МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА 3D ГРИ З ВИКОРИСТАННЯМ ІГРОВОГО РУШІЯ UNREAL ENGINE 5»**

Виконав: студент 4 курсу, групи 6.1221
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
Д.В. Смішко
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, к.т.н., доцент Борю. С.Ю.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А. О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., професор

_____ Шило Г.М.
(підпис)

" ____ " _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Смішку Данііілу Валерійовичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка 3D гри з використанням ігрового рушія Unreal Engine 5
керівник роботи Борю Сергій Юрійович, доцент, к.т.н.
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від « 20 » грудня 2024 року № 2001-с

2. Строк подання студентом роботи 02.06.2025
3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка 3D гри з використанням ігрового рушія Unreal Engine 5
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 20.12.2024

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	21.12.2024	
2.	Збір вихідних даних.	10.01.2025	
3.	Обробка методичних та теоретичних	03.02.2025	
	джерел.		
4.	Розробка першого та другого розділу.	17.03.2025	
5.	Розробка третього розділу.	13.04.2025	
6.	Оформлення та нормоконтроль	02.06.2025	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	18.06.2025	

Студент _____ Д.В. Смішко
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.Ю. Борю
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка 3D гри з використанням ігрового рушія Unreal Engine 5»: 51 с., 8 рис., 1 табл., 15 джерел.

БЛЮПРІНТИ, ВІЗУАЛІЗАЦІЯ, ГЕЙМПЛЕЙ, ІГРОВИЙ ПРОЦЕС, ІГРОВИЙ РУШІЙ, ІНТЕРАКТИВНІСТЬ, РЕАЛІСТИЧНА ГРАФІКА, РОЗРОБКА ІГОР, 3D ГРА, UNREAL ENGINE 5.

Об’єкт дослідження – процес створення тривимірних ігор із використанням сучасних ігрових рушіїв.

Мета роботи – розробити прототип 3D гри з використанням рушія Unreal Engine 5, що реалізує основні механіки геймплею, забезпечує високу якість візуалізації, інтуїтивну взаємодію з ігровим середовищем та відповідає сучасним вимогам до розробки ігор.

Метод дослідження – експериментальний, аналітичний.

В результаті розроблена 3D гра з використанням ігрового рушія Unreal Engine 5, яка може бути використована як технічна демонстрація, навчальний матеріал або основа для комерційного проєкту.

SUMMARY

Bachelor’s Qualification Theses "Development of a 3D Game Using the Unreal Engine 5 Game Engine": 51 pages, 8 figures, 1 tables, 15 references.

BLUEPRINTS, VISUALIZATION, GAMEPLAY, GAME PROCESS, GAME ENGINE, INTERACTIVITY, REALISTIC GRAPHICS, GAME DEVELOPMENT, 3D GAME, UNREAL ENGINE 5.

Object of research – the process of creating three-dimensional games using modern game engines.

Purpose of the work – to develop a prototype of a 3D game using Unreal Engine 5 that implements core gameplay mechanics, provides high-quality visualization, intuitive interaction with the game environment, and meets modern game development standards.

Research method – experimental, analytical.

As a result, a 3D game was developed using the Unreal Engine 5 game engine, which can be used as a technical demonstration, educational material, or the foundation for a commercial project.

ЗМІСТ

Завданн яна кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 8
1 Інформаційний огляд 9
1.1 Особливості методів розробки та створення 3D комп'ютерних ігор .. 9
1.2 Актуальність розробки 3D ігор з використанням сучасних рушіїв 9
1.3 Аналіз технологій, що використовуються при створенні 3D ігор на Unreal Engine 5 10
1.4 Постановка задачі 11
2 Теоритична підготовка до реалізації 3D гри з використанням ігрового рушія Unreal Engine 5 13
2.1 Аналіз задачі та створення плану розробки 13
2.2 Аналіз предметної області 14
2.3 Вибір стеку технологій для розробки проєкту 16
2.4 Налаштування оточення у проєкті 17
2.6 Розробка система прогресії 20
2.7 Побудова структури рівнів 22
2.8 Розробка AI та його поведінки 23
2.9 Побудова інтерфейсу користувача UMG 25
3 Програмна реалізація 28
3.1 Вивчення архітектури та логіки реалізації геймплею 28
3.2 Технічна підготовка середовища проєкту 30
3.3 Налаштування оточення у проэкті 30
3.4 Побудова базової логіки гри 32
3.5 Розробка системи прогресії 35
3.6 Побудова структури рівнів 38
3.7 Реалізація AI та бойової поведінки 40

3.8 Побудова інтерфейсу користувача UMG 42
3.9 Тестування, оптимізація, фінальна збірка 43
Висновки 47
Перелік посилань 49
Додаток А Посилання на скомпільовану версію гри51 51

ВСТУП

У сучасному світі індустрія відеоігор стрімко розвивається та займає провідні позиції серед цифрових технологій і розваг. Зростання вимог користувачів до графічної якості, реалістичної візуалізації та інтерактивності стимулює постійне вдосконалення інструментів розробки. Одним із найпотужніших ігрових рушіїв сьогодення є Unreal Engine 5 [1], який забезпечує розробникам доступ до широкого набору функцій для створення високоякісного контенту.

Метою цієї роботи є створення 3D-гри в жанрі Souls-like [2] із використанням Unreal Engine 5, яка поєднує сучасну візуалізацію, зручну систему взаємодії з ігровим середовищем і ключові механіки геймплею. У процесі розробки активно використовуються інструменти рушія для реалізації реалістичної графіки, фізики, аудіо-ефектів та оптимізації продуктивності, що дозволяє досягти високого рівня якості.

У ході виконання роботи аналізується архітектура гри, визначаються її функціональні можливості та реалізується прототип із базовим ігровим циклом. Особливу увагу приділено створенню ігрової логіки за допомогою системи Blueprints, яка дає змогу реалізувати складні сценарії без використання великої кількості коду.

Розробка демонструє практичне застосування сучасних інструментів створення ігор та є прикладом реалізації повноцінного 3D-проєкту, що відповідає актуальним стандартам галузі. Отримані результати можуть бути корисними для розробників-початківців і студентів, які прагнуть опанувати створення інтерактивного програмного продукту на базі сучасного ігрового рушія.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Особливості методів розробки та створення 3D комп’ютерних ігор

Розробка 3D комп’ютерних ігор є одним з найбільш складних напрямів сучасної цифрової індустрії. Ці ігри зазвичай включають складні ігрові сценарії, штучний інтелект, систему фізики, механіки взаємодії з навколишнім середовищем, а також велику кількість мультимедійного контенту.

Кожен з компонентів вимагає окремого проєктування, реалізації та тестування. Крім того, супровід, оновлення, балансування гри після релізу є тривалим і ресурсомістким процесом. Саме тому створення повноцінної 3D-гри потребує значних витрат часу, технічних знань і фінансів.

