ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Створення гри в жанрі Зомбі-шутер на базі рушія Unreal Engine»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Владислав КРИЦЬКИЙ
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

Владислав КРИЦЬКИЙ

Керівник: Треньов МИКИТА
асистент кафедри ІПЗ

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Крицькому Владиславу Ігоровичу

1. Тема кваліфікаційної роботи: «Створення гри в жанрі Зомбі-шутер на базі рушія Unreal Engine»
керівник кваліфікаційної роботи асистент кафедри ІПЗ Микита ТРЕНЬОВ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи розробки гри в жанрі зомбі-шутер на рушії Unreal Engine та способи реалізації ігрових механік.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   4.1 Аналіз предметної області
   4.2 Визначення вимог до реалізації
   4.3 Визначення засобів реалізації
   4.4 Розробка гри відповідно до визначених засобів та вимог

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Концепт гри.
3. Вимоги до програмного забезпечення.
4. Програмні засоби реалізації.
5. Use-case діаграма.
6. Діаграма класів.
7. Файлова структура проекту.
8. Екранні форми.
9. Екранні форми.
10. Екранні форми.
11. Екранні форми.
12. Демонстрація гри.
13. Апробація результатів дослідження.

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Аналіз можливостей ігрового рушія Unreal Engine при створенні ігор в жанрі зомбі шутер	13.03-30.03.2024	
4	Проектування геймплею гри INFECTED в жанрі зомбі-шутер	30.03-05.04.2024	
5	Підбір елементів ігрового світу для гри INFECTED на існуючих платформах	05.04-08.04.2024	
6	Розробка гри INFECTED та її тестування	08.04-29.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Владислав КРИЦЬКИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Микита ТРЕНЬОВ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 50 стор., 2 табл., 26 рис., 8 джерел.

Мета роботи – збільшення зацікавленості ігровим процесом в грі жанру "Зомбі-шутер" під платформу Windows за рахунок використання нескінченного геймплею і орієнтації на рекордний результат.

Об'єкт дослідження – ігровий процес в грі жанру зомбі-шутер під платформу Windows.

Предмет дослідження – гра в жанрі зомбі-шутер під платформу Windows з нескінченним ігровим процесом та орієнтацією на рекордний результат.

Короткий зміст роботи: В роботі проаналізовано історію розвитку відеоігор та популярні ігри в жанрі зомбі-шутер, такі як Left 4 Dead та DayZ. Проведено детальний аналіз ігрових рушіїв, зокрема Unreal Engine, Unity, CryEngine та Godot, а також середовищ розробки, таких як Visual Studio, Rider та Visual Studio Code. Розроблено алгоритми роботи гри INFECTED та програмно реалізовані ключові ігрові механіки, зокрема: головне меню гри, логіка смерті гравця, логіка підбирання набоїв, логіка перезаряджання, логіка регенерації здоров'я гравця, логіка отримання ушкоджень гравцем, логіка прицілювання, логіка стрільби, логіка нанесення ушкоджень, логіка руху зомбі, логіка прискорення зомбі за умови що гравець поруч, логіка наздоганяння гравця, логіка атакування гравця, логіка нанесення урону зомбі, логіка смерті зомбі, логіка початку гри, логіка початку наступної хвилі, логіка спавну зомбі, логіка перевірка чи всі зомбі в хвилі мертві, логіка зміни хвилі, логіка спавну ящиків з набоями. В роботі використано Unreal Engine 5, Visual Studio.

Сферою використання гри є надання гравцям можливості отримати нові враження та відпочити від різних психологічних навантажень.

КЛЮЧОВІ СЛОВА: ЗОМБІ-ШУТЕР, UNREAL ENGINE, ГЕЙМПЛЕЙ, BLUEPRINT, ГРА.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі відеоігри стали не лише засобом розваги, але й важливим елементом культури та комунікації. Гравці з усього світу взаємодіють у віртуальних просторах, обмінюються досвідом та формують спільноти за інтересами. Жанр зомбі-шутера залишається популярним завдяки своїй здатності поєднувати динамічний геймплей з елементами виживання та хорору, що забезпечує емоційне занурення та адреналін. Також в умовах стресу та постійного інформаційного тиску відеоігри виконують важливу психологічну функцію, надаючи можливість відволіктися та розрядитися. Ігри в жанрі зомбі-шутер дозволяють гравцям виплеснути накопичену агресію в безпечному середовищі, що сприяє зниженню рівня стресу. Крім того, ці ігри можуть розвивати навички швидкого прийняття рішень, стратегічного мислення та координації рухів. З огляду на сучасні світові події та виклики, зокрема пандемії та зростання соціальної напруги, тематика виживання та боротьби з загрозами стає ще більш актуальною. А оскільки рушій Unreal Engine є одним з найбільш потужних та універсальних інструментів для розробки відеоігор. Його використання дозволяє створювати високоякісну графіку, реалістичні фізичні ефекти та складні ігрові механіки. З розвитком технологій та збільшенням продуктивності сучасних пристроїв, особливо мобільних, створення зомбі-шутерів на базі Unreal Engine стає все більш реалістичним та доступним. Це відкриває нові горизонти для розробників та надає гравцям можливість насолоджуватися більш захоплюючими іграми.

Мета роботи – збільшення зацікавленості ігровим процесом в грі жанру "Зомбі-шутер" під платформу Windows за рахунок використання нескінченного геймплею і орієнтації на рекордний результат.

Об’єкт дослідження – ігровий процес в грі жанру зомбі-шутер під платформу Windows.

Предмет дослідження – гра в жанрі зомбі-шутер під платформу Windows з нескінченним ігровим процесом та орієнтацією на рекордний результат.

Для досягнення мети необхідно пройти наступні етапи:

1. Провести огляд та аналіз існуючих ігор жанру зомбі-шутер.
2. Провести аналіз можливостей ігрового рушія Unreal Engine при створенні ігор в жанрі зомбі-шутер.
3. Розробити концепт гри INFECTED та спроектувати її геймплей в жанрі зомбі-шутер.
4. Виконати підбір елементів ігрового світу для гри INFECTED на існуючих платформах.
5. Розробити файлову структуру проекту гри INFECTED, спроектувати класи застосунку.
6. Розробити програмне забезпечення гри INFECTED.
7. Провести тестування гри INFECTED.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Історія розвитку відеоігор

Відеоігри мають багату і захоплюючу історію, яка охоплює понад півстоліття інновацій та технічного прогресу. Початок цієї історії можна віднести до 1950-х років, коли у наукових лабораторіях з’явилися перші експериментальні ігри.

1950-1960-ті роки: У цей період відеоігри були в основному експериментальними проектами в університетах і дослідницьких установах. Однією з перших ігор була "Tennis for Two", створена в 1958 році Вільямом Хігінботамом для демонстрації на науковій виставці. Іншою значущою грою стала "Spacewar!" (1962) – це була перша гра, яка здобула популярність серед ентузіастів і програмістів.

1970-ті роки: Це десятиліття стало ерою аркадних ігор. "Pong" від Atari, випущена в 1972 році, стала першою комерційно успішною відеогрою, що започаткувала хвилю аркадних автоматів. У цей період також почали з’являтися домашні консолі, такі як Magnavox Odyssey (1972).

1980-ті роки: Вісімдесяті стали золотим віком аркадних ігор та появи домашніх ігрових консолей. Ігри, як "Pac-Man" (1980) і "Donkey Kong" (1981), стали культурними феноменами. У 1983 році Nintendo випустила Nintendo Entertainment System (NES), яка заклала основу для сучасної індустрії відеоігор. Ігри, такі як "Super Mario Bros." (1985) та "The Legend of Zelda" (1986), стали класикою і вплинули на розвиток ігрового дизайну.

1990-ті роки: Цей період ознаменувався переходом від двовимірної графіки до тривимірної (3D). Гра "Doom" (1993) від id Software стала революційною в жанрі шутерів від першої особи. Sony PlayStation, випущена в 1994 році, та Nintendo 64 (1996) стали популярними консолями з підтримкою 3D-

графіки. У цей час також почали з’являтися перші багатокористувацькі онлайн-ігри.

2000-ті роки: Початок нового тисячоліття ознаменувався швидким розвитком інтернет-технологій, що призвело до популяризації масових багатокористувацьких онлайн-ігор (MMORPG). Гра "World of Warcraft" (2004) стала однією з найуспішніших в цьому жанрі. Крім того, мобільні ігри почали набирати популярність завдяки розвитку смартфонів та планшетів.

2010-ті роки: Цей період відзначився розвитком технологій віртуальної реальності (VR) та доповненої реальності (AR). Випуск пристроїв, таких як Oculus Rift (2016), дозволив гравцям занурюватися в ігровий світ як ніколи раніше. Гра "Pokémon Go" (2016) популяризувала технологію AR. Також розвивалися інді-ігри, які стали важливою частиною індустрії завдяки платформам, таким як Steam та інших цифрових магазинів.

1.2 Аналіз популярних ігор у жанрі зомбі-шутер

Жанр зомбі-шутер завжди привертав увагу гравців завдяки поєднанню елементів хорору, виживання та інтенсивного бойового геймплею. Дві з найвідоміших та впливових ігор у цьому жанрі — Left 4 Dead та DayZ. Тому розглянемо їх детальніше.

Випущена компанією Valve у 2008 році, Left 4 Dead стала важливим кроком у розвитку кооперативних шутерів. Гра побудована навколо співпраці гравців. Вона підтримує режим гри в командах по чотири людини, що змушує гравців працювати разом, щоб вижити в світі, заповненому ордами зомбі.

А також спеціальний штучний інтелект, який змінює розташування ворогів і події в залежності від дій гравців. Завдяки цьому кожен прохід гри унікальний, і гравці не можуть передбачити, що станеться далі. У грі також представлені різні персонажі з унікальними навичками та історіями, що додає грі глибини і дозволяє гравцям вибирати стилі гри, які їм найбільше підходять. Left 4 Dead

пропонує швидкий і захоплюючий бойовий геймплей, де гравцям доводиться постійно бути напоготові та швидко реагувати на атаки зомбі.

DayZ — це гра у жанрі виживання з елементами зомбі-шутера, розроблена Bohemia Interactive. Спочатку була випущена як модифікація для ARMA 2 у 2012 році, а пізніше стала самостійною грою. DayZ відрізняється відкритим світом, де гравці повинні вижити, збираючи ресурси, їжу, воду та інше необхідне для виживання. Гра має реалістичну систему потреб, таких як голод і спрага. Однією з ключових особливостей DayZ є взаємодія з іншими гравцями. Гравці можуть об'єднуватися у групи для виживання або боротися один проти одного за ресурси. Проте якщо гравець гине, він втрачає всі свої ресурси і починає з самого початку. Це додає грі високого рівня напруги і змушує гравців ретельно обдумувати свої дії. DayZ пропонує великий відкритий світ, який гравці можуть досліджувати. Кожне місце має свої унікальні ресурси і небезпеки.

Обидві гри залишили значний слід у жанрі зомбі-шутерів і продовжують мати велику аудиторію гравців. Кожна з них пропонує унікальний підхід до теми зомбі-апокаліпсису, що робить їх важливими представниками жанру.

Рис. 1.1. Логотипи ігор

На рис.1.2 та 1.3 продемонстровані скріншоти на яких продемонстрована частина геймплею з ігор які порівнюються. Далі в таб.1.1 буде наведена порівняльна таблиця.

Рис. 1.2. Скріншот гри Left 4 Dead

Рис. 1.3. Скріншот гри DayZ

Таблиця 1.1

Аналіз аналогів

Характеристика гри	**Left 4 Dead**	**DayZ**	**INFECTED**
Режим гри	**Кооперативний**	**Онлайн**	**Офлайн**
Розмір на жорсткому диску	**10 ГБ**	**20 ГБ**	**5 ГБ**
Кількість гравців	**1-4**	**1-60**	**1**
Графіка	**3D**	**3D**	**3D**
Формат геймплею	**Кооперативні кампанії**	**Відкритий світ**	**Проходження на рекорд, геймплей нескінченний**

В таблиці було показано що розроблена гра INFECTED займає менше місця а також має нескінченний геймплей, а ще оскільки гра INFECTED створювалась нещодавно то ми можемо побачити велику різницю в 3D графіці яка використовується в усіх трьох іграх.

1.3 Висновки до розділу

Аналіз історії розвитку відеоігор показує, що цей сектор постійно еволюціонує, адаптуючись до нових технологій та змін у вподобаннях гравців. Жанр зомбі-шутер виділяється серед інших завдяки своїй здатності поєднувати елементи хорору, виживання та інтенсивного бойового геймплею. Популярні ігри цього жанру, такі як Left 4 Dead та DayZ, демонструють важливість якісного наративу, атмосферного дизайну та інноваційних механік. Для створення успішної гри в цьому жанрі необхідно враховувати ці аспекти і використовувати сучасні технології для реалізації захопливого ігрового досвіду.

2. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ РЕАЛІЗАЦІЇ ІГРОВОГО ДОДАТКУ

2.1. Аналіз популярних ігрових двигунів

Вибір ігрового рушія є важливим кроком у процесі розробки гри, оскільки кожен з них має свої особливості та переваги. Серед найпопулярніших рушіїв можна виділити наступні:

Unreal Engine: Створений компанією Epic Games, цей рушій вирізняється високоякісною графікою, підтримкою різних платформ (ПК, консолі, мобільні пристрої, VR) та великою спільнотою розробників. Unreal Engine використовує C++ і надає можливість візуального програмування за допомогою Blueprints, що сприяє швидкому створенню прототипів та впровадженню ігрових механік. До його переваг також належить велика бібліотека готових ресурсів та інтеграція з багатьма іншими інструментами для розробки ігор. Unreal Engine також забезпечує високий рівень реалістичності завдяки таким технологіям, як трасування променів (Ray Tracing) та підтримка великих відкритих світів, що робить його вибором номер один для розробки високобюджетних AAA ігор.

