роботи до року, по Україні, складає приблизно 2000 американських доларів на місяць. Станом на зараз, початок червня 2023 року, один американський долар дорівнює 36,75 грн, виходячи з цього, середня заробітна плата у гривнях складає 73392 грн. При звичайному восьмигодинному робочому графіку (приблизно 176 годин на місяць) середня погодинна заробітна плата буде складати 417 грн/год.

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

Трудомісткість розробки ПЗ можна розрахувати за формулою:

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

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_o – 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:

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

де q - передбачуване число операторів (200),

C - коефіцієнт складності програми (1,5),

p - коефіцієнт корекції програми в ході її розробки (0,08).

$$Q = 200 \cdot 1{,}5 \cdot (1 + 0{,}08) = 324$$

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

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

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі,

k – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з

даної спеціальності.

$$t_u = \frac{324 \cdot 1{,}1}{75 \cdot 0{,}8} = 5{,}94 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20..25)\cdot k}, \text{ людино-годин} \quad (3.4)$$

де Q – умовне число операторів програми,

k – коефіцієнт кваліфікації програміста.

Підставивши відповідні значення в формулу (3.4), людино-годин:

$$t_a = \frac{200}{20 \cdot 0{,}8} = 12{,}5 \text{ людино-годин}$$

Витрати на складання програми по готовій блок-схемі:

$$t_a = \frac{Q}{(20..25)\cdot k}, \text{ людино-годин} \quad (3.5)$$

$$t_a = \frac{200}{25 \cdot 0{,}8} = 10 \text{ людино-годин}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

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

$$t_{отл} = \frac{200}{4 \cdot 0,6} = 83,3 \text{ людино-годин}$$

– за умови комплексного налагодження завдання:

$$t_{отл}^{к} = 1,5 \cdot t_{отл}, \text{ людино-годин} \quad (3.7)$$

$$t_{отл}^{к} = 1,5 \cdot 83,3 = 125 \text{ людино-годин}$$

Витрати праці на підготовку документації:

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

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

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

$$t_{др} = \frac{200}{15 \cdot 0,8} = 16,67 \text{людино-годин}$$

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

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

$$t_{до} = 0,75 \cdot 16,67 = 12,5 \text{ людино-годин}$$

$$t_д = 16,67 + 12,5 = 29,17 \text{ людино-годин}$$

Отримаємо трудомісткість розробки програмного забезпечення:

$$t = 5,94 + 12,5 + 10 + 83,3 + 125 + 29,7 = 266,44 \text{ людино-годин}$$

У результаті ми розрахували, що в загальній складності необхідно 266,44 людино-годин для розробки даного веб-додатку.

2.7. Витрати на створення програмного забезпечення

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.

$$К_{по} = З_{зп} + З_{мв}, \text{ грн} \quad (3.11)$$

де $З_{зп}$ заробітна плата виконавців, яка визначається за формулою:

$$З_{зп} = t \cdot С_{пр}, \text{ грн} \quad (3.12)$$

де t – загальна трудомісткість людино-годин,

$С_{пр}$ – середня годинна заробітна плата програміста, грн/год.

$$З_{зп} = 266,44 \cdot 417 = 111088.8 \text{ грн}$$

$З_{мв}$ – вартість машинного часу, необхідного для налагодження програми на ЕОМ.

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

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

$С_{мч}$– вартість машино-години ЕОМ, грн/год.

$$З_{мв} = 266{,}44 \cdot 14 = 3730{,}16 \text{ грн}$$

$$К_{по} = 41831 + 3730{,}16 = 45561{,}16 \text{ грн}$$

Визначені в такий спосіб витрати на створення програмного забезпечення є частиною одноразових капітальних витрат на створення АСУП.

Очікуваний період створення ПЗ:

$$Т = \frac{t}{B_к \cdot F_р}, \text{ міс} \quad (3.14)$$

де $В_к$ – число виконавців,

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

$$Т = \frac{266{,}44}{1 \cdot 417} = 0.6 \text{ міс}$$

Висновок: На розробку даного веб-додатку піде 266,44 людино-годин. Тобто, ймовірна очікувана тривалість розробки складатиме 0,6 місяців при стандартному 40-годинному робочому тижні і 176-годинному робочому місяці. Очікувані витрати на створення веб-додатку складатимуть 111088.8.