У зв’язку з цим виникає потреба в автоматизації основних етапів розробки. Одним з ефективних рішень є використання ігрових рушіїв – програмних середовищ, які дозволяють описувати логіку гри за допомогою спеціалізованих мов програмування або візуального середовища. Такі рушії після виконання ряду нескладних дій дозволяють отримати програму, яка реалізує заданий сценарій гри.

Подібний підхід значно скорочує строки розробки, полегшує реалізацію складних механік та знижує поріг входу для новачків.

1.2 Актуальність розробки 3D ігор з використанням сучасних рушіїв

Індустрія відеоігор є однією з найдинамічніших галузей цифрових технологій. Особливо активно зростає сегмент 3D ігор, які забезпечують занурення користувача в інтерактивне віртуальне середовище завдяки фотореалістичній графіці, реалістичним ефектам і динамічному геймплею.

Типові сценарії 3D ігор включають: відкриті світи (open world), PvE-ігри (бої з навколишнім середовищем або NPC), PvP-баталії, стелс-ігри, платформери, survival horror, а також ігри з глибоким наративом або

розгалуженою системою квестів. Їх реалізація передбачає великий обсяг роботи з геймдизайну, анімації, графіки та програмування.

Особливу увагу останніми роками отримав жанр Souls-like, що поєднує елементи RPG, хардкорного бою, прокачування персонажа та дослідження складних рівнів. Аналіз подібного жанру дозволяє виявити ключові геймплейні механіки, що можуть бути адаптовані до власного проєкту.

Сучасні ігрові рушії, як-от Unreal Engine 5, надають розробникам інструменти, що дозволяють задовольнити високі вимоги користувачів до графіки, фізики об’єктів, ШІ та зручності керування .

Технології Nanite [3] (віртуалізована геометрія) та Lumen [4] (глобальне освітлення в реальному часі) дозволяють створювати високодеталізовані сцени без значного зниження продуктивності. Рушій підтримує як роботу з мовою програмування C++ [5], так і з візуальним середовищем Blueprints [6], що дозволяє значно спростити реалізацію ігрової логіки.

UE5 підтримується активною спільнотою, яка станом на 2024 рік налічує понад 13 млн користувачів [7], включаючи інді-розробників, великі студії та університети.

Недоліки рушія UE5: високі системні вимоги до обладнання, складність освоєння для новачків при роботі з C++, потреба у великому обсязі оперативної пам’яті та відеопам’яті для розгортання складних сцен.

Таким чином, використання Unreal Engine 5 для створення 3D-гри є актуальним, практичним та професійно орієнтованим завданням.

1.3 Аналіз технологій, що використовуються при створенні 3D ігор на Unreal Engine 5

Unreal Engine 5 – це потужний багатофункціональний рушій для розробки ігор, візуалізацій, симуляцій та інтерактивних додатків. Його архітектура є модульною, що дозволяє гнучко керувати компонентами проєкту: візуалізацією, фізикою, анімацією, логікою поведінки тощо.

Ключові технології рушія:

– Nanite – віртуалізована геометрія для відображення мільйонів полігонів без втрати продуктивності;
– Lumen – глобальне освітлення в реальному часі з підтримкою динамічних тіней і відбиттів;
– Blueprints – візуальна мова програмування, яка дозволяє створювати логіку без коду;
– підтримка C++ для створення кастомної логіки;
– вбудовані інструменти для фізики, звуку, колізій, VR/AR, мережевої взаємодії та багатоплатформеного експорту;
– інтеграція з Blender, Maya, 3ds Max без додаткової конверсії;
– інструменти Profiler, Session Frontend, Automation Testing Framework для оптимізації та тестування.

Типові знання, необхідні розробнику при використанні UE5:

– основи 3D-графіки та моделювання;
– принципи роботи ігрових рушіїв;
– ООП і основи програмування (C++, Blueprints);
– основи анімації, логіки бою, побудови рівнів;
– UI/UX у відеоіграх;
– навички оптимізації продуктивності.

Unreal Engine 5 автоматизує процес генерації, компіляції та збирання гри як у разі використання C++, так і Blueprints. Після написання логіки проєкт можна зібрати (Build) в повноцінний виконуваний файл для цільової платформи (Windows, Android, iOS тощо). Це включає генерацію *.exe, *.pak, *.ucas, *.utoc та інших службових файлів, що формують готовий продукт.

Такий підхід дозволяє реалізувати повний життєвий цикл розробки – від ідеї до створення, налагодження, компіляції та експорту білду, який можна передати кінцевому користувачу.

1.4 Постановка задачі

Мета роботи – розробка 3D-гри жанру Souls-like з використанням ігрового рушія Unreal Engine 5 з акцентом на візуальне програмування.

Для досягнення мети необхідно вирішити наступні задачі:

– аналіз предметної області та виявлення особливостей жанру Souls-like для формування відповідного геймплею;
– виявлкння практичних можливостей рушія UE5 та принципів візуального програмування сценаріїв;
– розробка технічного проєкту з урахуванням можливостей рушія, використовуючи виключно Blueprints;
– створення архітектури гри (логіка гравця, ворогів, бойова система, переміщення між рівнями);
– формування інформаційної моделі (дані про персонажів, предмети, характеристики, ефекти);
– розробка інтерфейсів (UI): інвентар, характеристики, екіпірування, механіки прокачування;
– реалізація алгоритму проходження гри, включаючи однонапрямлену навігацію, процедурну генерацію ресурсів та ландшафту, логіку фінального бою.

2 ТЕОРИТИЧНА ПІДГОТОВКА ДО РЕАЛІЗАЦІЇ 3D ГРИ З ВИКОРИСТАННЯМ ІГРОВОГО РУШІЯ UNREAL ENGINE 5

2.1 Аналіз задачі та створення плану розробки

Ця робота передбачає створення 3D гри жанру Souls-like з використанням ігрового рушія Unreal Engine 5, з фокусом на побудовііігрової логіки виключно через Blueprint-сценарії без використання C++ або MetaHuman.

Основна мета полягає в реалізації ключових механік, притаманних іграм даного жанру, у стислому прототипі, який демонструє геймплейну петлю.