Unity: Завдяки своїй гнучкості та доступності, Unity є одним із найпопулярніших ігрових рушіїв. Він підтримує численні платформи, включаючи ПК, мобільні пристрої, консолі та VR, та використовує C#. Зручний інтерфейс, широкі можливості та велика бібліотека ресурсів роблять Unity ідеальним для створення як 2D, так і 3D ігор. Крім того, активна спільнота розробників допомагає вирішувати проблеми та ділитися досвідом. Unity також підтримує AR (доповнену реальність) та MR (змішану реальність), що робить його дуже привабливим для розробників, які прагнуть створювати інноваційні та інтерактивні проекти. Завдяки Asset Store, розробники можуть придбати або завантажити безліч готових ресурсів, що значно пришвидшує процес розробки.

CryEngine: Відомий своєю реалістичною графікою та високою продуктивністю, цей рушій від Crytek використовується у багатьох високобюджетних іграх. CryEngine забезпечує вражаючу деталізацію, має потужні інструменти для розробки графіки та фізики, проте може бути складнішим у вивченні. Для програмування він використовує C++ та Lua. CryEngine дозволяє створювати неймовірно реалістичні ігрові світи завдяки таким функціям, як розширена система частинок, реалістичне освітлення та передові технології тіней. Цей рушій також забезпечує високу продуктивність та оптимізацію, що є критично важливим для створення ігор з великою кількістю деталей та складною фізикою.

Godot: Це відкритий рушій з відкритим вихідним кодом, який швидко набирає популярність завдяки своїй простоті у використанні та активній спільноті розробників. Godot підтримує розробку як 2D, так і 3D ігор, використовуючи мову програмування GDScript (аналогічну Python), а також C# та VisualScript. Зручний інтерфейс та модульна структура дозволяють легко налаштовувати середовище під потреби розробників. Godot також відзначається своєю легкістю у налаштуванні та гнучкістю, що дозволяє розробникам швидко адаптувати рушій під свої проекти. Завдяки відкритому вихідному коду, розробники можуть вносити власні зміни та покращення, що робить Godot привабливим вибором для інді-розробників та невеликих студій, які шукають доступний та функціональний інструмент для створення ігор.

Кожен з цих рушіїв має свої унікальні особливості, що робить їх відповідними для різних типів проектів. Вибір конкретного рушія залежить від цілей проекту, технічних вимог та рівня підготовки команди розробників. Незалежно від вибору, всі ці рушії пропонують потужні інструменти та можливості для створення вражаючих ігор, що відповідають найвищим стандартам якості.

2.2. Аналіз середовищ розробки

Вибір середовища розробки є критично важливим етапом у процесі створення гри, оскільки воно впливає на ефективність роботи команди розробників і загальну якість проекту. Нижче наведено аналіз найпопулярніших середовищ розробки:

Visual Studio: Це потужне середовище розробки від Microsoft, яке часто використовується разом з Unreal Engine та іншими рушіями. Visual Studio підтримує різні мови програмування, зокрема C++, C# та Python, і має інтеграцію з системами контролю версій, такими як Git та SVN. Середовище пропонує широкий набір інструментів для налагодження та оптимізації коду, включаючи розширені можливості для роботи з великими проектами. Завдяки вбудованим інструментам для аналізу продуктивності та діагностики, розробники можуть ефективно виявляти та усувати проблеми в коді. Visual Studio також підтримує різноманітні плагіни, що розширюють функціональність та дозволяють налаштувати середовище під конкретні потреби проекту.

Rider: Це середовище розробки від JetBrains, яке також підтримує Unreal Engine та Unity. Rider спеціалізується на роботі з C# і надає розширені можливості для налагодження та аналізу коду. Воно має інтеграцію з системами контролю версій та інші корисні інструменти для розробників ігор, включаючи аналізатор коду, який допомагає знаходити помилки та потенційні проблеми ще на етапі написання коду. Rider також підтримує роботу з базами даних, що робить його зручним для проектів, які потребують складних серверних рішень. Завдяки інтелектуальним підказкам та автоматичному завершенню коду, розробка стає швидшою та ефективнішою.

Visual Studio Code: Це легке, але потужне середовище розробки від Microsoft, яке підтримує безліч мов програмування завдяки розширенням. Visual Studio Code є безкоштовним та має велику спільноту розробників, які створюють додатки та плагіни для розширення його функціональності. Це середовище ідеально підходить для тих, хто шукає легке та налаштовуване середовище для

розробки ігор. Visual Studio Code підтримує інтеграцію з системами контролю версій та має інструменти для налагодження, що робить його чудовим вибором для інді-розробників та невеликих студій. Крім того, це середовище дозволяє швидко переключатися між різними проектами та мовами програмування, що є важливим для мультидисциплінарних команд.

Eclipse: Це популярне середовище розробки для Java, яке також може бути використане для розробки ігор, особливо для мобільних платформ. Eclipse має розширення для підтримки інших мов програмування та інтеграції з різними інструментами для розробки. Воно надає потужні інструменти для налагодження, аналізу та оптимізації коду, що робить його привабливим для розробників, які працюють з великими та складними проектами. Eclipse також підтримує інтеграцію з системами контролю версій та має велику кількість плагінів, що дозволяє налаштовувати середовище під специфічні потреби проекту. Завдяки широким можливостям налаштування та масштабованості, Eclipse є хорошим вибором для команд, які шукають стабільне та надійне середовище розробки.

Кожне з цих середовищ розробки має свої унікальні особливості та переваги, що робить їх відповідними для різних типів проектів та рівнів досвіду розробників. Вибір конкретного середовища залежить від вимог проекту, технологічного стека та особистих уподобань команди. Незалежно від вибору, всі ці середовища надають потужні інструменти та можливості для створення високоякісних ігор.

2.3. Висновки до розділу

Аналіз популярних ігрових рушіїв та середовищ розробки показує, що кожен з них має свої унікальні особливості та переваги. Unreal Engine виділяється своєю високоякісною графікою та гнучкістю, що робить його ідеальним вибором для створення візуально вражаючої гри у жанрі зомбі-шутер.

Середовища розробки, такі як Visual Studio, Rider та Visual Studio Code, забезпечують розробникам потужні інструменти для створення та налагодження ігор. Але оскільки гра буде створюватись на базі рушія Unreal Engine то було вирішено використовувати Visual Studio.

3. ПРОЕКТУВАННЯ ТА РОЗРОБКА ГРИ

3.1. Опис геймплею гри

У нашій грі гравець опиняється в світі де зомбі ніколи не закінчуються, і його завдання - вижити якомога довше. Гравець має обмежений запас здоров'я, яке зменшується кожен раз, коли він отримує ушкодження від зомбі. Проте, якщо протягом 5 секунд гравець не отримує жодного ушкодження, його здоров'я починає відновлюватися.

Гравець змушений битися зі зброєю проти безлічі зомбі, які намагаються його атакувати. Щоб уникнути смерті, гравець повинен активно слідкувати за своїм здоров'ям та уникати отримання великої кількості ушкоджень. Крім того, якщо гравець надто наблизиться до зомбі, вони почнуть рухатися швидше, що ускладнить уникнення атак.

Для того, щоб допомогти гравцю у виживанні, після кожної хвилі на мапі генерується ящик з комплектом набоїв. Гравець може підібрати цей ящик, щоб поповнити свій запас боєприпасів та продовжити боротьбу проти зомбі.

Гра триває, доки гравець не помре від атак зомбі. Чим довше йому вдасться виживати, тим вищий його рахунок.

3.2. Планування розробки ПЗ

Для розробки зомбі-шутера потрібно зібрати основні ігрові елементи: моделі будівель, транспорту і об’єкти оточення. Їх можна знайти у відкритих бібліотеках 3D-моделей, таких як Unity Asset Store, Unreal Marketplace. Після цього слід завантажити та адаптувати ці моделі під стиль гри та технічні вимоги, такі як формат, кількість полігонів і оптимізація.

Першим кроком у створенні гри є вибір ігрового двигуна (наприклад Unreal Engine) і створення нового проекту. Далі додається і налаштовується

камера, а також налаштовується інтерфейс користувача (UI) з такими елементами, як лічильник здоров'я та патрони.

Після чого для головного меню потрібно створити нову сцену, додати елементи UI, такі як кнопки "почати гру" та "вихід", написати скрипти для обробки натискань кнопок та додати анімації або графічні елементи для покращення зовнішнього вигляду.

Створюємо ігрового персонажа, що передбачає знаходження або створення 3D-моделі, налаштування анімацій для різних дій (рух, атака, стрибки), додавання контролера для управління його рухами та реалізацію взаємодії з оточенням, наприклад, зіткнень або використання об'єктів.

Додаємо ворогів-зомбі для яких необхідно знайти або створити 3D-модель, налаштувати анімації (хід, атака, смерть), додати AI для керування їх поведінкою (переслідування гравця, атака) та реалізувати системи здоров'я та пошкоджень.

Створення ігрової логіки включає визначення основних механік (стрільба, підбір предметів, переміщення), написання скриптів для їх обробки, тестування логіки та реалізацію систем збереження прогресу та завдань.

Далі йде створення ігрової локації яка потребує створення або завантаження 3D-моделей оточення, налаштування сцен (додавання моделей, текстур, освітлення), розміщення об'єктів та ворогів, а також додавання тригерів та інших елементів, які впливають на геймплей (наприклад, пастки, двері, ліфти).

І остання деталь створення ящиків з набоями які також потребують знаходження або створення 3D-моделі, налаштування анімацій для взаємодії (відкриття ящика), написання скриптів, які додаватимуть патрони до інвентаря гравця при взаємодії з ящиком, та розміщення ящиків на локаціях з урахуванням балансу гри.

3.3. Реалізація гри

3.3.1 Головне меню гри

На знімку екрана на рис. 3.1 представлено головне меню гри, розробленої на базі рушія Unreal Engine, а також його ключові компоненти. У цьому меню користувачі можуть побачити основні опції, такі як початок гри та вихід з гри, що забезпечує зручну навігацію ігровим процесом.

Рис. 3.1 Головне меню гри

3.3.2 Логіка смерті гравця

На знімку екрана на рис. 3.2 відображена послідовність подій, що відбуваються під час смерті гравця. Ця логіка реалізована за допомогою Blueprints. Після спрацьовування цього івенту змінна отримує значення, що викликає відображення вікна смерті гравця. Після цього гравцю втрачається можливість

керувати персонажем, а замість цього він отримує можливість керувати курсором, щоб обрати одну з запропонованих кнопок в меню.

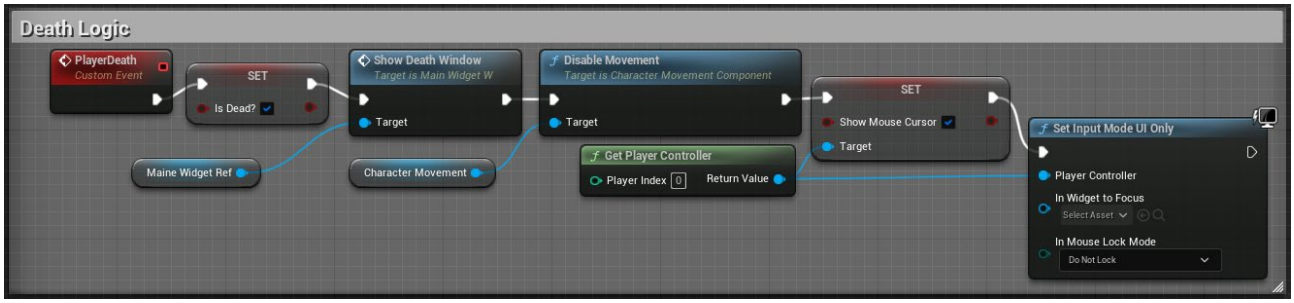


Рис. 3.2 Логіка смерті гравця

3.3.3 Логіка підбирання набоїв

Знімок екрана на рис. 3.3 показує подію підбирання набоїв, після виконання якої гравцеві автоматично додається 100 набоїв, а також відтворюється звукова доріжка.