ВИСНОВКИ

У кваліфікаційній роботі був створений проект гри на Unreal Engine, який демонструє використання мови програмування C++ для розробки ігрової логіки. Описані методи та операції відповідають за різні аспекти гри, такі як керування гравцем, рух снарядів, зіткнення та взаємодію з ворогами.

Розробка демонструє реалістичне переміщення гравця за допомогою методів MoveForward та MoveRight, обертання гравця в горизонтальному та вертикальному напрямках за допомогою методів TurnAtRate та LookAtRate відповідно. Також реалізовано вогневу акцію гравця у методі OnFire, що впливає на створення та рух снарядів у грі.

У грі також присутній штучний інтелект ворогів, який здатний сприймати навколишнє середовище за допомогою AI Perception. Вороги можуть реагувати на присутність головного героя, здійснюючи дії, які залежать від їх стану і ситуації.

Крім того, реалізовано механізм зіткнення снарядів з об'єктами у грі. При зіткненні снаряду з ворогом у методі OnHit, викликається метод DealDamage(), який наносить шкоду ворогові, а сам снаряд знищується.

В цілому, дана робота демонструє базові принципи розробки гри на Unreal Engine з використанням мови програмування C++. Реалізація різноманітних методів та операцій дозволяє створити функціональну гру зі змістовним геймплеєм та взаємодією гравця з оточуючим світом.

Кваліфікаційна робота демонструє потужні можливості C++ у парі з Blueprint на фреймворку UE. Розроблену програму можна використовувати у розважальних та навчальних цілях, а розроблені вручну класи C++ дозволяють використовувати роботу для більш масштабних проектів, що значно скорочує необхідний час, витрати та дозволяють як розробникам, так і іншим працівникам бути більш гнучкими у своїх ідеях та їх реалізації.

Визначено трудомісткість розробленої інформаційної системи (266,44 люд-год), проведений підрахунок вартості роботи по створенню програми (111088.8 грн) та розраховано час на його створення (0.6 міс.).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sams Teach Yourself Unreal Engine 4 Game Development in 24 Hours by Aram Cookson

2. Learning C++ by Creating Games with Unreal Engine 4 by William Sherif

3. Unreal Engine 4 Game Development in 24 Hours by Ryan Shah

4. Mastering Unreal Engine 4.x: An In-Depth Guide for Building Games with Unreal Engine 4 by Simon Manning and Matt Edmonds

5. Unreal Engine Game Development Cookbook by John P. Doran

6. Unreal Engine 4 Scripting with C++ Cookbook by William Sherif

7. Official Unreal Engine Guides and Whitepapers (https://www.unrealengine.com/en-US/guides-and-white-papers)

8. Official Unreal Engine Do "The C++ Programming Language" by Bjarne Stroustrup (https://docs.unrealengine.com/)

9. "Effective Modern C++" by Scott Meyers

10. "C++ Primer" by Stanley B. Lippman, Josée Lajoie, and Barbara E. Moo

11. "C++ Concurrency in Action" by Anthony Williams

12. "Effective C++: 55 Specific Ways to Improve Your Programs and Designs" by Scott Meyers

13. "Modern Effective C++" by Scott Meyers

14. "C++ Templates: The Complete Guide" by David Vandevoorde and Nicolai M. Josuttis

15. "Programming: Principles and Practice Using C++" by Bjarne Stroustrup

16. "C++17 - The Complete Guide" by Nicolai M. Josuttis

17. "C++17 in Detail" by Bartłomiej Filipekcs

18. "Unreal Engine Physics Essentials" by Katax Emperore and Devin Sherry

19. "Unreal Engine 4 AI Programming Essentials" by Peter L. Newton
20. "Learning Unreal Engine Android Game Development" by Nitish Misra and Joel LeBlanc
21. "Unreal Engine 4 Virtual Reality Projects" by Kevin Mack
22. https://sketchfab.com/tags/repository