Для успішної реалізації проєкту було визначено наступний етапний план розробки, що відображає послідовність виконання задач:

а) аналіз предметної області:
   - дослідження жанру Souls-like, його типових механік та очікувань гравця;
   - вивчення геймплейних циклів: дослідження, бої, збереження, прогресія;
   - аналіз прикладів ігор: Dark Souls, Lies of P [8], The Witcher 3 [9];

б) технічна підготовка:
   - вибір стеку технологій;
   - вивчення можливостей стеку технологій;

в) налаштування оточення проекту:
   - імпорт базових ресурсів: Quixel Bridge, Sketchfab (FAB) [10];
   - створення тестової сцени та 2 рівня з базовим ландшафтом;
   - підключення PCG Framework для процедурної генерації;

г) побудова базової логіки гри:
   - створення контролера персонажа (рух, камера, взаємодія);
   - реалізація анімаційної системи (ходьба, біг, атаки, перекати);
   - налаштування combat-системи (урони, блок, AI);

д) система прогресії:

- проєктування інвентаря, екіпірування, витратників;
- реалізація системи характеристик персонажа (сила, витривалість, тощо);
- меню прокачування біля вогнища;

е) побудова структури рівнів:
- підземелля з ворогами, патрулюванням та переходами;
- відкритий світ на ландшафті (зібрані ресурси, крафт);
- бос-арена;
- система точок збереження;

ж) реалізація AI та бойової поведінки:
- використання Behavior Tree і Blackboard;
- додавання AI Perception для виявлення гравця;
- поведінка боса;

з) побудова інтерфейсу UMG:
- інвентар, характеристики, екіпірування;
- повідомлення про події, смерті, підказки;
- екран завантаження при зміні рівнів;

и) тестування, оптимізація, фінальна збірка:
- перевірка стабільності логіки;
- відлагодження переходів, боїв та прокачування;
- створення фінальної збірки гри для демонстрації.

2.2 Аналіз предметної області

Жанр Souls-like, на основі якого розробляється цей проєкт, є піджанром екшен-RPG, що відомий своєю високою складністю, тактичним бойовим стилем, системою збереження через контрольні точки, а також глибокою системою прогресії та дослідженням світу.

Вибір саме цього жанру зумовлений кількома факторами. По-перше, у пункті 1.4 було зазначено, що метою проєкту є реалізація ігрової петлі на основі

бою, дослідження і розвитку персонажа – елементів, які органічно поєднані саме в Souls-like іграх. По-друге, жанр фентезі, який є основою більшості Souls-like проєктів, близький мені тематично та естетично, а отже, викликає щиру зацікавленість у глибшому його вивченні. Крім того, мені було цікаво дослідити, як професійні розробники втілюють складні механіки цього жанру – від бойової системи до структури світу.

Ключовими рисами Souls-like є:

– методичні бої з ворогами – бої вимагають від гравця уважності до таймінгів, грамотного використання витривалості, ухилень і блоків;
– поступове дослідження світу – локації пов’язані між собою переходами, секретами та лінійним прогресом через зони;
– контрольні точки збереження – гравець зберігає прогрес у спеціальних зонах, де також може покращити характеристики;
– покрокова рольова система розвитку – досвід, отриманий у боях, витрачається на покращення параметрів персонажа (сила, витривалість тощо).

Запропонована гра реалізується у вигляді стислого прототипу, адаптованого під можливості Blueprint-розробки. Проєкт включає три рівні, які демонструють основні елементи Souls-like циклу:

– Підземелля з ворогами – стартова зона з лінійною побудовою, де гравець тренує основи бою;
– Відкрита зона – місцевість без ворогів, де гравець збирає ресурси для крафту і знаходить перехід до наступного рівня;
– Бос-арена – заключний рівень з боєм проти сильного супротивника, який перевіряє навички гравця.

Акцент у грі зроблено на базових елементах: бою, дослідженні, прокачуванні та зборі ресурсів. Це дозволяє зосередитися на демонстрації ігрової петлі у компактному форматі та досягти чіткого результату в межах навчального проєкту.

Під час вивчення предметної області було проаналізовано низку ключових проєктів. Dark Souls став зразком побудови локацій, бойової системи та загальної

атмосфери. Lies of P надихнув візуальним стилем і структурою боїв. Також були враховані окремі елементи з The Witcher 3, зокрема система збирання ресурсів і дослідження відкритого світу.

2.3 Вибір стеку технологій для розробки проєкту

У процесі створення гри було використано низку інструментів і технологій, доступних в Unreal Engine 5.5, які забезпечують повноцінну реалізацію геймплейної логіки, візуальної складової, а також процедурних і інтерфейсних систем. Усі елементи гри реалізовані без використання C++ – виключно засобами Blueprint Visual Scripting.

Нижче наведено перелік основних технологій, які використовувалися в розробці, із зазначенням їх призначення в проєкті (табл. 2.1)

Таблиця 2.1 – Ключові технології розробки

№	Технологія	Призначення	Використання в проєкті
1	Blueprint Visual Scripting	Візуальне програмування логіки гри	Рух, атаки, блок, взаємодія, UI, AI, бойова система, переходи між рівнями
2	Animation Blueprint, Blend Space, State Machines	Анімація персонажів і ворогів з плавними переходами	Ходьба, біг, атаки, перекати, анімації смерті та взаємодій
3	AI Controller та Behavior Tree	Побудова логіки ворогів	Патрулювання, переслідування, атака, втрата гравця

Продовження таблиці 2.1

4	UMG (Unreal Motion Graphics)	Побудова користувацького інтерфейсу	Інвентар, характеристики, UI екіпірування, повідомлення, екран завантаження

5	Level Streaming та Trigger Volumes	Управління підрівнями та подіями	Завантаження нових рівнів, відкриття воріт, активація переходів
6	Procedural Content Generation (PCG)	Генерація об’єктів ландшафту	Автоматичне розміщення дерев, ресурсів, елементів середовища
7	Landscape Tools	Створення відкритого світу	Рельєф, текстурування, навігація
8	Quixel Bridge та Fab	Імпорт 3D-моделей та матеріалів	Дерева, кущі, каміння, архітектурні елементи, текстури

2.4 Налаштування оточення у проєкті

У межах підготовчого етапу було визначено базову архітектуру ігрового середовища, яке включає дві ключові сцени з різною функціональністю:

– тестова сцена – призначена для оперативної перевірки основних ігрових механік (рух, бойова система, UI);
– відкрита сцена – слугує основою для реалізації процедурно згенерованого ігрового світу з підтримкою масштабування, дослідження та взаємодії гравця з середовищем.

Для побудови візуального оточення було заплановано використання готових 3D-активів із платформ Quixel Bridge і FAB (Sketchfab + ArtStation), що дозволяє уникнути витрат на ручне моделювання й водночас зберегти високу якість графіки.

Акценти розробки робляться на:

– природних об’єктах: дерева, кущі, каміння;
– архітектурних елементах: руїни, огорожі, брами;
– текстурах для різних типів поверхні: скелі, мох, ґрунт.

Окрему увагу приділено плануванню впровадження PCG Framework (Procedural Content Generation) – інструменту Unreal Engine, що забезпечує автоматичне, але контрольоване розміщення об’єктів у просторі. Це рішення було обрано для:

– динамічного формування природного середовища з урахуванням рельєфу;
– гнучкої конфігурації структури рівня під різні ігрові сценарії;
– пришвидшення побудови рівня при збереженні візуального реалізму.

Таким чином, було сформовано концепцію побудови ігрового світу, який поєднує ручне опрацювання ключових зон із автоматизованим генеруванням ландшафту. Це забезпечує як швидкий старт розробки, так і перспективу масштабування в рамках відкритого ігрового світу.