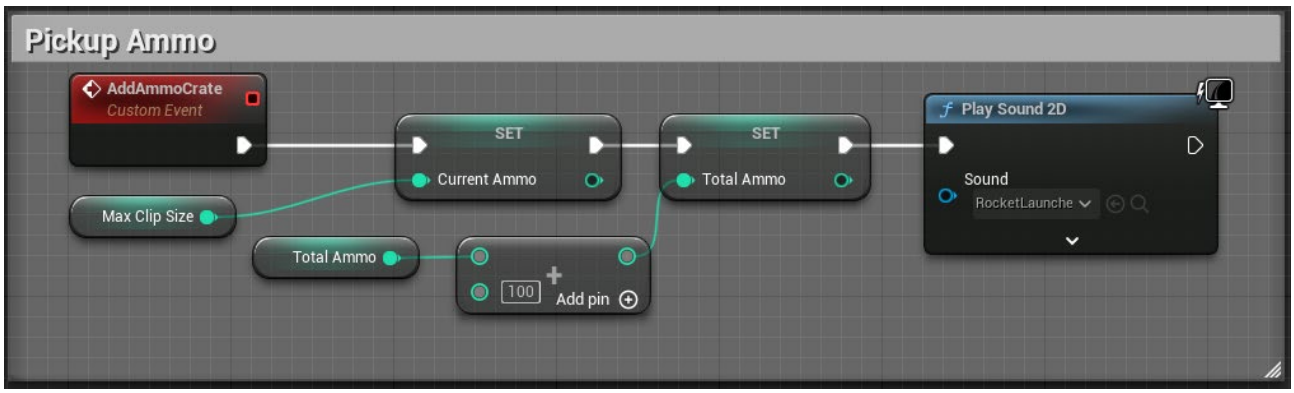


Рис. 3.3 Логіка підбирання набоїв

3.3.4 Логіка перезаряджання

На знімку екрана на рис. 3.4 демонструється логіка перезаряджання зброї у грі, де натискання клавіші R викликає певну подію. Після цього перевіряється кількість набоїв у магазині: якщо кількість набоїв не досягла максимального значення, проводяться розрахунки щодо того, скільки набоїв потрібно для перезаряджання, скільки залишиться у стволі та скільки буде у запасі після перезаряджання. Після цього дані оновлюються, і відтворюється анімаційний

монтаж та анімація процесу перезаряджання зброї. Для уникнення циклічності процесу, значення змінної, яка відповідає за перевірку можливості перезарядки, змінюється.

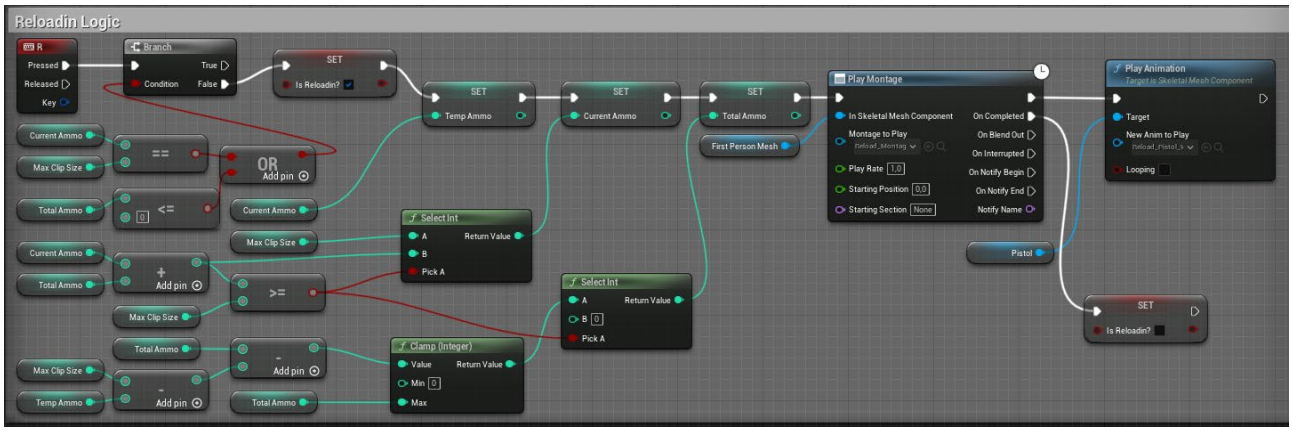


Рис. 3.4 Логіка перезаряджання

3.3.5 Логіка регенерації здоровя гравця

На знімку екрана на рис. 3.5 відображена логіка регенерації здоров'я гравця, в якій якщо протягом 5 секунд гравець не отримує ушкоджень і його здоров'я менше 90, то кожні 5 секунд гравець регенерує 5 одиниць здоров'я. Проте якщо його здоров'я вже дорівнює 90, таймер зупиняється. Коли рівень здоров'я гравця перевищує 50 одиниць, екран припиняє червоніти, що сигналізує гравцю про поліпшення стану його персонажа.

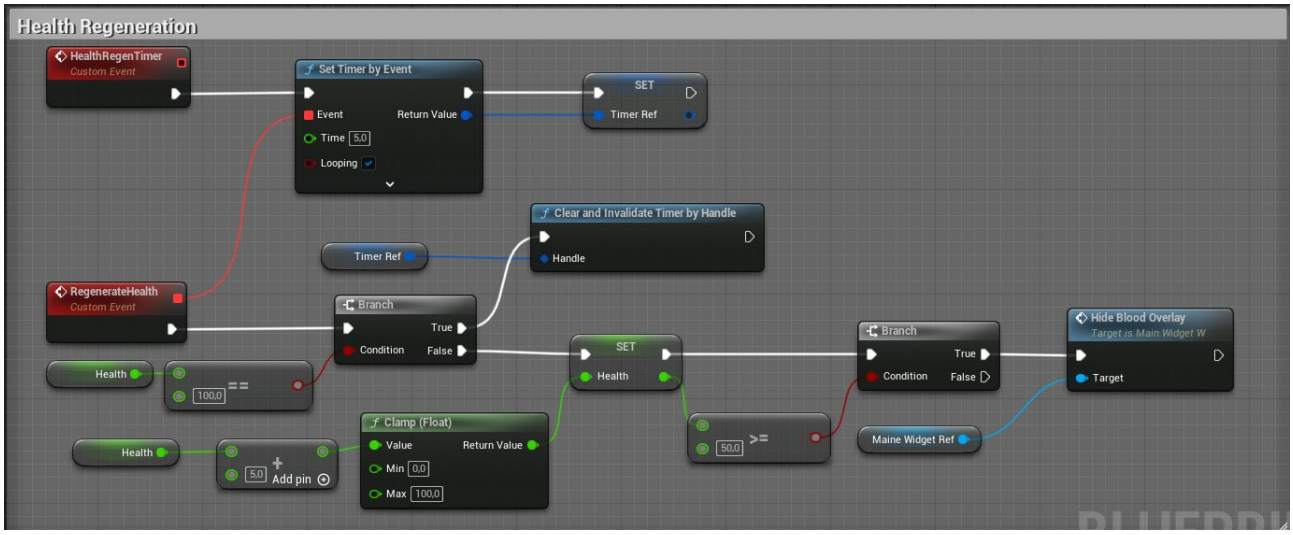


Рис. 3.5 Логіка регенерації здоровя гравця

3.3.6 Логіка отримання ушкоджень гравцем

На знімку екрана на рис. 3.6 представлено логіку отримання ушкоджень гравцем, яка активується об'єктом зомбі. Починається вона з перевірки статусу гравця: якщо він мертвий, івент зупиняється, в іншому випадку віднімається 20 одиниць здоров'я. Після цього проводиться перевірка, чи здоров'я гравця менше або дорівнює нулю. Якщо так, виконується логіка смерті гравця; якщо гравець ще живий, запускається таймер регенерації. Якщо рівень здоров'я нижче 50, екран пофарбовується червоним, як попередження про поганий стан гравця, інакше цей колір прибирається.

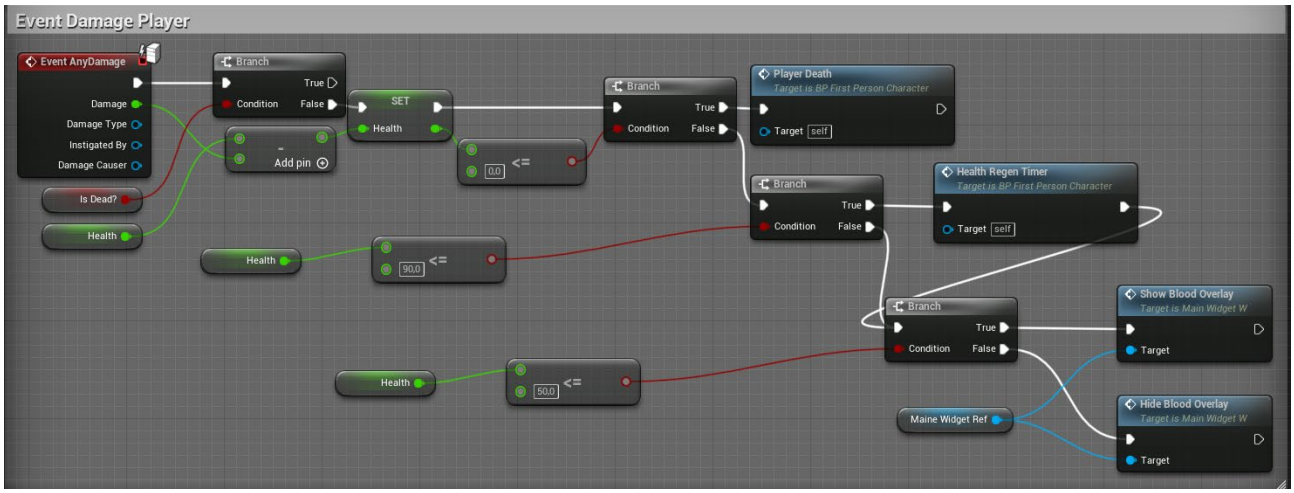


Рис. 3.6 Логіка отримання ушкоджень гравцем

3.3.7 Логіка прицілювання

На знімку екрана на рис. 3.7 показана логіка прицілювання, яка активується під час натискання правої клавіші миші. Поки ця клавіша утримується, значення змінної "True", і швидкість гравця встановлюється на 200. Коли клавішу відпускають, значення змінної змінюється на "False", а швидкість гравця знову повертається до значення 600.

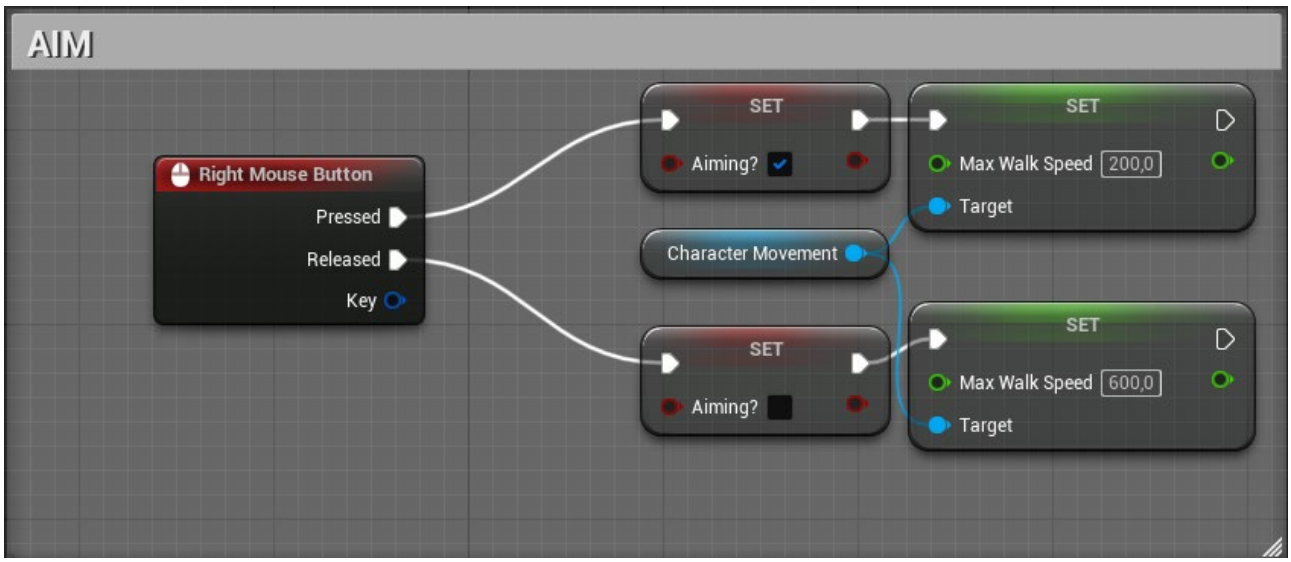


Рис. 3.7 Логіка прицілювання

3.3.8 Логіка стрільби

На знімку екрана на рис. 3.8 відображена логіка стрільби, яка активується при натисканні лівої клавіші миші. Спочатку проводиться перевірка кількості набоїв: якщо їх немає, відтворюється аудіодоріжка про осечку і на цьому івент закінчується; якщо ж набої є, віднімається один. Потім виконується перевірка значення змінної, отриманої з логіки прицілювання, від якої залежить вибір монтажу та анімації для відтворення. Після цього виконання коду переходить до логіки трасування лінії.