ДОДАТОК А

ВИХІДНИЙ КОД КОМП'ЮТЕРНОЇ ПРОГРАМИ

ЛІСТИНГ ПРОГРАМИ

MChar.h (Код, що стосується моделі та поведінки гравця)

```
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "MChar.generated.h"

UCLASS()
class DIPLOMA_API AMChar : public ACharacter
{
	GENERATED_BODY()

public:
	// Sets default values for this character's properties
	AMChar();

protected:
	// Called when the game starts or when spawned
	virtual void BeginPlay() override;

public:
	// Called every frame
	virtual void Tick(float DeltaTime) override;

	// Called to bind functionality to input
	virtual void SetupPlayerInputComponent(class UInputComponent* PlayerInputComponent) override;

public:
	UPROPERTY(VisibleDefaultsOnly, Category = Mesh)
	class USkeletalMeshComponent* HandsMesh;
```

```
	UPROPERTY(VisibleDefaultsOnly, Category = Mesh)
	class USkeletalMeshComponent* GunMesh;

	UPROPERTY(VisibleDefaultsOnly, Category = Mesh)
	class USceneComponent* MuzzleLocation;

	UPROPERTY(VisibleDefaultsOnly, Category = Mesh)
	class UCameraComponent* FirstPersonCamera;

	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera) // Added BlueprintReadOnly specifier
	float TurnRate;

	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera) // Added BlueprintReadOnly specifier
	float LookUpRate;

	UPROPERTY(VisibleAnywhere, BlueprintReadWrite, Category = Gameplay) // Added BlueprintReadWrite specifier
	FVector GunOffset;

protected:
	void OnFire();

	void MoveForward(float Value);
	void MoveRight(float Value);

	void TurnAtRate(float Rate);
	void LookAtRate(float Rate);

public:
	UPROPERTY(EditDefaultsOnly, Category = Projectile)
		TSubclassOf<class AProjectile> Projectile;

	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Gameplay)
		class USoundBase* FireSound;

	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Gameplay)
		class UAnimMontage* FireAnimation;

	class UAnimInstance* AnimInstance;

	class UWorld* World;
```

```
	FRotator SpawnRotation;
	FVector SpawnLocation;

	UPROPERTY(EditAnywhere, BlueprintReadOnly)
		float Health = 100.0f;
public:
	void DealDamage(float DamageAmount);
};
```