2.5 Побудова базової логіки гри

На цьому етапі була реалізована основна логіка ігрового процесу, що включає переміщення персонажа, бойову систему, анімації, а також базову взаємодію з оточенням. Уся логіка реалізована у Blueprint персонажа, заснованого на стандартному класі ThirdPersonCharacter із шаблону від Epic Games.

Контролер персонажа та система руху:

– рух та спринт: реалізовано переміщення по осях (вперед/назад, вліво/вправо) з можливістю перемикання між ходьбою та спринтом. Логіка перемикання швидкостей вбудована в Blueprint і контролюється змінною швидкості (Max Walk Speed), яка оновлюється при натисканні відповідної клавіші;
– взаємодія з оточенням: додано систему виявлення об’єктів навколо персонажа через Multi Sphere Trace, яка визначає актори, що реалізують необхідні інтерфейси взаємодії (наприклад, предмети, NPC, інтерфейсні панелі).

Анімаційна система:

– Blend Space 1D: використовується для анімацій переміщення (ходьба, біг, спринт), оскільки проєкт реалізований з видом від третьої особи;

– перекати: додано повний набір перекатів у 8 напрямках із відповідними анімаціями;
– атаки: бойові анімації реалізовано через Animation Montage, з можливістю проведення серії атак (до 4 ударів), а також силової атаки з більшою затримкою та підвищеним уроном;
– State Machine: використовується для перемикання між станами персонажа – звичайний рух, атака, перекат, спринт, блокування, що забезпечує правильне відтворення анімацій у залежності від ситуації.

Бойова система:

– урон і колізії: система урону побудована на перевірці колізій між зброєю та ворогом. При влучанні перевіряється, чи є актор противником, після чого застосовується урон;
– типи урону: у проєкті передбачено декілька типів урону:
  1) фізичні: колючий, рублячий, дроблячий;
  2) ефекти: кровотеча, отруєння, замороження;
– блокування: звичайний фізичний урон може бути частково або повністю заблокований. Проте ефекти від урону не блокуються – вони накопичуються та активуються при досягненні порогового значення.

Компонент зброї: кожен тип зброї має власний компонент, який визначає:

– параметри колізії;
– базовий урон;
– коефіцієнт поглинання урону при блокуванні;
– тип нанесеного урону.

Загальний урон також залежить від характеристик персонажа (наприклад, сила, спритність), які гравець може покращувати протягом гри.

Система витривалості: усі дії гравця (атака, спринт, перекат) витрачають витривалість. Вона автоматично відновлюється під час бездіяльності.

Штучний інтелект ворогів:

AI Controller та Behavior Tree: логіка ворогів реалізована через Behavior Tree, який включає стани:

– патрулювання (режим спокою);

– переслідування гравця при його виявленні;
– атака;
– повернення у режим очікування при втраті цілі або її смерті.

AI Perception: використовується для виявлення гравця через зорові сенсори. Реакція на гравця запускає зміну поведінки в дереві рішень.

Бойові здібності AI: так само як гравець, AI може виконувати силову атаку з високим уроном, а також виконувати комбо-атаки.

2.6 Розробка система прогресії

Система прогресії у проєкті реалізована через набір модульних Actor Components, які інтегруються до персонажа й відповідають за управління інвентарем, екіпіруванням, крафтом та розвитком характеристик. Такий підхід забезпечує гнучкість і зручність розширення функціональності без необхідності значних змін у базових класах.

Інвентар персонажа має динамічний розмір і підтримує збереження предметів різних типів: зброї, амулетів, витратних матеріалів та ресурсів. Для полегшення взаємодії реалізовано сортування предметів за категоріями. Кожен предмет описується уніфікованою структурою, що спрощує обробку та взаємодію з різними типами об’єктів.

Структура предмета включає:

– назву, опис та іконку;
– кількість одиниць та ліміт у стеку;
– тип предмета;
– унікальний автоматично генерований ідентифікатор (ID) та можливість вручну задавати ID для конкретного класу предметів;
– модель зброї;
– рівень прокачування;
– базовий урон та бонус від характеристик персонажа;
– бонус урону за кожен рівень прокачування;

– розподіл фізичного урону за трьома типами: колючий, рублячий, дроблячий;
– кількість урону, поглинаємого при блокуванні;
– належність зброї (гравець чи ворог).

Для деталізації та гнучкого налаштування зброї використовується мастер-актор зброї, від якого створюються дочірні актори з індивідуальними параметрами, що дозволяє оптимізувати управління різними видами зброї.

Для амулетів передбачено збереження типів покращень із відповідними значеннями, а також списку ID амулетів, із якими вони несумісні.

Матеріали мають базові характеристики, що визначають їх як ресурси для крафту.

Витратні предмети описуються окремою структурою, яка включає тип витратника, перелік необхідних матеріалів для виготовлення, структуру ефектів застосування та масив додаткових пасивних ефектів.

Для реалізації логіки лута створено окремий Actor, який розміщується на сцені та має дочірні об’єкти у вигляді світла та статичних мешів.

Для реалізації логіки лута створено мастер-актор з базовою логікою. Від нього успадковуються дочірні актори лута, що представлені у 2 виглядах світла та статичних мешів, це забезпечує візуальне та функціональне відображення предметів на рівні, а також зацікавленність гравця у зборі всіх предметів.

Система екіпірування дозволяє одночасно екіпірувати:

– 2 одиниці зброї;
– 3 амулети;
– 3 витратних предмети.

У разі зайнятості слота новий предмет замінює попередній, який повертається до інвентаря.

Система крафту контролює можливість створення предметів, списує необхідні ресурси та додає виготовлені предмети до інвентаря.

Система розвитку персонажа передбачає накопичення досвіду, підвищення рівня, нарахування очок характеристик і їх розподіл між п’ятьма основними параметрами:

– сила;
– спритність;
– витривалість;
– живучість;
– опірність.

Підвищення характеристик доступне виключно поблизу спеціальних точок збереження.

2.7 Побудова структури рівнів

У проєкті реалізовано трирівневу структуру світу, кожен сегмент якої має унікальну ігрову логіку, архітектурну побудову та механіки взаємодії з гравцем. Рівні об’єднані системою переходів і точок збереження, що дозволяє формувати послідовний ігровий досвід у межах єдиної сесії.

Перший рівень є лінійним підземеллям, у межах якого гравець взаємодіє з ворогами, досліджує простір і просувається до виходу. У підземеллі реалізовано:

– система ворогів-патрульних – використано актори з індивідуальними маршрутами патрулювання, які задаються через сплайни або заздалегідь визначені контрольні точки. Це забезпечує динамічну загрозу та потребу в тактичному підході до пересування;
– три точки збереження – спеціальні області, у яких гравець може здійснити збереження поточного прогресу в межах ігрової сесії;
– перехід до наступного рівня – у кінці підземелля розташовано Actor-ворота, що виконують функцію логічного тригера переходу. Після того як гравець підходить до воріт, запускається анімація відкриття, і через короткий інтервал ч асу відбувається переміщення на другий рівень.