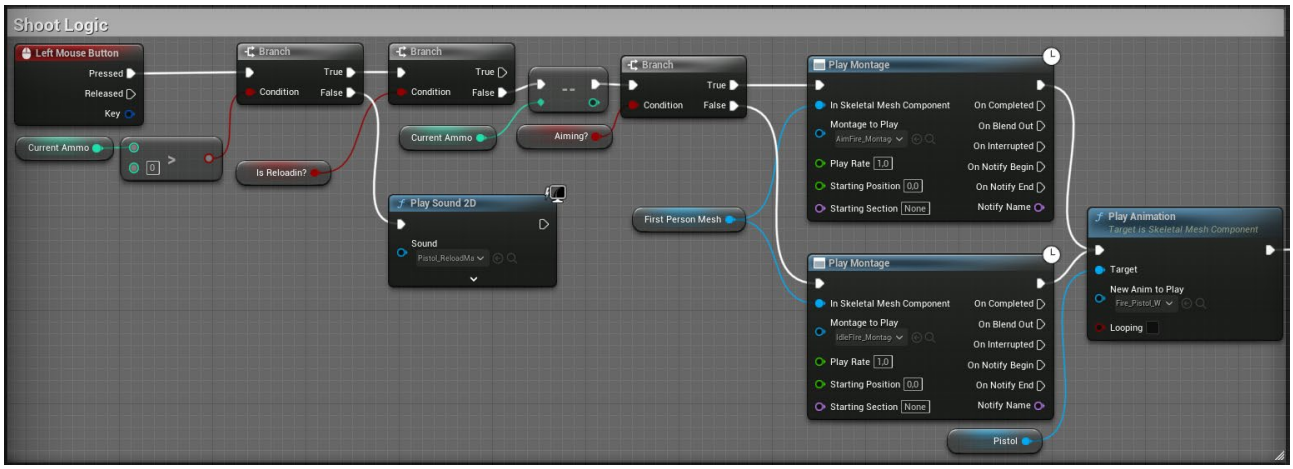


Рис. 3.8 Логіка стрільби

3.3.9 Логіка трасування лінії

На знімку екрана на рис. 3.9 представлена логіка трасування лінії, що виконується після завершення логіки стрільби та заміняє кулю у грі. Після завершення логіки стрільби виконується логіка трасування лінії, яка використовує значення положення камери та його напрямок. На основі цих даних проводяться розрахунки, і створюється пряма лінія, яка вказує у напрямку, куди дивиться гравець, але яку він не бачить. Якщо ця лінія зіштовхується з об'єктом, вона активує логіку нанесення ушкоджень.

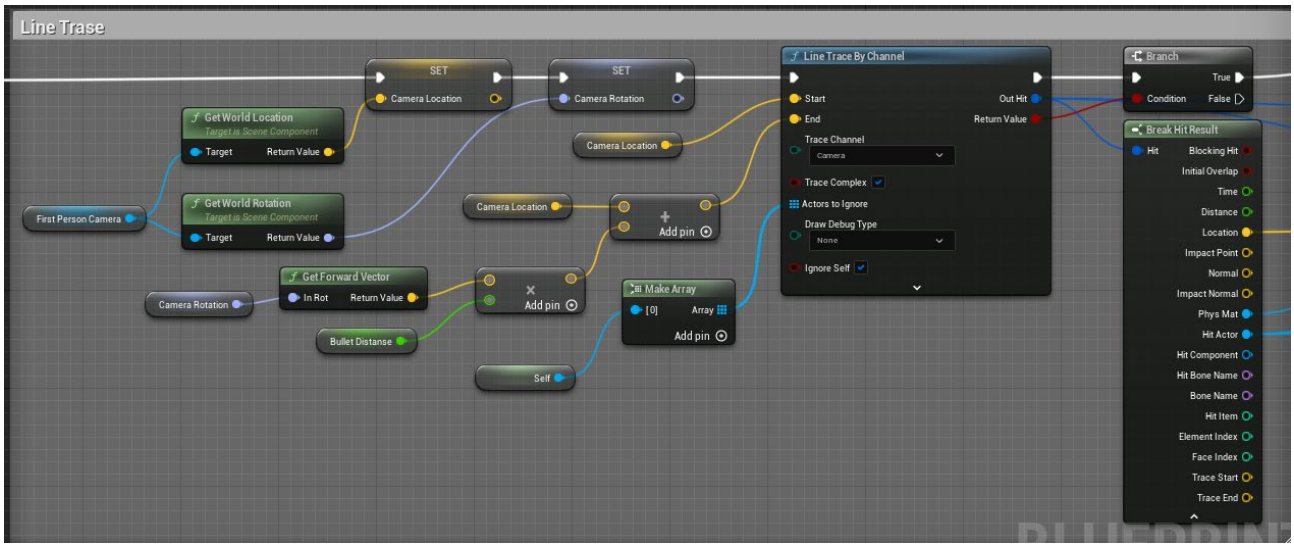


Рис. 3.9 Логіка трасування лінії

3.3.10 Логіка нанесення ушкоджень

На знімку екрана на рис. 3.10 показана логіка нанесення ушкоджень, що виконується після логіки трасування лінії. Ця логіка відповідає за нанесення 50 одиниць ушкоджень, якщо куля потрапляє в голову зомбі, 20 одиниць ушкоджень, якщо в інші частини тіла зомбі, а також не наносить ушкоджень, якщо куля потрапляє в інші об'єкти гри. Спочатку відбувається перевірка того, в який об'єкт потрапила лінія: якщо це зомбі, відтворюються спецефекти

виливання крові, а якщо це інший об'єкт, відтворюються ефекти розлітаючихся частинок, а також створюється візуальний ефект ушкодження від кулі.

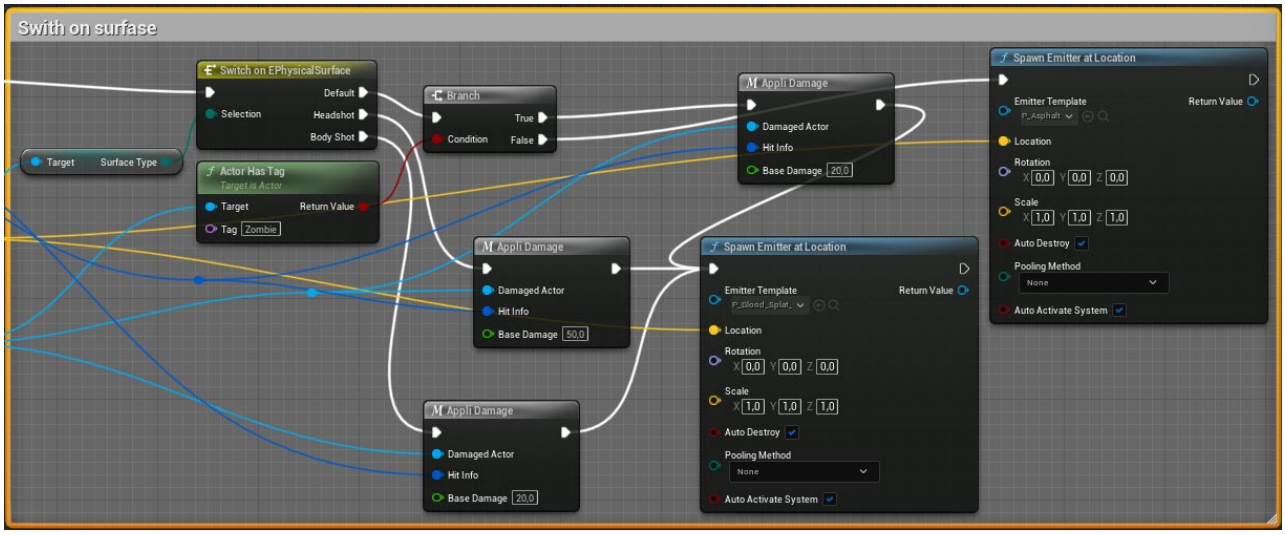


Рис. 3.10 Логіка нанесення ушкоджень

3.3.11 Логіка руху зомбі

На знімку екрана на рис. 3.11 представлена логіка руху зомбі, що активується після початку гри. Вона викликає логіку прискорення зомбі.

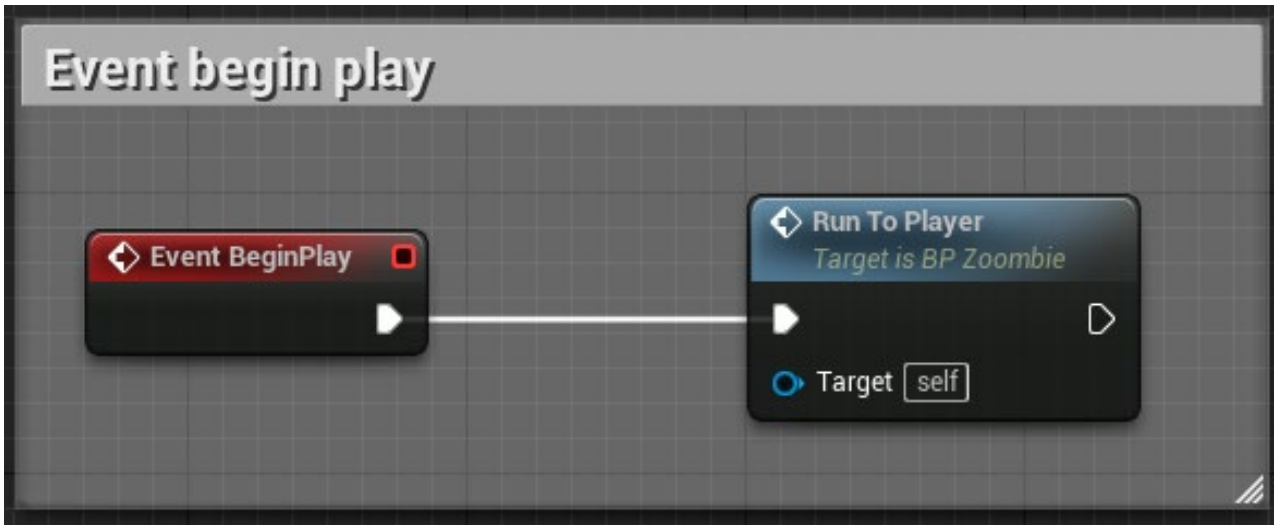


Рис. 3.11 Логіка руху зомбі

3.3.12 Логіка прискорення зомбі за умови що гравець поруч

На знімку екрана на рис. 3.12 представлена логіка прискорення зомбі у випадку, коли гравець знаходиться поруч. Ця логіка активується, якщо відстань між зомбі і гравцем стає меншою за певне значення, вказане в класі зомбі. Після цього швидкість зомбі збільшується до 360.

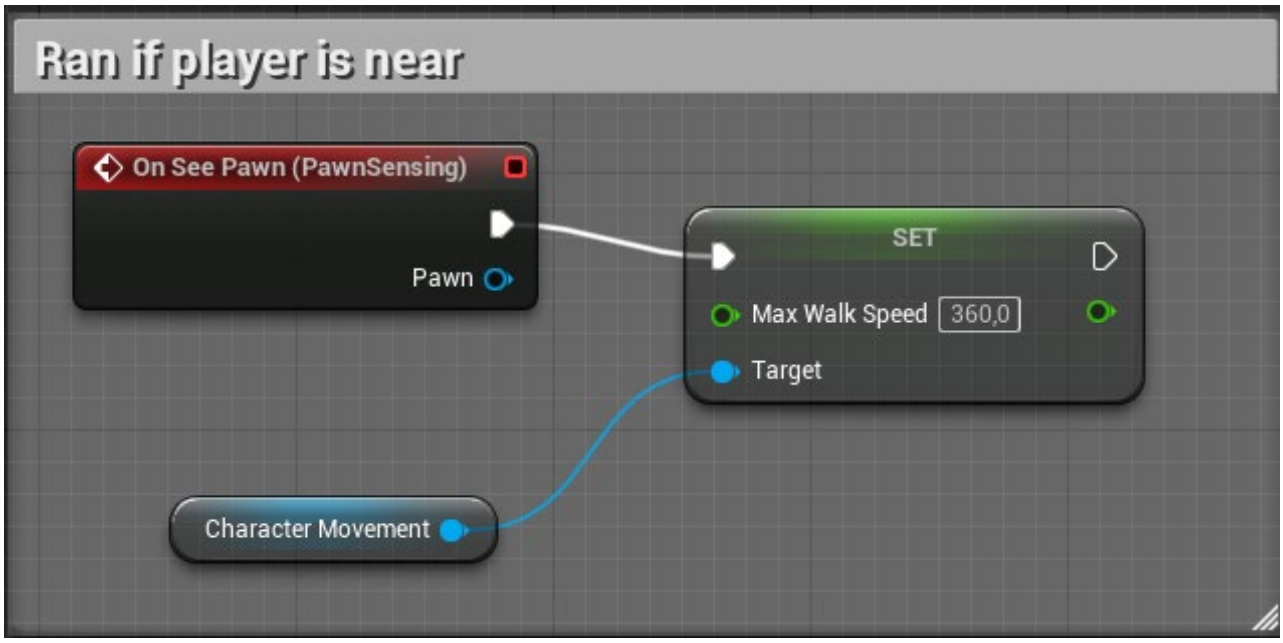


Рис. 3.12 Логіка прискорення зомбі за умови що гравець поруч

3.3.13 Логіка наздоганяння гравця

На знімку екрана на рис. 3.13 відображена логіка наздоганяння гравця. По-перше, ця функція перевіряє, чи зомбі ще живий. Якщо так, то штучний інтелект направляє його рухатись в напрямку гравця. Після цього перевіряється, чи вже зомбі наздогнав гравця. Якщо так, викликається логіка атакування гравця. Якщо ні, функція знову викликає себе, щоб зомбі продовжив наздоганяти гравця.