MChar.cpp

```
#include "MChar.h"
#include "Camera/CameraComponent.h"
#include "Components/CapsuleComponent.h"
#include "Components/InputComponent.h"
#include "Projectile.h"
#include "Animation/AnimInstance.h"
#include "Kismet/GameplayStatics.h"
#include "MainGM.h"
// Sets default values
AMChar::AMChar()
{
	// Set this character to call Tick() every frame.  You can turn this off to improve performance if you don't need it.
	PrimaryActorTick.bCanEverTick = true;

	GetCapsuleComponent()->InitCapsuleSize(40.0f, 95.0f);

		TurnRate = 45.0f;
		LookUpRate = 45.0f;

		FirstPersonCamera = CreateDefaultSubobject<UCameraComponent>(TEXT("First Person Camera"));
		FirstPersonCamera->SetupAttachment(GetCapsuleComponent());
		FirstPersonCamera->AddRelativeLocation(FVector(-39.65f, 1.75f, 64.0f));
		FirstPersonCamera->bUsePawnControlRotation = true;

		HandsMesh = CreateDefaultSubobject<USkeletalMeshComponent>(TEXT("Character Mesh"));
		HandsMesh->SetOnlyOwnerSee(true);
		HandsMesh->SetupAttachment(FirstPersonCamera);
		HandsMesh->bCastDynamicShadow = false;
		HandsMesh->CastShadow = false;
		HandsMesh->AddRelativeRotation(FRotator(1.9f, -19.19f, 5.2f));
```

```
		HandsMesh->AddRelativeLocation(FVector(-0.5f, -4.4f, -155.7f));

		GunMesh = CreateDefaultSubobject<USkeletalMeshComponent>(TEXT("GUN"));
		GunMesh->SetOnlyOwnerSee(true);
		GunMesh->bCastDynamicShadow = false;
		GunMesh->CastShadow = false;

		MuzzleLocation = CreateDefaultSubobject<USceneComponent>(TEXT("Muzzle Location"));
		MuzzleLocation->SetupAttachment(GunMesh);
		MuzzleLocation->SetRelativeLocation(FVector(0.2f, 48.4f, -10.6f));

		GunOffset = FVector(100.0f, 0.0f, 10.0f);
}

// Called when the game starts or when spawned
void AMChar::BeginPlay()
{
	Super::BeginPlay();

	GunMesh->AttachToComponent(HandsMesh,
		FAttachmentTransformRules::SnapToTargetIncludingScale,
		TEXT("GripPoint"));

	World = GetWorld();
	AnimInstance = HandsMesh->GetAnimInstance();

}

// Called every frame
void AMChar::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);

}

// Called to bind functionality to input
void AMChar::SetupPlayerInputComponent(UInputComponent* PlayerInputComponent)
{
	Super::SetupPlayerInputComponent(PlayerInputComponent);
	PlayerInputComponent->BindAction("Jump", IE_Pressed, this, &ACharacter::Jump);
	PlayerInputComponent->BindAction("Jump", IE_Released, this, &ACharacter::Jump);
```

```
	PlayerInputComponent->BindAction("Fire", IE_Pressed, this, &AMChar::OnFire);

	PlayerInputComponent->BindAxis("MoveForward", this, &AMChar::MoveForward);
	PlayerInputComponent->BindAxis("MoveRight", this, &AMChar::MoveRight);

	PlayerInputComponent->BindAxis("Turn", this, &AMChar::TurnAtRate);
	PlayerInputComponent->BindAxis("LookUp", this, &AMChar::LookAtRate);
}

void AMChar::OnFire()
{
	if (World != NULL)
	{

		SpawnRotation = GetControlRotation();

		SpawnLocation = ((MuzzleLocation != nullptr) ?
			MuzzleLocation->GetComponentLocation() :
			GetActorLocation()) + SpawnRotation.RotateVector(GunOffset);

		FActorSpawnParameters ActorSpawnParams;
		ActorSpawnParams.SpawnCollisionHandlingOverride =
			ESpawnActorCollisionHandlingMethod::AdjustIfPossibleButDontSpawnIfColliding;

		World->SpawnActor<AProjectile>(Projectile,
			SpawnLocation, SpawnRotation, ActorSpawnParams);

		if (FireSound != NULL)
		{
			UGameplayStatics::PlaySoundAtLocation(this, FireSound, GetActorLocation());
		}
		if (FireAnimation != NULL && AnimInstance != NULL)
		{
			AnimInstance->Montage_Play(FireAnimation, 1.0f);
		}
	}
}

void AMChar::MoveForward(float Value)
{
	if (Value != 0.0f)
	{
```

```
		AddMovementInput(GetActorForwardVector(), Value);
	}
}

void AMChar::MoveRight(float Value)
{
	if (Value != 0.0f)
	{
		AddMovementInput(GetActorRightVector(), Value);
	}
}

void AMChar::TurnAtRate(float Rate)
{
	AddControllerYawInput(Rate * TurnRate * GetWorld()->GetDeltaSeconds());
}

void AMChar::LookAtRate(float Rate)
{
	AddControllerPitchInput(Rate * LookUpRate * GetWorld()->GetDeltaSeconds());
}

void AMChar::DealDamage(float DamageAmount)
{
	Health -= DamageAmount;
	if (Health <= 0.0f)
	{
		AMainGM* MyGameMode =
			Cast<AMainGM>(UGameplayStatics::GetGameMode(GetWorld()));
		if (MyGameMode)
		{
			MyGameMode->RestartGameplay(false);
		}
		Destroy();
	}
}
```

Projectile.h (код, що описує снаряд)

```
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "Projectile.generated.h"

UCLASS()
class DIPLOMA_API AProjectile : public AActor
{
	GENERATED_BODY()

public:
	// Sets default values for this actor's properties
	AProjectile();

protected:
	// Called when the game starts or when spawned
	virtual void BeginPlay() override;

public:
	// Called every frame
	virtual void Tick(float DeltaTime) override;

	UPROPERTY(VisibleDefaultsOnly, Category = Projectile)
		class USphereComponent* CollisionSphere;

	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Movement)
		class UProjectileMovementComponent* ProjectileMovement;

	UFUNCTION()
		void OnHit(UPrimitiveComponent* HitComp, AActor* OtherActor, UPrimitiveComponent* OtherComp,
int32 OtherBodyIndex,
			bool bFromSweep, const FHitResult& Hit);

	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Projectile) // Added BlueprintReadWrite specifier
		float DamageValue = 20.0f;
};
```