Другий рівень є відкритим світом, збудованим на базі ландшафтного актору. Простір наповнений процедурно розміщеним контентом і спрямований на дослідження, підготовку до фінального бою та взаємодію з ігровими системами.

- ресурси для крафту – на рівні розміщено статичні меші, які представляють матеріали для створення предметів. Гравець може підбирати їх упродовж подорожі, взаємодіючи з інвентарною та крафтовою системами;
- відсутність ворогів – рівень не передбачає бойових зіткнень, що дозволяє гравцю зосередитися на зборі ресурсів та орієнтації в просторі;
- невидима точка переходу – доступ до наступного рівня прихований: гравець має самостійно дослідити територію, щоби знайти гостел на пагорбі, який виконує роль переходу. Вхід до храму автоматично запускає переміщення на фінальний рівень;
- точка збереження біля костелу – розміщена неподалік від місця переходу й дозволяє зафіксувати прогрес перед фінальною битвою;
- третій рівень – це ізольована бойова арена, призначена виключно для поєдинку з фінальним босом. Це замкнутий простір без можливості повернення або втечі;
- єдиний ворог – бос – має поведінку, як у ворогів із підземелля, але значно вищі характеристики персонажа та зброї;
- фінальність рівня – після перемоги над босом подальший перехід не передбачено, що символізує завершення ігрової сесії;
- смерть гравця – у разі загибелі гравець повертається до останньої точки збереження на другому рівні, біля костелу, що підкреслює важливість правильної підготовки.

Уся система збереження у проєкті має сесійний характер – прогрес зберігається лише в межах однієї сесії гри та після виходу з гри дані очищуються.

2.8 Розробка AI та його поведінки

У проєкті реалізовано модульну систему штучного інтелекту, яка дозволяє керувати поведінкою ворожих акторів за допомогою інструментів Unreal Engine

– Behavior Tree, Blackboard та AI Perception System. Такий підхід забезпечує масштабованість, повторне використання логіки та адаптацію до різних типів супротивників, включно з босом.

Базова бойова AI-система побудована на таких компонентах:

– Behavior Tree використовується для формалізації логіки ухвалення рішень. Він визначає черговість дій, таких як патрулювання, переслідування, перевірка останнього місцезнаходження гравця та бойова атака;
– Blackboard виконує роль пам’яті бота. У ньому зберігаються ключові змінні, зокрема: ціль (гравець), останнє відоме положення, стан тривоги тощо;
– AI Perception System (зір) – налаштовано з використанням каналу "зору" як основного методу виявлення гравця. У разі візуального контакту бот миттєво переходить у бойовий стан.

Базовий цикл взаємодії виглядає так:

а) виявлення гравця – якщо гравець потрапляє у зону видимості, бот отримує відповідне повідомлення через AI Perception;
б) перехід у бойову поведінку – у контролері супротивника відбувається зміна стану, і бот починає переслідування та атаки на гравця;
в) втеча або смерть гравця – якщо гравець залишає зону видимості, бот запам’ятовує останнє відоме положення й вирушає туди;
г) втрата цілі – у разі відсутності гравця в цільовій точці, бот переходить у пасивний режим, який може включати:
   – патрулювання по заздалегідь заданому сплайну;
   – вибір випадкової точки на навігаційній сітці (Nav Mesh);
   – пасивне спостереження в одному напрямку.

Навігація ботів реалізована через Nav Mesh, який розміщено на першому (підземелля) та третьому (арена боса) рівнях, що забезпечує коректне планування маршруту супротивників.

Бос має загальну логіку поведінки, подібну до звичайних ворогів, однак з розширеним набором параметрів:

– підвищені бойові характеристики – бос має значно більший запас здоров’я, шкоду та радіус виявлення;
– сприйнятливість до всіх типів ефектів – на відміну від звичайних супротивників, які вразливі лише до ефекту заморозки, бос може бути уражений усіма трьома типами статусних ефектів (наприклад, кровотеча, заморозка, отруєння);
– фінальна арена – бос розміщується на замкненому рівні, де бою не уникнути, а втеча неможлива;

Винагороди за перемогу:

– очки досвіду – необхідні для підвищення рівня персонажа та розвитку характеристик;
– ігрову валюту – використовується в механіках купівлі або обміну;
– можливий лут – випадкові або гарантовані предмети, які додаються до інвентаря.

Такий підхід забезпечує інтеграцію бойової системи зі всіма ключовими ігровими механіками: прогресією, інвентарем, крафтом і переходами між рівнями.

2.9 Побудова інтерфейсу користувача UMG

Інтерфейс користувача побудовано за допомогою системи UMG (Unreal Motion Graphics). Його структура розроблена з акцентом на інтуїтивну взаємодію та логічне розміщення елементів. Кожен функціональний блок (інвентар, екіпірування, крафт, прокачування, налаштування тощо) має трирівневу побудову, що дозволяє гравцю швидко орієнтуватися в інформації та виконувати необхідні дії без надлишкового навантаження на увагу.

Вікно інвентаря розділено на три основні зони:

– список предметів та фільтрація – головна область, що містить усі наявні у гравця предмети. Над сіткою розміщені кнопки фільтрації за

категоріями (зброя, ресурси, амулети тощо), які дозволяють швидко знаходити потрібне;

– інформаційна панель предмета – при наведенні курсора або виборі предмета з’являється його розширений опис: назва, тип, кількість, властивості, ефекти та іконка;
– характеристики персонажа – область відображає поточні базові параметри гравця (сила, витривалість, спритність тощо), що дозволяє аналізувати вплив предметів на показники.

Інтерфейс екіпірування реалізований у три частини:

– слоти екіпірування – блок, де представлені осередки для зброї, амулетів і витратних предметів. Натиснувши на слот, гравець може обрати інший предмет зі списку доступного спорядження;
– інформація про предмет – виводиться розгорнутий опис предмета, на який наведено курсор, з усіма відповідними характеристиками та ефектами;
– параметри персонажа – розміщені поруч або під слотами, змінюються в режимі реального часу відповідно до обраного екіпірування.

Інтерфейс створення предметів складається з трьох панелей:

– список доступних для створення предметів – відображає всі об’єкти, які можливо створити. Якщо ресурсів достатньо, активується можливість перегляду подробиць при наведенні (назва, іконка, необхідні матеріали, їх кількість) та кнопка «Створити»;
– інформація про вибраний предмет – розміщена праворуч або під списком предметів, включає опис ігрового ефекту, тип і потенційні бонуси;
– характеристики персонажа – інформативна панель, яка дозволяє оцінити вплив нових предметів на поточні параметри.