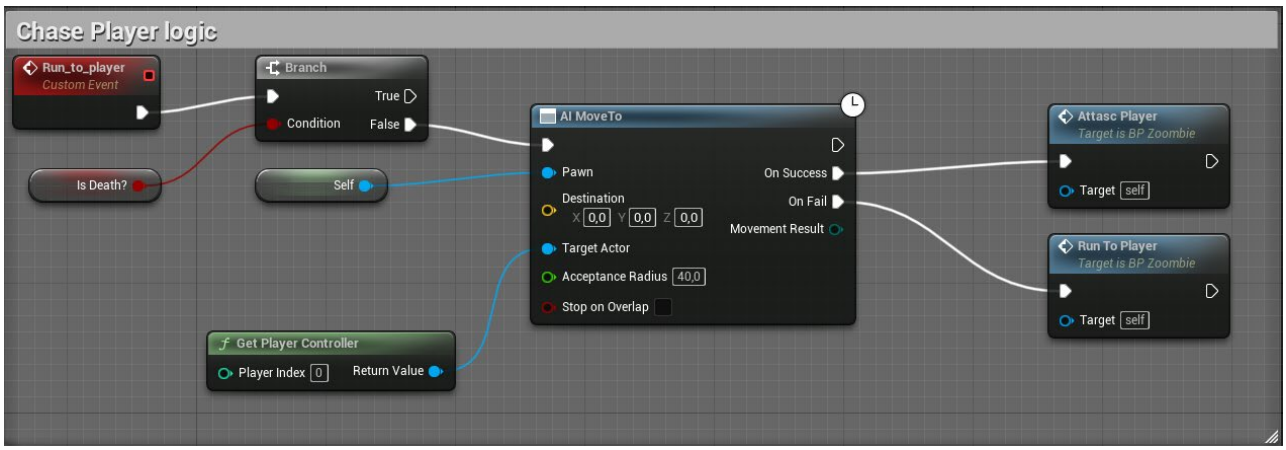


Рис. 3.13 Логіка наздоганяння гравця

3.3.14 Логіка атакування гравця

На знімку екрана на рис. 3.14 відображена логіка атакування гравця. Відразу починається відтворення монтажу атаки та звукової доріжки атаки. Після цього обчислюється, які об'єкти зазнали впливу сфери, що створюється під час атаки. Після визначення потерпілих об'єктів ця сфера видаляється, щоб не займати зайву пам'ять. Якщо гравець опиняється в цій сфері, йому завдаються ушкодження на 20 одиниць здоров'я. Після цього, якщо гравець вдається втекти або уникнути атаки, викликається логіка наздоганяння.

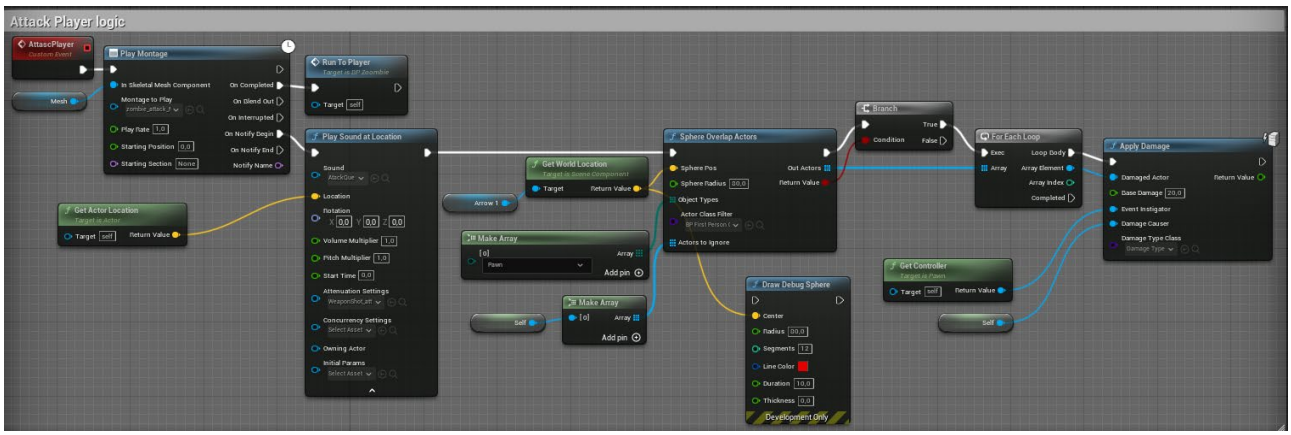


Рис. 3.14 Логіка атакування гравця

3.3.15 Логіка отримання ушкоджень зомбі

На знімку екрана на рис. 3.15 показана логіка отримання ушкоджень зомбі. Після виклику перевіряється, чи рівень здоров'я зомбі менше або дорівнює нулю. Якщо це так, викликається логіка смерті зомбі. Якщо ні, здоров'я зомбі зменшується на кількість отриманих ушкоджень. Після цього знову перевіряється, чи ще живий зомбі. Після цього відтворюється монтаж.

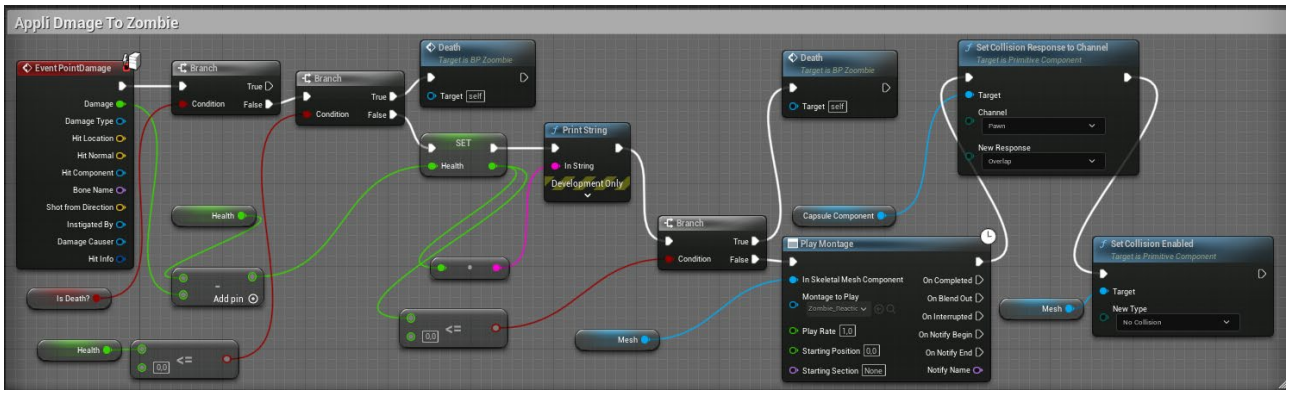


Рис. 3.15 Логіка отримання ушкоджень зомбі

3.3.16 Логіка смерті зомбі

На знімку екрана на рис. 3.16 показана логіка смерті зомбі, яка активується, коли у зомбі залишається 0 або менше одиниць здоров'я. Після виклику цієї логіки змінна, яка вказує на те, чи зомбі вже мертвий, змінюється на True. Після цього зупиняється рух зомбі, відтворюється монтаж смерті зомбі, прибирається колізія, і викликається логіка перевірки, чи всі зомбі в хвилі мертві. Після цього затримкою в 10 секунд актор зомбі видаляється.

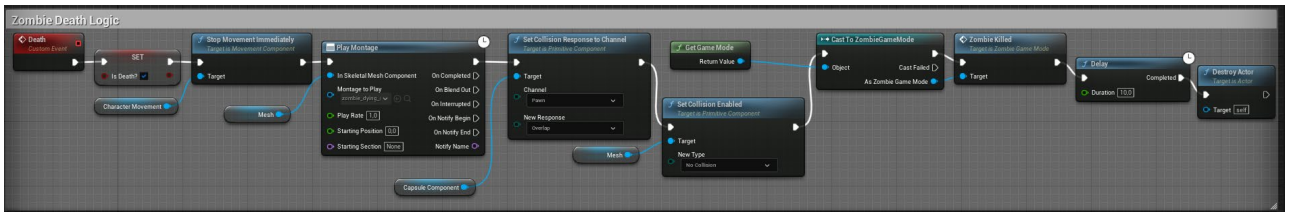


Рис. 3.16 Логіка смерті зомбі

3.3.17 Логіка початку гри

На знімку екрана на рис. 3.17 показана логіка початку гри, яка активується після натискання кнопки "Почати гру" в головному меню. Ця логіка викликає івент "Почати гру", який, у свою чергу, викликає логіку спавну зомбі. Таким чином, при натисканні кнопки гравцем гра починається, і зомбі починають спавнитися.

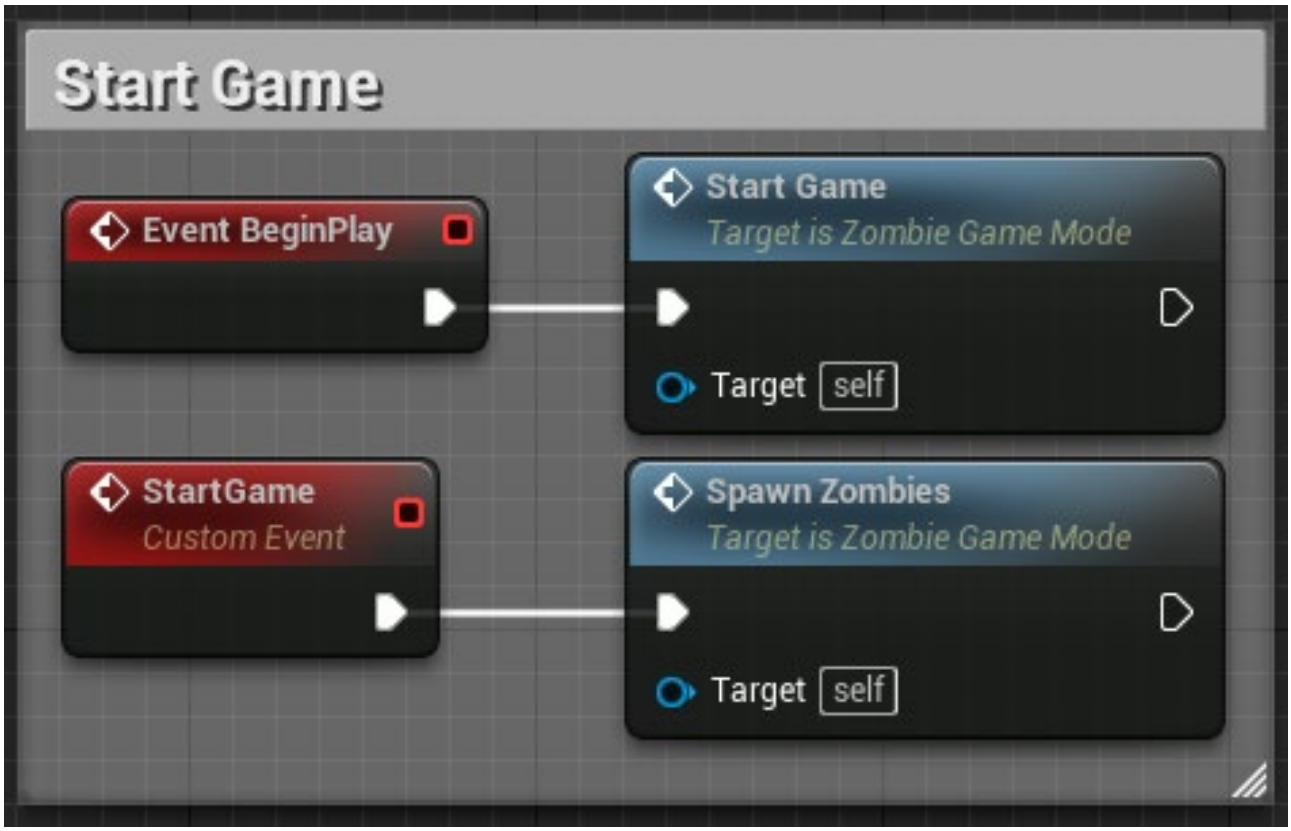


Рис. 3.17 Логіка початку гри

3.3.18 Логіка початку наступної хвилі

На знімку екрана на рис. 3.18 відображена логіка початку наступної хвилі. Ця логіка викликається, коли всі зомбі з попередньої хвилі мертві. Після цього вона показує гравцеві анімований напис про початок наступної хвилі. Після виклику логіки спавну зомбі спеціально затримують на 5 секунд, щоб наступна хвиля почалася після цієї затримки.

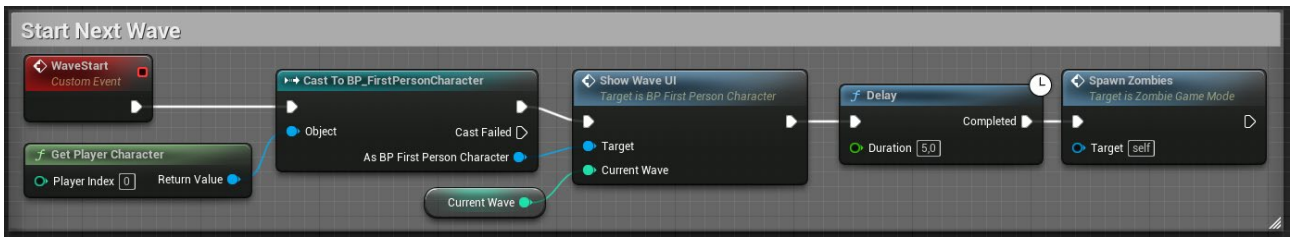


Рис. 3.18 Логіка початку наступної хвилі

3.3.19 Логіка спавну зомбі

На знімку екрана на рис. 3.19 показана логіка спавну зомбі. Після її виклику активується логіка спавну ящиків з набоями. Потім номер хвилі множиться на 2, щоб визначити кількість зомбі в хвилі. Після цього отримане число ділиться на 2, і на кожній спавн-точці з двох сторін спавниться по половині хвилі зомбі.