Projectile.cpp

```
#include "Projectile.h"
#include "Components/SphereComponent.h"
#include "GameFramework/ProjectileMovementComponent.h"
#include "Enemy.h"
// Sets default values
AProjectile::AProjectile()
{
	// Set this actor to call Tick() every frame.  You can turn this off to improve performance if you don't need it.
	PrimaryActorTick.bCanEverTick = true;

	CollisionSphere = CreateDefaultSubobject<USphereComponent>(TEXT("Sphere Collision"));
	CollisionSphere->InitSphereRadius(20.0f);

	RootComponent = CollisionSphere;

	ProjectileMovement =
		CreateDefaultSubobject<UProjectileMovementComponent>(TEXT("Projectile Movement"));
	ProjectileMovement->UpdatedComponent = CollisionSphere;
	ProjectileMovement->InitialSpeed = 3000.0f;
	ProjectileMovement->MaxSpeed = 3000.0f;
	ProjectileMovement->bRotationFollowsVelocity = true;
	ProjectileMovement->bShouldBounce = true;

	InitialLifeSpan = 3.0f;


}

// Called when the game starts or when spawned
void AProjectile::BeginPlay()
{
	Super::BeginPlay();

	CollisionSphere->OnComponentBeginOverlap.AddDynamic(this, &AProjectile::OnHit);

}

// Called every frame
```

```
void AProjectile::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);

}

void AProjectile::OnHit(UPrimitiveComponent* HitComp, AActor* OtherActor, UPrimitiveComponent* OtherComp, int32
OtherBodyIndex, bool bFromSweep, const FHitResult& Hit)
{
	AEnemy* Enemy = Cast<AEnemy>(OtherActor);
	if (Enemy)
	{
		Enemy->DealDamage(DamageValue);
		Destroy();
	}
}
```

MainGM.h (код, що описує умови ігрового середовища)

```
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/GameMode.h"
#include "MainGM.generated.h"

/**
 *
 */
UCLASS()
class DIPLOMA_API AMainGM : public AGameMode
{
	GENERATED_BODY()

public:
	void RestartGameplay(bool Won);

private:
	void ResetLevel();
public:
	UPROPERTY(BlueprintReadOnly)
		int TimerCount = 300;
```

```
private:
	FTimerHandle CountDownTimerHandle = FTimerHandle();
	void CountdownTimer();
public:
	void BeginPlay() override;

};
```

MainGM.cpp

```
#include "MainGM.h"
#include "Kismet/GameplayStatics.h"
void AMainGM::RestartGameplay(bool Won)
{
	if (Won)
	{
		ResetLevel();
	}
	else
	{
		FTimerHandle TimerHandle;
		GetWorldTimerManager().SetTimer(TimerHandle, this,
			&AMainGM::ResetLevel, 3.0f);
	}
}

void AMainGM::ResetLevel()
{
	UGameplayStatics::OpenLevel(GetWorld(), "Gameplay");
}

void AMainGM::CountdownTimer()
{
	TimerCount--;

	if (TimerCount == 0)
	{
		GetWorldTimerManager().ClearTimer(CountDownTimerHandle);
		ResetLevel();
```

```
	}
}

void AMainGM::BeginPlay()
{
	Super::BeginPlay();

	GetWorldTimerManager().SetTimer(CountDownTimerHandle, this,
		&AMainGM::CountdownTimer, 1.0f, true, 1.0f);
}
```