Система підвищення рівня реалізована у вигляді трьох функціональних зон:

– загальна інформація – верхня частина екрану відображає кількість очок навичок, рівень персонажа, доступні гроші, поточний досвід та досвід до наступного рівня. У нижній частині – розподіл очок характеристик;
– базові параметри – верхній блок з HP і витривалістю; нижче – сума урону від двох активних видів зброї;
– система опору – відображення фізичного захисту від різних типів урону (колючий, рублячий, дроблячий).

Меню налаштувань дозволяє змінювати ключові параметри візуалізації:

а) графічні налаштування: Shadow, Reflection, Global Illumination, Visual Effects, Textures, Shading, Foliage, Anti-Aliasing, Resolution, режим вікна (повноекранний, віконний без рамки або віконний);
б) кнопка «Застосувати» – для підтвердження змін;
в) кнопка виходу з гри.

Контекстні повідомлення інформують гравця про можливі дії:

– підказки для підбору предметів та взаємодії (наприклад, «Натисніть [E], щоб підібрати», «[E], щоб взаємодіяти»);
– системні сповіщення про смерть.

Повідомлення виводяться автоматично.

При зміні рівнів демонструється спеціальне вікно завантаження, яке включає:

– ілюстрацію рівня або тематичне зображення;
– індикатор завантаження Circular Throbber.

Інтерфейс користувача реалізований за допомогою системи UMG, що забезпечує інтуїтивну навігацію та зручну взаємодію з усіма ключовими механіками гри. Завдяки трирівневій структурі кожного розділу (інвентар, крафт, екіпірування тощо), гравець може швидко знаходити потрібну інформацію та виконувати дії без перевантаження уваги, що особливо важливо для динамічного геймплею.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Вивчення архітектури та логіки реалізації геймплею

Архітектура проєкту побудована на основі рушія Unreal Engine 5 з дотриманням модульної структури. Основна логіка поведінки персонажа, а також системи інвентаря, екіпірування, крафту і розвитку характеристик реалізовані через Actor Component – незалежні компоненти, що прикріплюються до акторів та дозволяють легко масштабувати функціональність без жорсткого зв’язування з базовим класом.

Для програмування геймплейної логіки використовувалась візуальна система Blueprint. Це інтерпретована система сценаріїв, що дозволяє створювати поведінку ігрових об’єктів у вигляді вузлів (nodes) та подієвих графів (event graphs) без необхідності писати C++ код. Основна перевага Blueprint – це висока швидкість розробки, інтуїтивна візуалізація логіки та гнучка інтеграція з іншими системами рушія.

У процесі розробки, логіка в Blueprint спочатку зберігається як Blueprint Asset, що містить опис вузлів, змінних, подій і з’єднань між ними. Під час компіляції:

а) графи Blueprint перетворюються в Blueprint Bytecode (BPC) – власний байт-код Unreal Engine;

б) BPC інтерпретується Blueprint Virtual Machine (BVM), яка виконує логіку в реальному часі;

в) у режимі Editor / Development, код виконується динамічно з підтримкою налагодження, гарячої заміни та візуалізації стану;

г) у режимі Shipping (релізна збірка) відбувається оптимізація:

- Blueprint-код перетворюється на C++ через Nativization (опційно) [11] – система автоматично генерує C++ код із Blueprint-логіки, який потім компілюється до нативного коду;
- усі неактивні Debug-механізми та інтерпретатори вимикаються;

– активується оптимізований Garbage Collector;
– застосовується inlining, dead-code elimination та інші оптимізації компілятора Clang/MSVC.

Це дозволяє суттєво знизити навантаження на процесор та підвищити продуктивність гри на кінцевому пристрої.

У проєкті реалізовано подію, яка спрацьовує при натисканні клавіші 3. Ця подія активує спеціальну процедуру очищення рівня від усіх ворогів, які були створені спавнерами. Вся логіка реалізована у Blueprint у вигляді послідовності вузлів і виглядає наступним чином (рис. 3.1).

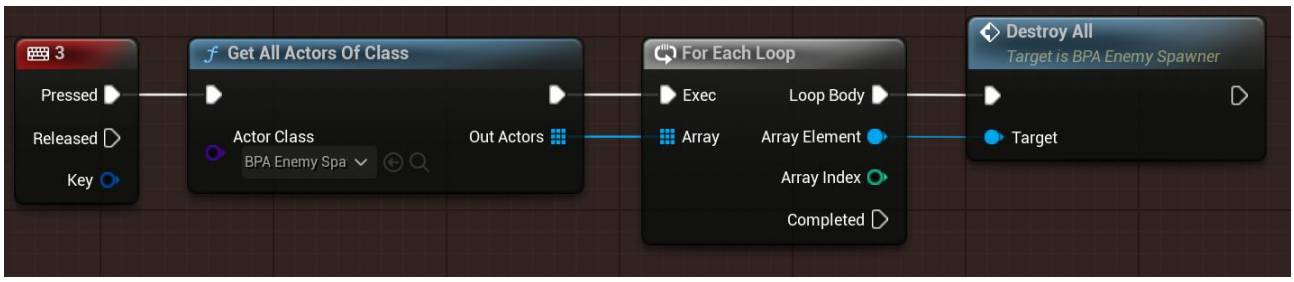


Рисунок 3.1 – Blueprint BPC_Player. Логіка знижчення ворогів

Кроки виконання логіки:

а) обробка натискання клавіші 3 – через подієвий вузол Input Key (3) відслідковується натискання відповідної клавіші;
б) отримання всіх спавнерів на рівні – використовується вузол Get All Actors of Class, де як клас вказано BPA_EnemySpawner. Результатом є масив усіх спавнерів, які існують на поточному рівні;
в) цикл перебору всіх спавнерів – за допомогою вузла For Each Loop відбувається ітерація по масиву спавнерів. Кожен спавнер передається як елемент у тілі циклу;
г) виклик функції очищення у кожного спавнера – для кожного спавнера викликається кастомна функція DestroyAll, яка реалізована всередині класу спавнера.

Завдяки використанню Blueprint та Actor Components вдалося досягти високої модульності, прозорості логіки та зручності тестування ігрових систем у середовищі Unreal Engine 5.

3.2 Технічна підготовка середовища проєкту

Перед початком безпосередньої реалізації проєкту було здійснено підготовку середовища розробки. Для реалізації проєкту було обрано стек технологій, який включає:

– Unreal Engine 5.5 – як основний рушій, що забезпечує підтримку сучасних геймплейних і графічних систем (Blueprint, AI, Niagara, UMG, World Partition тощо);
– Visual Studio 2022 – як інтегроване середовище розробки (IDE), необхідне для побудови проєкту, підключення SDK, компіляції C++-коду рушія та підтримки платформеної збірки під Windows.

У межах підготовки було виконано такі дії:

– встановлено Unreal Engine 5 через Epic Games Launcher;
– налаштовано Visual Studio з необхідними компонентами: Game development with C++, Desktop development with C++, а також Windows 11 SDK;
– створено базову структуру проєкту з шаблону "Third Person" з активацією стартових можливостей для подальшого розширення;
– перевірено коректність збірки й запуску базового проєкту під цільову платформу – Windows x64.