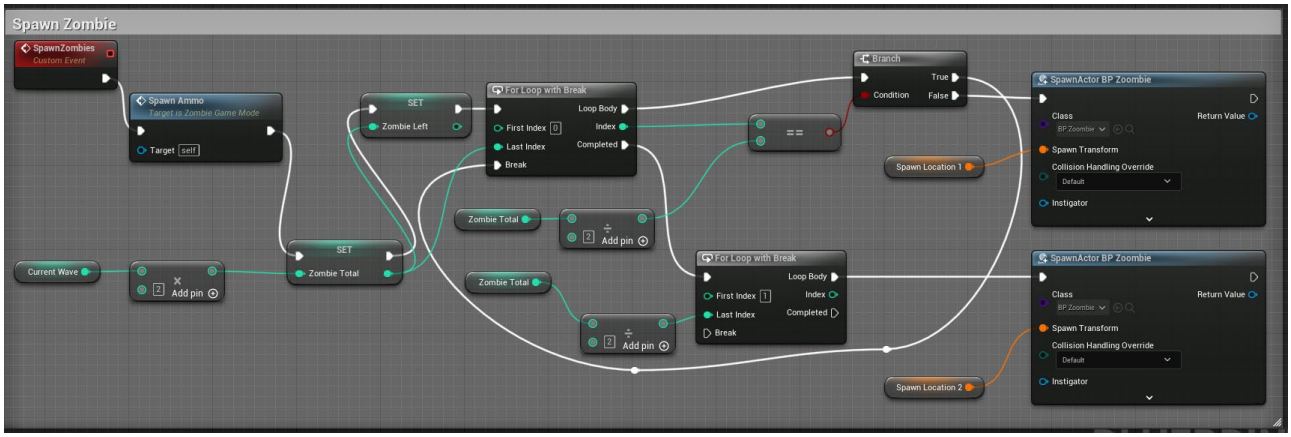


Рис. 3.19 Логіка спавну зомбі

3.3.20 Логіка перевірка чи всі зомбі в хвилі мертві

На знімку екрана на рис. 3.20 показана логіка перевірки, чи всі зомбі в хвилі мертві. Після виклику вона порівнює кількість живих зомбі з нулем. Якщо кількість живих зомбі дорівнює нулю, викликається логіка зміни хвилі.

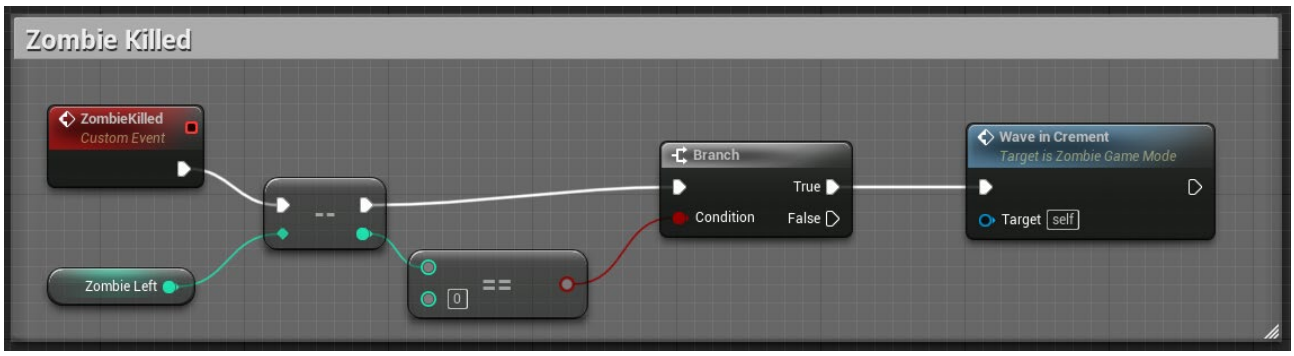


Рис. 3.20 Логіка перевірка чи всі зомбі в хвилі мертві

3.3.21 Логіка зміни хвилі

На знімку екрана на рис. 3.21 показана логіка зміни хвилі. Після виклику ця логіка перевіряє номер поточної хвилі і максимальну хвилю. Оскільки максимальна хвиля дорівнює 999, гра працює в псевдо нескінченному режимі. Потім до номера теперішньої хвилі додається одиниця, і викликається логіка початку наступної хвилі.

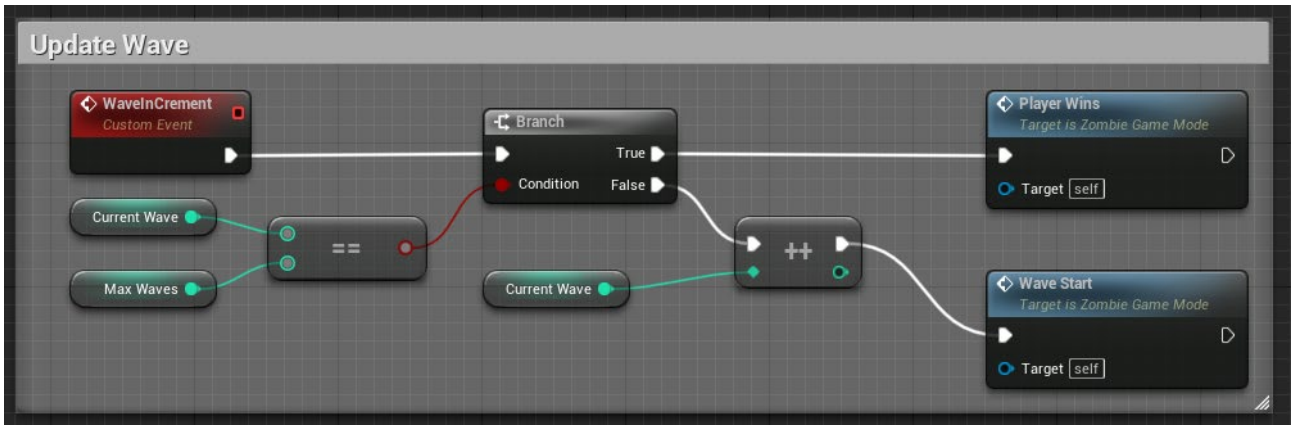


Рис. 3.21 Логіка зміни хвилі

3.3.22 Логіка спавну ящиків з набоями

На знімку екрана на рис. 3.22 показана логіка спавну ящиків з набоями. Після виклику ця логіка спавнить об'єкти на мапі, які надають гравцеві набої.

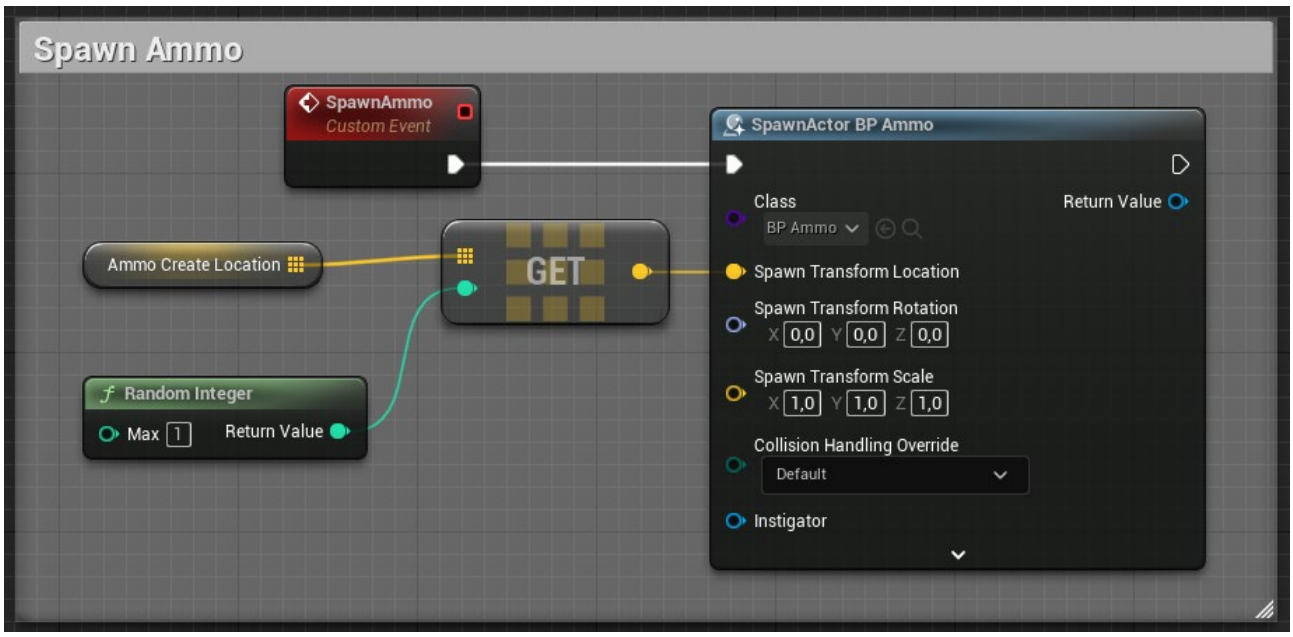


Рис. 3.22 Логіка спавну ящиків з набоями

3.4 Реалізація коду C++ на основі Blueprint прототипу

На рисунку 3.23 буде представлено графічне зображення, що відображає структуру класів, пов'язаних із ігровими механіками. Ця діаграма буде включати основні класи, які забезпечують основний функціонал у грі. Вона послужить важливим інструментом для розуміння внутрішньої логіки та взаємодії ключових компонентів у грі. Відображення цих класів допоможе з'ясувати, як різні частини програмного забезпечення гри взаємодіють між собою, сприяючи створенню цілісного геймплею.

Ці класи разом утворюють основу для гри з виживанням проти зомбі, де гравець (представлений класом MyCharacter) має боротися проти хвиль зомбі (керованих класами ZombieCharacteriZombieGameMode). Клас ZombieCharacter.

AZombieCharacter() - Конструктор класу.

BeginPlay() - Викликається при початку гри для ініціалізації.

Tick(DeltaTime: float) - Викликається кожного кадру для оновлення стану зомбі.

OnSeePawn(Pawn: APawn*) - Викликається, коли зомбі бачить гравця (або іншу ціль).

RunToPlayer() - Змушує зомбі бігти до гравця.

AttackPlayer() - Викликається для атаки гравця.

ApplyDamageToZombie(Damage, InstigatedBy, DamageCauser) - Застосовує пошкодження до зомбі.

HandleDeath() - Викликається, коли зомбі помирає.

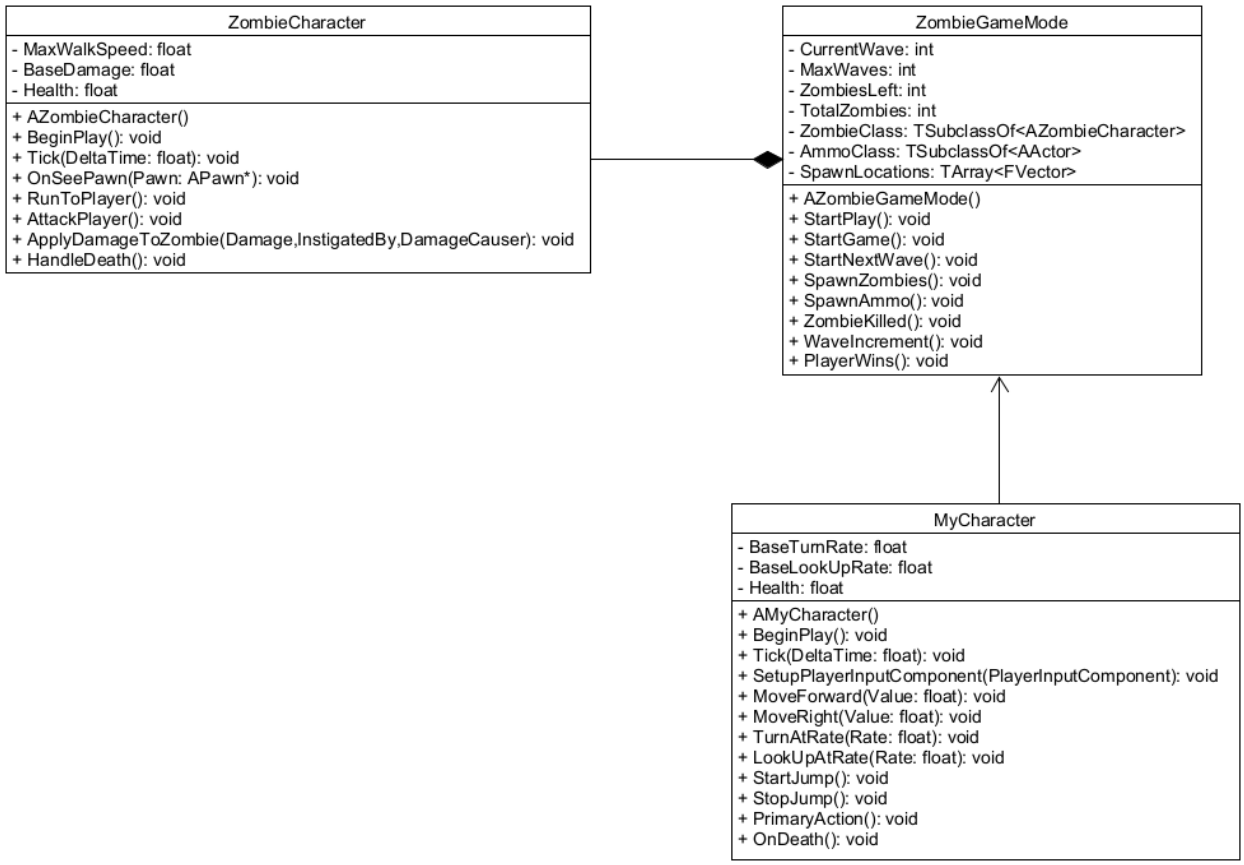


Рис. 3.23 Діаграма класів

Клас ZombieGameMode

AZombieGameMode() - Конструктор класу.