Enemy.h (код, який описує поведінку підконтрольного ШІ НІП)

```
#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "Enemy.generated.h"

UCLASS()
class DIPLOMA_API AEnemy : public ACharacter
{
	GENERATED_BODY()

public:
	// Sets default values for this character's properties
	AEnemy();

protected:
	// Called when the game starts or when spawned
	virtual void BeginPlay() override;

public:
	// Called every frame
	virtual void Tick(float DeltaTime) override;

	UPROPERTY(EditAnywhere)
		class UBoxComponent* DamageCollision;

	UFUNCTION()
		void OnHit(UPrimitiveComponent* HitComp, AActor* OtherActor, UPrimitiveComponent* OtherComp,
int32 OtherBodyIndex,
			bool bFromSweep, const FHitResult& Hit);
```

```
	UPROPERTY(VisibleDefaultsOnly, Category = Enemy)
		class UAIPerceptionComponent* AIPerComp;

	UPROPERTY(VisibleDefaultsOnly, Category = Enemy)
		class UAISenseConfig_Sight* SightConfig;

	UFUNCTION()
		void OnSensed(const TArray<AActor*>& UpdatedActors);

	UPROPERTY(VisibleAnywhere, Category = Movement)
		FRotator EnemyRotation;

	UPROPERTY(VisibleAnywhere, Category = Movement)
		FVector BaseLocation;

	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Movement)
		FVector CurrentVelocity;

	UPROPERTY(VisibleAnywhere, Category = Movement)
		float MovementSpeed;

	void SetNewRotation(FVector TargetPosition, FVector CurrentPosition);

	bool BackToBaseLocation;
	FVector NewLocation;
	float DistanceSquared;

	UPROPERTY(EditAnywhere, BlueprintReadOnly)
		float Health = 100.0f;

	UPROPERTY(EditAnywhere)
		float DamageValue = 5.0f;

public:
	void DealDamage(float DamageAmount);
};
```

Enemy.cpp

```
 #include "Enemy.h"
```

```
#include "Components/BoxComponent.h"
#include "MChar.h"
#include "Perception/AISenseConfig_Sight.h"
#include "Perception/AIPerceptionComponent.h"
// Sets default values
AEnemy::AEnemy()
{
	// Set this character to call Tick() every frame.  You can turn this off to improve performance if you don't need it.
	PrimaryActorTick.bCanEverTick = true;

	DamageCollision = CreateDefaultSubobject<UBoxComponent>(TEXT("Damage Collision"));
	DamageCollision->SetupAttachment(RootComponent);

	AIPerComp = CreateDefaultSubobject<UAIPerceptionComponent>(TEXT("AI Perception Component"));
	SightConfig = CreateDefaultSubobject<UAISenseConfig_Sight>(TEXT("Sight Config"));

	SightConfig->SightRadius = 1250.0f;
	SightConfig->LoseSightRadius = 1280.0f;
	SightConfig->PeripheralVisionAngleDegrees = 90.0f;
	SightConfig->DetectionByAffiliation.bDetectEnemies = true;
	SightConfig->DetectionByAffiliation.bDetectFriendlies = true;
	SightConfig->DetectionByAffiliation.bDetectNeutrals = true;
	SightConfig->SetMaxAge(0.1f);

	AIPerComp->ConfigureSense(*SightConfig);
	AIPerComp->SetDominantSense(SightConfig->GetSenseImplementation());
	AIPerComp->OnPerceptionUpdated.AddDynamic(this, &AEnemy::OnSensed);

	CurrentVelocity = FVector::ZeroVector;
	MovementSpeed = 375.0f;

	DistanceSquared = BIG_NUMBER;


}

// Called when the game starts or when spawned
void AEnemy::BeginPlay()
{
	Super::BeginPlay();

	DamageCollision->OnComponentBeginOverlap.AddDynamic(this, &AEnemy::OnHit);
```

```
	BaseLocation = this->GetActorLocation();
}

// Called every frame
void AEnemy::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);

	if (!CurrentVelocity.IsZero())
	{
		NewLocation = GetActorLocation() + CurrentVelocity * DeltaTime;

		if (BackToBaseLocation)
		{
			if ((NewLocation - BaseLocation).SizeSquared2D() < DistanceSquared)
			{
				DistanceSquared = (NewLocation - BaseLocation).SizeSquared2D();

			}
			else
			{
				CurrentVelocity = FVector::ZeroVector;
				DistanceSquared = BIG_NUMBER;
				BackToBaseLocation = false;

				SetNewRotation(GetActorForwardVector(), GetActorLocation());
			}
		}
		SetActorLocation(NewLocation);
	}

}


void AEnemy::OnHit(UPrimitiveComponent* HitComp, AActor* OtherActor, UPrimitiveComponent* OtherComp, int32
OtherBodyIndex, bool bFromSweep, const FHitResult& Hit)
{
	AMChar* Char = Cast<AMChar>(OtherActor);
	if (Char)
	{
		Char->DealDamage(DamageValue);
```

```
	}
}

void AEnemy::OnSensed(const TArray<AActor*>& UpdatedActors)
{
	for (int i = 0; i < UpdatedActors.Num(); i++)
	{
		FActorPerceptionBlueprintInfo Info;

		AIPerComp->GetActorsPerception(UpdatedActors[i], Info);
		if (Info.LastSensedStimuli[0].WasSuccessfullySensed())
		{
			FVector dir = UpdatedActors[i]->GetActorLocation() - GetActorLocation();
			dir.Z = 0.0f;

			CurrentVelocity = dir.GetSafeNormal() * MovementSpeed;

			SetNewRotation(UpdatedActors[i]->GetActorLocation(), GetActorLocation());

		}
		else
		{
			FVector dir = BaseLocation - GetActorLocation();
			dir.Z = 0.0f;

			if (dir.SizeSquared2D() > 1.0f)
			{
				CurrentVelocity = dir.GetSafeNormal() * MovementSpeed;
				BackToBaseLocation = true;

				SetNewRotation(BaseLocation, GetActorLocation());

			}
		}
	}
}

void AEnemy::SetNewRotation(FVector TargetPosition, FVector CurrentPosition)
{

	FVector NewDirection = TargetPosition - CurrentPosition;
	NewDirection.Z = 0.0f;
```

```
	EnemyRotation = NewDirection.Rotation();

	SetActorRotation(EnemyRotation);

}

void AEnemy::DealDamage(float DamageAmount)
{
	Health -= DamageAmount;
	if (Health <= 0.0f)
	{
		Destroy();
	}
}
```