Таке технічне підґрунтя дало змогу забезпечити стабільну основу для реалізації геймплейної логіки, UI, AI та інших ключових систем.

3.3 Налаштування оточення у проэкті

Для реалізації візуальної складової ігрового середовища було здійснено низку технічних дій з налаштування та оптимізації сцен і процедурної генерації контенту.

Для формування оточення використовувались ресурси з таких платформ:

– Quixel Bridge – як джерело високоякісних Megascans-матеріалів (текстури, 3D-моделі рослинності, скель, поверхонь);
– Sketchfab / FAB – для інтеграції готових сцен і моделей з відкритого каталогу. Зокрема, було імпортовано набори [12].

Після імпорту:

– створено окрему тестову сцену, на яку було перенесено цей асет;
– усі зайві об’єкти (статичні меші, освітлення, актори ефектів) було видалено;
– сам ландшафт було вирівняно по сітці та приведено до коректної системи координат;
– налаштовано колізію ландшафту з використанням Collision Presets для забезпечення коректної взаємодії з гравцем і системами AI.

На другому рівні проєкту, що відповідає відкритому світу, було активовано Procedural Content Generation Framework (PCG Framework) [13] – офіційний плагін від Epic Games, який дозволяє реалізовувати генерацію контенту на основі правил та модулів безпосередньо в редакторі або під час виконання гри.

Кроки реалізації:

– через Plugins активовано Procedural Content Generation Framework;
– створено PCG Graph, що керує розміщенням об’єктів (наприклад, дерев, каміння, ресурсів);
– налаштовано джерело даних (ландшафт) та розміщення статичних мешів відповідно до параметрів висоти, схилу, маски шару.

Приклад файлу процедурної генерації (рис. 3.2).

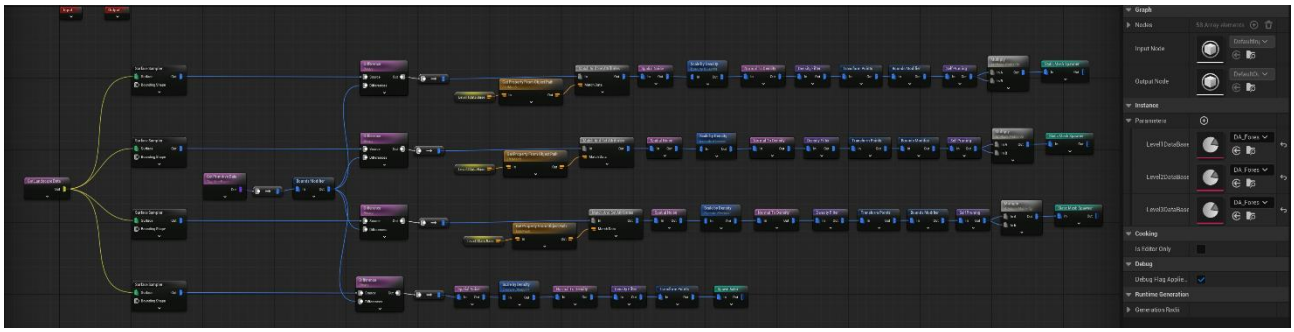


Рисунок 3.2 – Blueprint PCG_Forest. Логіка розміщення об’єктів на рівні

Для візуального і геймплейного було підключено плагін Water від Epic Games [14]. Це дозволило інтегрувати водні поверхні з підтримкою:

– симуляції хвиль;
– динамічної глибини;
– відображень і прозорості.

Після активації плагіна:

– додано Water Body Lake до другого рівня;
– налаштовано висоту рівня води, колізію, межі берегової лінії.

Таким чином, середовище гри було побудоване на основі зовнішніх високоякісних ассетів з інтеграцією процедурної генерації та сучасних систем симуляції природних елементів Unreal Engine 5.

3.4 Побудова базової логіки гри

На цьому етапі була реалізована основна геймплейна логіка, яка включає управління персонажем, систему анімацій, бойову механіку та базову поведінку противників.

В основі руху гравця використано стандартний шаблон Third Person Character з Unreal Engine 5. Він уже містить налаштовану систему вводу (Input Mapping Context), базову логіку руху, стрибків та керування камерою, що дозволило швидко запустити прототип.

Додатково до базових можливостей були реалізовані такі механіки:

– спринт – перемикання швидкості руху (у компоненті CharacterMovement) при натисканні відповідної клавіші;
– перекати (Dodge) – виконуються з допомогою тригерів дій (для обчислення напрямку перекату перевіряються напрямки руху персонажа значень: -1; 0; 1 для осей X та Y) і запуску відповідної анімації;

– атака – базова атака з можливістю робити combo – атаки до 4 ударів за 1 combo через Blueprint-умови. Для досягнення можливості робити combo – атаки використовувались таймер, та змінна для вістежування стадії комбо. Це означае, що у подальшому система може буди доповнена новими ударами.

Вся логіка доповнюється візуальними блоками коду у Blueprint-класі персонажа (рис. 3.3–3.4).