StartPlay() - Викликається при початку гри для налаштування ігрового режиму.

StartGame() - Починає гру.

StartNextWave() - Починає наступну хвилю зомбі.

SpawnZombies() - Спавнить (створює) зомбі.

SpawnAmmo() - Спавнить боєприпаси.

ZombieKilled() - Викликається, коли зомбі вбито.

WaveIncrement() - Збільшує лічильник хвиль.

PlayerWins() - Викликається, коли гравець виграє.

Клас MyCharacter

AMyCharacter() - Конструктор класу.

BeginPlay() - Викликається при початку гри для ініціалізації.

Tick(DeltaTime: float) - Викликається кожного кадру для оновлення стану гравця.

SetupPlayerInputComponent(PlayerInputComponent) - Налаштовує компоненти для введення гравця.

MoveForward(Value: float) - Рух вперед.

MoveRight(Value: float) - Рух вправо.

TurnAtRate(Rate: float) - Поворот з заданою швидкістю.

LookUpAtRate(Rate: float) - Підйом погляду з заданою швидкістю.

StartJump() - Початок стрибка.

StopJump() - Зупинка стрибка.

PrimaryAction() - Основна дія гравця.

OnDeath() - Викликається, коли гравець помирає.

4 ТЕСТУВАННЯ ІГРОВОГО ЗАСТОСУНКУ

4.1 Особливості тестування ігор шутерів

Тестування зомбі шутерів має свої власні особливості порівняно з іншими типами тестування в галузі програмного забезпечення. Ці особливості пов'язані з унікальними елементами гри, її ігровим середовищем, характером противників та різними платформами, на яких гра може бути доступна. Це означає, що Тестувальник повинен ретельно перевірити всі можливі сценарії під час гри особливо механіку стрільби, зокрема точність, реакцію на вогонь, рух та вплив різних збройних систем. Важливо також протестувати ворогів у грі та чі правильно вони взаємодіють з гравцем.

4.2 Тестові сценарії гри “ INFECTED ”

Для тестування застосунку використано функціональне тестування, яке виконано методом мануального тестування. Для тестування було обрано наступні функції:

Тестові сценарії та результати їх виконання наведено в таблиці 4.1.

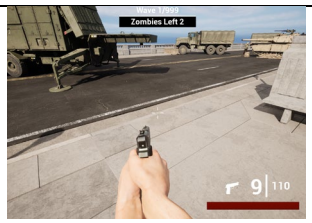
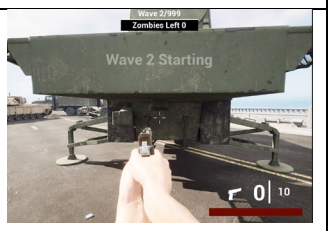
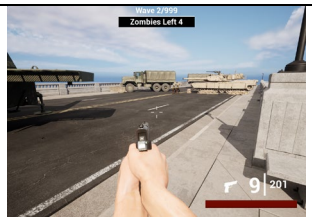
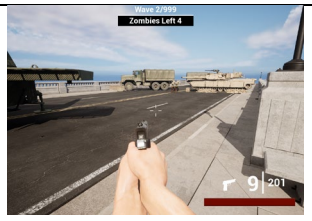
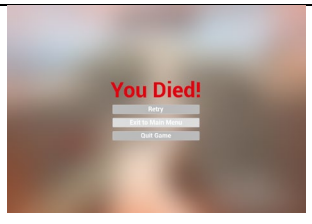
Таблиця 4.1

Тестові сценарії гри

№	Дії	Очікуваний результат	Фактичний результат
1	Отримання пошкоджень які знизять рівень здоров’я до 50% і менше від ворогів.	Вмикається кровавий підсвіт.	

Продовження таблиці 4.1

Тестові сценарії гри

№	Дії	Очікуваний результат	Фактичний результат
2	Зробити п’ять пострілів в зомбі.	Рівень здоров’я знизиться до нуля і спрацює анімація смерті ворога.	
3	Підбирання ящика з набоями.	Запас набоїв збільшиться на 100 одиниць.	
4	Проходження хвилі.	Після проходження хвилі з’являється напис про початок наступної.	
5	Вбити всіх зомбі у хвилі.	Лічильник кількості зомбі оновить свої данні	
6	Пережити хвилю зомбі.	Кількість зомбі в наступній хвилі збільшиться.	
7	Гравець втрачає все здоров’я.	З’являється вікно смерті.	

Всі тест-кейси пройдено успішно, гра працює коректно.

ВИСНОВКИ

1. Проведено аналіз відеоігор в жанрі зомбі-шутер. Визначено тенденції, популярні механіки та уподобання гравців в жанрі зомбі-шутер. На основі аналізу аналогічних ігор, таких як DayZ та Left 4 Dead, визначено ключові вимоги до гри INFECTED.

2. Проведено аналіз можливостей ігрового рушія Unreal Engine при створенні ігор в жанрі зомбі-шутер, зокрема, використання Blueprint дозволило спростити процес написання програмного коду, роботу з анімаціями та ефектами.

4. Розроблено концепт гри INFECTED та спроектовано її геймплей в жанрі зомбі-шутер, який, зокрема, передбачає нескінчену гру та появу зомбі хвилями, диференційовану систему уражень ворогів в залежності від зони попадання кулі.

5. Розроблено файлову структуру проекту гри INFECTED, спроектовано класи застосунку мовою C++.

6. Розроблено програмне забезпечення гри INFECTED. Для створення основних та додаткових механік гри, роботи з анімаціями та спецефектами використано Blueprint. C++ використано для реалізації класу персонажу.

7. Проведено функціональне тестування гри INFECTED методом мануального тестування. Всі тест-кейси завершено з позитивним результатом.

Робота пройшла апробацію на двох наукових конференціях, за результатами апробації опубліковано тези доповідей:

1. Крицький В.І., Треньов М.Г. Функціональність візуального програмування в Unreal Engine: Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ». Збірник тез. 24.04.2024, ДУІКТ, м. Київ. К.: ДУІКТ, 2024. С. 16.
2. Крицький В.І., Треньов М.Г. Сучасні методи розробки ігор з використанням ігрового рушія Unreal Engine: Матеріали Всеукраїнської Науково-практичної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. 15.05.2024, ДУІКТ, м. Київ. К.: ДУІКТ, 2024. С. 10.

ПЕРЕЛІК ПОСИЛАНЬ

1. Крицький В.І., Треньов М.Г. Функціональність візуального програмування в Unreal Engine: Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ». Збірник тез. 24.04.2024, ДУІКТ, м. Київ. К.: ДУІКТ, 2024. С. 16.
2. Крицький В.І., Треньов М.Г. Сучасні методи розробки ігор з використанням ігрового рушія Unreal Engine: Матеріали Всеукраїнської Науково-практичної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. 15.05.2024, ДУІКТ, м. Київ. К.: ДУІКТ, 2024. С. 10.
3. Game Programming Patterns / Robert Nystrom та ін. USA: Genever Benning, 2014. 354 с.
4. Level Up! The Guide to Great Video Game Design / Scott Rogers та ін. West Sussex: Wiley, 2014. 552 с.
5. Unreal Engine 4 Game Development in 24 Hours, Sams Teach Yourself / Aram Cookson, Ryan DowlingSoka, Clinton Crumpler та ін. Indianapolis: Sams Publishing, 2016. 432 с.
6. Unreal Engine 4 Game Development Cookbook / William Sherif, Stephen Whittle та ін. Birmingham: Packt Publishing, 2015. 468 с.
7. Unreal Engine 4 for Design Visualization: Developing Stunning Interactive Visualizations, Animations, and Renderings / Tom Shannon та ін. Birmingham: Packt Publishing, 2017. 558 с.
8. Blueprints Visual Scripting for Unreal Engine / Brenden Sewell, Marcos Romero та ін. Birmingham: Packt Publishing, 2015. 340 с.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Створення гри в жанрі Зомбі-шутер на базі рушія Unreal Engine

Виконав студент 4 курсу
групи ПД-41
Крицький Владислав Ігорович
Керівник роботи
Асистент кафедри ІПЗ, Треньов Микита Георгійович

Київ – 2024

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** – збільшення зацікавленості ігровим процесом в грі жанру "Зомбі-шутер" під платформу Windows за рахунок використання нескінченного геймплею і орієнтації на рекордний результат.
- **Об’єкт дослідження** – ігровий процес в грі жанру зомбі-шутер під платформу Windows.
- **Предмет дослідження** – гра в жанрі зомбі-шутер під платформу Windows з нескінченним ігровим процесом та орієнтацією на рекордний результат.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести огляд та аналіз існуючих ігор жанру зомбі-шутер.
2. Провести аналіз можливостей ігрового рушія Unreal Engine при створенні ігор в жанрі зомбі-шутер.
3. Розробити концепт гри INFECTED та спроектувати її геймплей в жанрі зомбі-шутер.
4. Виконати підбір елементів ігрового світу для гри INFECTED на існуючих платформах.
5. Розробити файлову структуру проекту гри INFECTED, спроектувати класи застосунку.
6. Розробити програмне забезпечення гри INFECTED.
7. Провести тестування гри INFECTED.

3

АНАЛІЗ АНАЛОГІВ

Характеристика гри	Left 4 Dead	DayZ	INFECTED
Режим гри	Кооперативний	Онлайн	Офлайн
Розмір на жорсткому диску	10 ГБ	20 ГБ	5 ГБ
Кількість гравців	1-4	1-60	1
Графіка	3D	3D	3D
Формат геймплею	Кооперативні кампанії	Відкритий світ	Проходження на рекорд, геймплей нескінченний

4

КОНЦЕПТ ГРИ

- Гравець знаходиться в світі де зомбі ніколи не закінчуються але гравець може померти.
- Протягом гри гравець б'ється з ворогами і тому може втрачати здоров'я але якщо він не буде отримувати ушкодження впродовж 5 секунд він почне відновлювати рівень здоров'я.Також якщо під час гри гравець підійде занадто близько до ворогів вони почнуть бігти швидше.
- Після кожної хвилі на мапі генерується ящик з комплектом набоїв які гравець може підібрати.
- Гра закінчується коли гравець помирає.

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги

1. Можливість переміщення персонажа, стрільби та прицілювання.
2. Реалізувати механіку нескінченної кількості хвиль.
3. Забезпечити графіку, анімації та візуальні ефекти в стилі зомбі апокаліпсису.
4. Реалізувати механіку підбирання набоїв з ящиків що з'являються на мапі після кожної хвилі.

Нефункціональні вимоги

1. Гра повинна працювати на операційній системі Windows.
2. Рухи всіх персонажів та об'єктів, з якими може взаємодіяти гравець, мають бути анімованими.
3. У грі мають бути присутній звукові ефекти.