Door.h (код, який описує двері)

```
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "Door.generated.h"

UCLASS()
class DIPLOMA_API ADoor : public AActor
{
	GENERATED_BODY()

public:
	// Sets default values for this actor's properties
	ADoor();

protected:
	// Called when the game starts or when spawned
	virtual void BeginPlay() override;

public:
	// Called every frame
	virtual void Tick(float DeltaTime) override;
```

```
	UPROPERTY(EditAnywhere)
		class UStaticMeshComponent* DoorMesh;

	UPROPERTY(EditAnywhere)
		class UBoxComponent* CollisionComponent;

	UFUNCTION()
		void OnHit(UPrimitiveComponent* HitComp, AActor* OtherActor,
			UPrimitiveComponent* OtherComp, int32 OtherBodyIndex,
			bool bFromSweep, const FHitResult& Hit);


};
```

Door.cpp

```
#include "Door.h"
#include "Components/StaticMeshComponent.h"
#include "Components/BoxComponent.h"
#include "MChar.h"
#include "MainGM.h"
#include "Kismet/GameplayStatics.h"

// Sets default values
ADoor::ADoor()
{
	// Set this actor to call Tick() every frame.  You can turn this off to improve performance if you don't need it.
	PrimaryActorTick.bCanEverTick = true;

	DoorMesh = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Door Mesh"));
	RootComponent = DoorMesh;
	CollisionComponent = CreateDefaultSubobject<UBoxComponent>(TEXT("Collision Component"));
	CollisionComponent->SetupAttachment(DoorMesh);
}

// Called when the game starts or when spawned
void ADoor::BeginPlay()
{
	Super::BeginPlay();
```

```
	CollisionComponent->OnComponentBeginOverlap.AddDynamic(this, &ADoor::OnHit);
}

// Called every frame
void ADoor::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);

}

void ADoor::OnHit(UPrimitiveComponent* HitComp, AActor* OtherActor, UPrimitiveComponent* OtherComp, int32
OtherBodyIndex, bool bFromSweep, const FHitResult& Hit)
{
	AMChar* Char = Cast<AMChar>(OtherActor);

	if (Char)
	{
		AMainGM* MyGameMode =
			Cast<AMainGM>(UGameplayStatics::GetGameMode(GetWorld()));

		if (MyGameMode)
		{
			MyGameMode->RestartGameplay(true);
		}
	}
}
```

ДОДАТОК Б

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файлу	**Опис**
Пояснювальні документи	
КР_Гамзін.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
КР_Гамзін.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Код	Пояснювальна записка до кваліфікаційної роботи, що містить код програми, у форматі txt
Програма	
Program.rar	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація_Гамзін.ppt	Презентація кваліфікаційної роботи