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

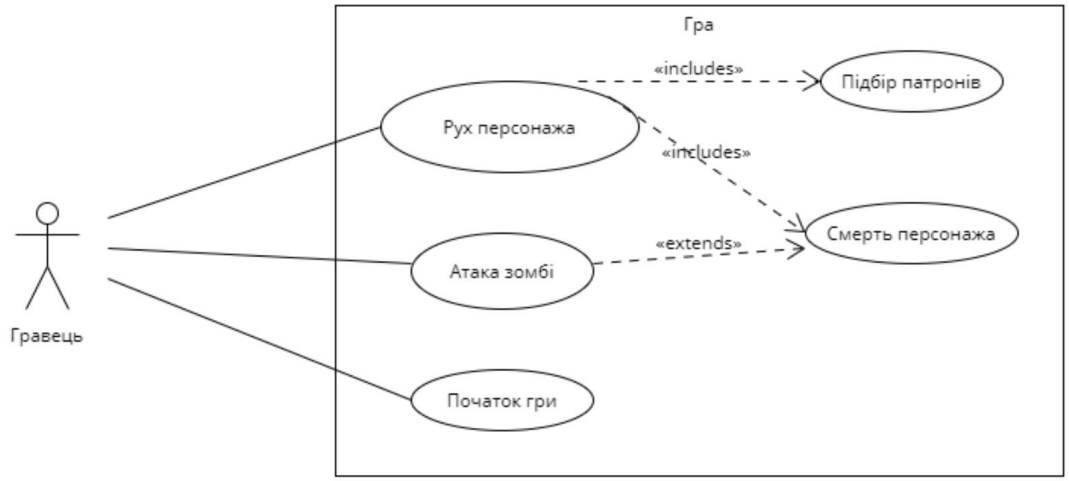
Unreal Engine

Visual Studio

C++

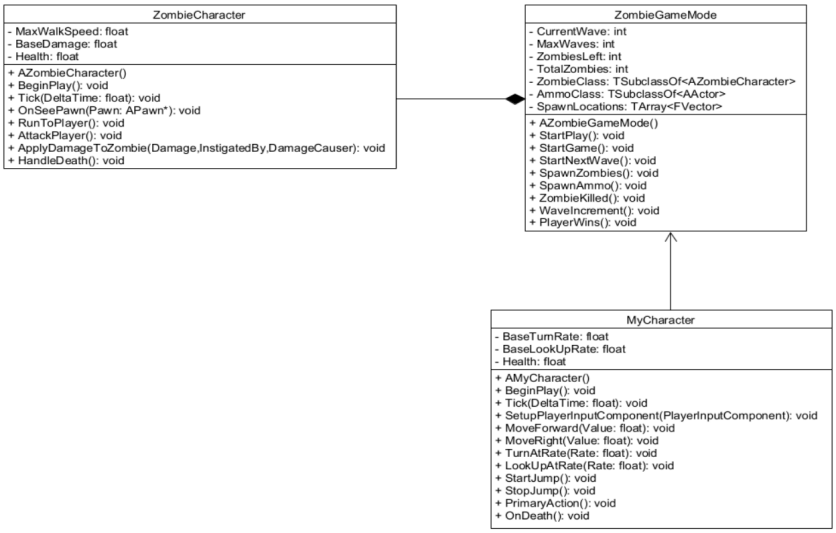
7

USE-CASE ДІАГРАМА



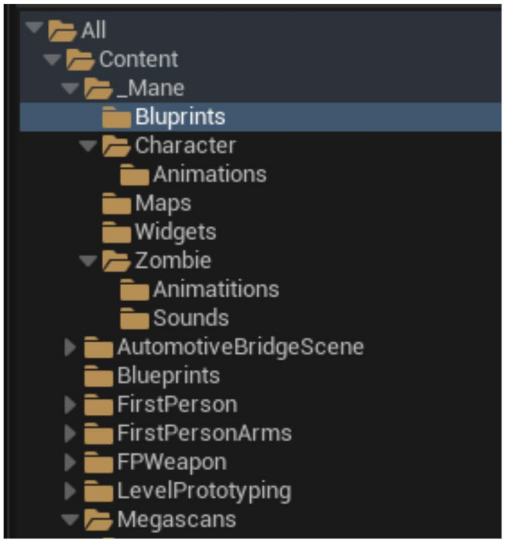
8

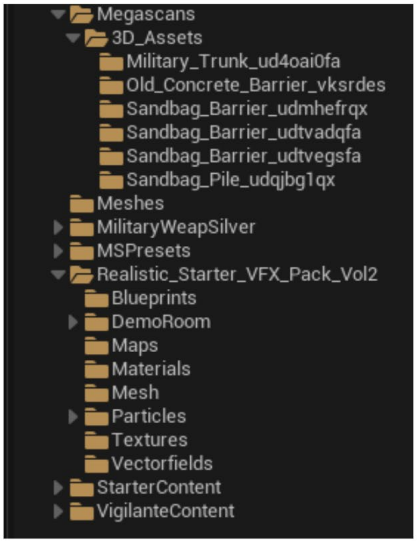
ДІАГРАМА КЛАСІВ



9

ФАЙЛОВА СТРУКТУРА ПРОЕКТУ





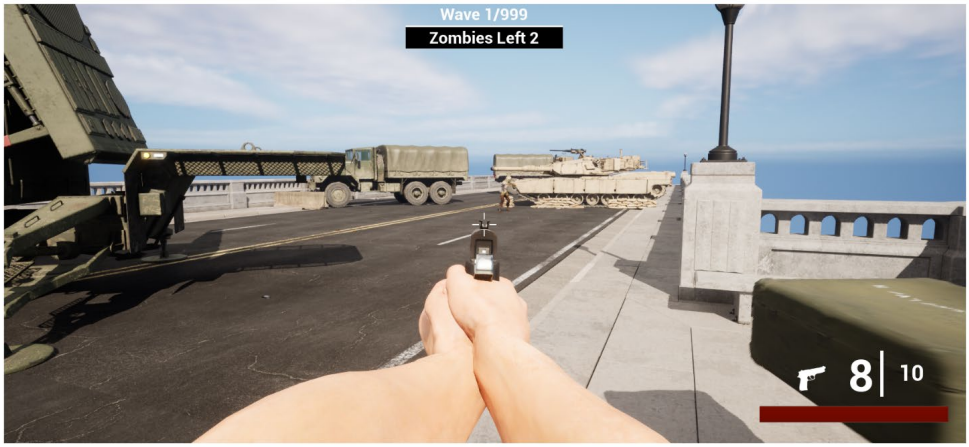
10

ЕКРАННІ ФОРМИ

Головне меню

11

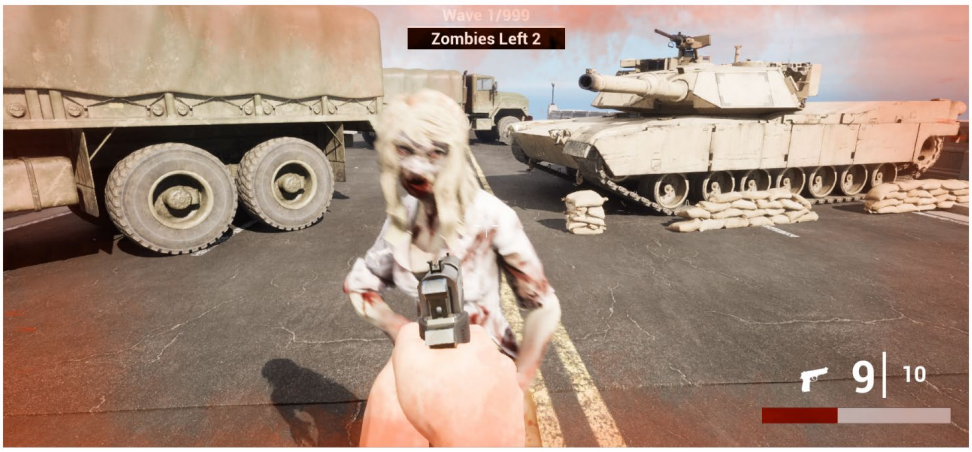
ЕКРАННІ ФОРМИ



Екран гри

12

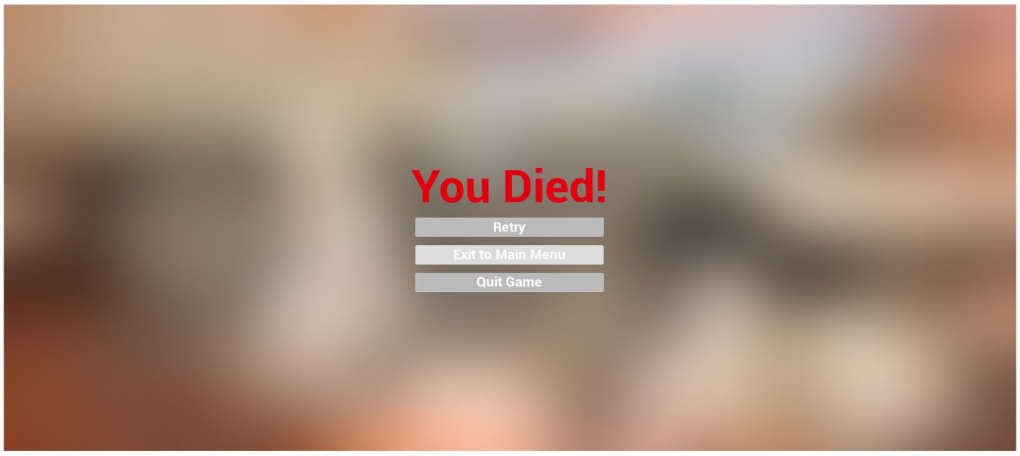
ЕКРАННІ ФОРМИ



Екран під час отримання ушкоджень

13

ЕКРАННІ ФОРМИ



Екран програшу

14

ДЕМОНСТРАЦІЯ ГРИ

15

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Крицький В.І., Треньов М.Г. Функціональність візуального програмування в Unreal Engine: Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ». Збірник тез. 24.04.2024, ДУІКТ, м. Київ. К.: ДУІКТ, 2024. С. 16.
2. Крицький В.І., Треньов М.Г. Сучасні методи розробки ігор з використанням ігрового рушія Unreal Engine: Матеріали Всеукраїнської Науково-практичної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. 15.05.2024, ДУІКТ, м. Київ. К.: ДУІКТ, 2024. С. 10.

16

ВИСНОВКИ

1. Проведено аналіз відеоігор в жанрі зомбі-шутер. Визначено тенденції, популярні механіки та уподобання гравців в жанрі зомбі-шутер. На основі аналізу аналогічних ігор, таких як DayZ та Left 4 Dead, визначено ключові вимоги до гри INFECTED.
2. Проведено аналіз можливостей ігрового рушія Unreal Engine при створенні ігор в жанрі зомбі-шутер, зокрема, використання Blueprint дозволило спростити процес написання програмного коду, роботу з анімаціями та ефектами.
4. Розроблено концепт гри INFECTED та спроектовано її геймплей в жанрі зомбі-шутер, який, зокрема, передбачає нескінчену гру та появу зомбі хвилями, диференційовану систему уражень ворогів в залежності від зони попадання кулі.
5. Розроблено файлову структуру проекту гри INFECTED, спроектовано класи застосунку мовою C++.
6. Розроблено програмне забезпечення гри INFECTED. Для створення основних та додаткових механік гри, роботи з анімаціями та спецефектами використано Blueprint. C++ використано для реалізації класу персонажу.
7. Проведено функціональне тестування гри INFECTED методом мануального тестування. Всі тест-кейси завершено з позитивним результатом.

ДОДАТОК В. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
// MyCharacter.h
#pragma once

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

UCLASS()
class YOURGAME_API AMyCharacter : public ACharacter
{
   GENERATED_BODY()

public:
   AMyCharacter();

protected:
   virtual void BeginPlay() override;

public:
   virtual void Tick(float DeltaTime) override;
   virtual void SetupPlayerInputComponent(class UInputComponent*
PlayerInputComponent) override;

   UFUNCTION()
   void MoveForward(float Value);

   UFUNCTION()
   void MoveRight(float Value);
```

```
    UFUNCTION()
    void TurnAtRate(float Rate);

    UFUNCTION()
    void LookUpAtRate(float Rate);

    UFUNCTION()
    void StartJump();

    UFUNCTION()
    void StopJump();

    UFUNCTION()
    void PrimaryAction();

    UFUNCTION()
    void OnDeath();

private:
    UPROPERTY(EditAnywhere, Category="Movement")
    float BaseTurnRate;

    UPROPERTY(EditAnywhere, Category="Movement")
    float BaseLookUpRate;

    UPROPERTY(EditAnywhere, Category="Combat")
    float Health;
};
// MyCharacter.cpp
```

```
#include "MyCharacter.h"
#include "GameFramework/Controller.h"
#include "GameFramework/SpringArmComponent.h"
#include "Camera/CameraComponent.h"
#include "Kismet/GameplayStatics.h"

AMyCharacter::AMyCharacter()
{
   PrimaryActorTick.bCanEverTick = true;

   BaseTurnRate = 45.f;
   BaseLookUpRate = 45.f;
   Health = 100.f;
}

void AMyCharacter::BeginPlay()
{
   Super::BeginPlay();
}

void AMyCharacter::Tick(float DeltaTime)
{
   Super::Tick(DeltaTime);
}

void AMyCharacter::SetupPlayerInputComponent(UInputComponent*
PlayerInputComponent)
{
   Super::SetupPlayerInputComponent(PlayerInputComponent);
```

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

    PlayerInputComponent->BindAxis("Turn", this,
&APawn::AddControllerYawInput);
    PlayerInputComponent->BindAxis("TurnRate", this,
&AMyCharacter::TurnAtRate);
    PlayerInputComponent->BindAxis("LookUp", this,
&APawn::AddControllerPitchInput);
    PlayerInputComponent->BindAxis("LookUpRate", this,
&AMyCharacter::LookUpAtRate);

    PlayerInputComponent->BindAction("Jump", IE_Pressed, this,
&AMyCharacter::StartJump);
    PlayerInputComponent->BindAction("Jump", IE_Released, this,
&AMyCharacter::StopJump);

    PlayerInputComponent->BindAction("PrimaryAction", IE_Pressed, this,
&AMyCharacter::PrimaryAction);
}

void AMyCharacter::MoveForward(float Value)
{
    if ((Controller != nullptr) && (Value != 0.0f))
    {
        const FRotator Rotation = Controller->GetControlRotation();
        const FRotator YawRotation(0, Rotation.Yaw, 0);
```

```
        const FVector Direction =
FRotationMatrix(YawRotation).GetUnitAxis(EAxis::X);
        AddMovementInput(Direction, Value);
    }
}

void AMyCharacter::MoveRight(float Value)
{
    if ( (Controller != nullptr) && (Value != 0.0f) )
    {
        const FRotator Rotation = Controller->GetControlRotation();
        const FRotator YawRotation(0, Rotation.Yaw, 0);

        const FVector Direction =
FRotationMatrix(YawRotation).GetUnitAxis(EAxis::Y);
        AddMovementInput(Direction, Value);
    }
}

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

void AMyCharacter::LookUpAtRate(float Rate)
{
    AddControllerPitchInput(Rate * BaseLookUpRate * GetWorld()-
>GetDeltaSeconds());
}
```

```
void AMyCharacter::StartJump()
{
   bPressedJump = true;
}

void AMyCharacter::StopJump()
{
   bPressedJump = false;
}

void AMyCharacter::PrimaryAction()
{
   UE_LOG(LogTemp, Warning, TEXT("Character has died."));
   UGameplayStatics::SpawnEmitterAtLocation(GetWorld(), DeathParticle,
GetActorLocation());
   Destroy();
}

void AMyCharacter::OnDeath()
{
   Health -= DamageAmount;
   if (Health <= 0)
   {
      OnDeath();
   }
}
```