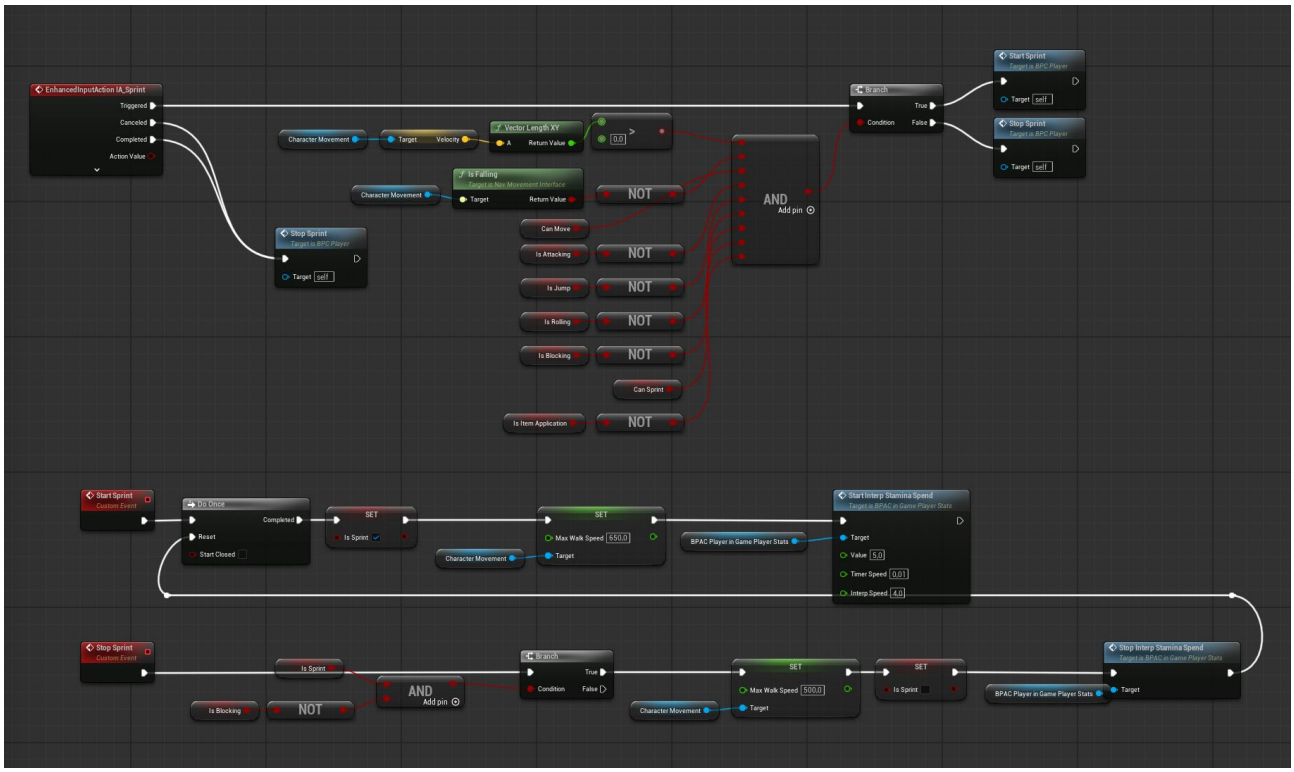


Рисунок 3.3 – Blueprint BPC_Player. Реалізація спринту

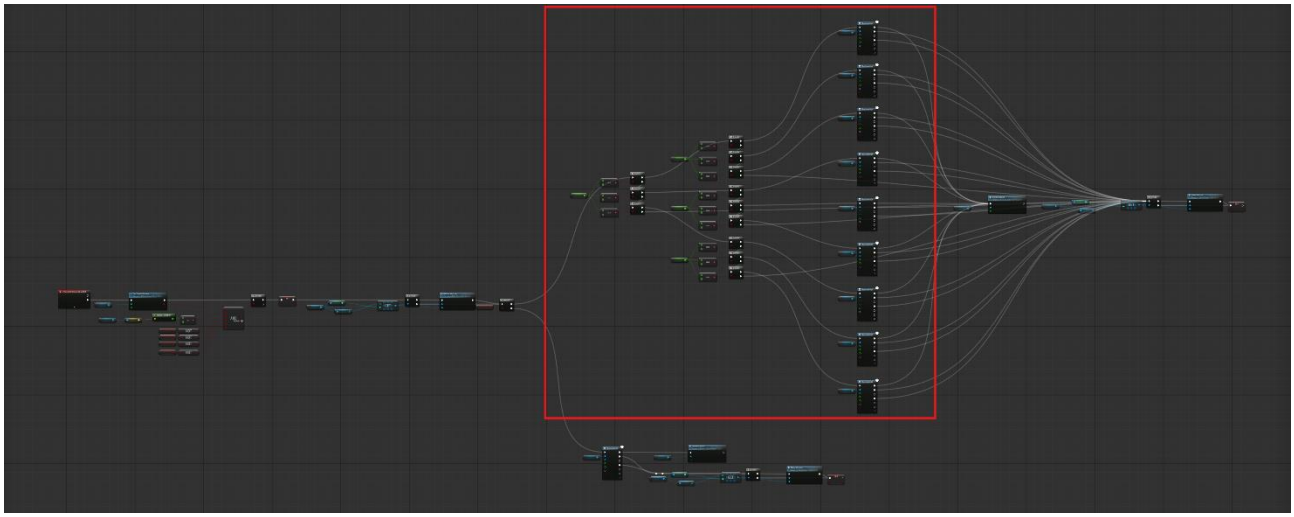


Рисунок 3.4 – Blueprint BPC_Player. Реалізація перекату

Після завершення налаштування ігрового персонажа, цей клас був скопійований для створення ворогів – таким чином, значна частина коду була пере використана, що пришвидшило розробку та зменшило дублювання логіки.

Анімаційна система побудована на базі Animation Blueprint, з використанням:

– State Machine – для перемикання між станами ходьби, бігу, стрибків тощо;
– Blend Space – для плавного переходу між рухами;
– Animation Montage – для реалізації атак, блоків та перекатів.
– Використання Montage дало можливість:
– запускати складні анімації з тригерами подій;
– вставляти логіку (наприклад, ефекти або пошкодження) у певні фрейми анімації;
– спростити розширення бойової системи в майбутньому.

Бойова механіка реалізована через кастомні події та функції у Blueprint. Основні можливості:

– нанесення шкоди (Damage) – виконується через функції Damage та Події (Get Damage) із застосуванням кастомного типу пошкоджень;
– блокування атак (Block) – перевіряється статус блоку та запускає або перериває відповідну анімацію. Це було зроблено за рахунок можливості зациклення анімації у Montage. При старті блоку запускається зациклена анімація а при завершені блоку нода Stop Anim Montage перериває анімацію (рис. 3.5);
– AI противника – для ворожих акторів були створені події замість інпутів.

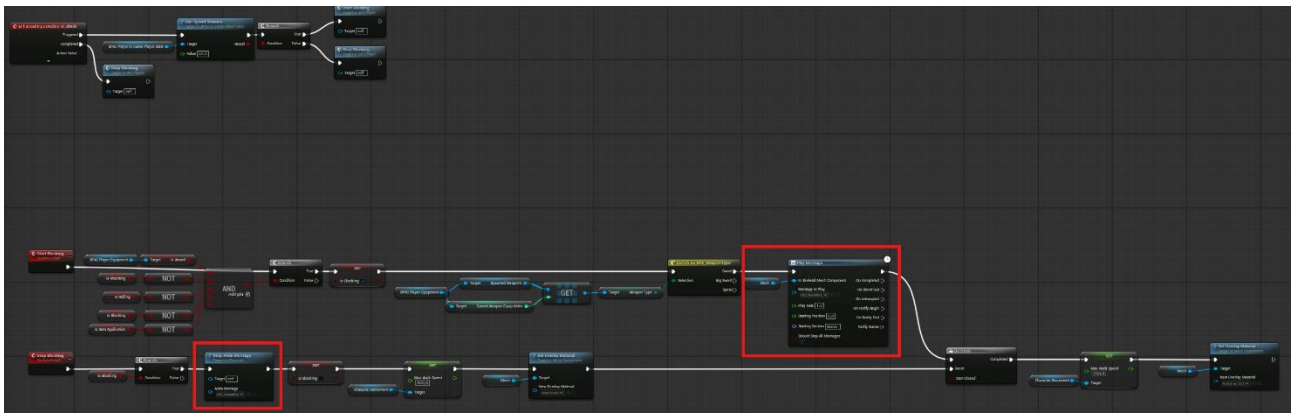


Рисунок 3.5 – Blueprint BPC_Player. Реалізація блоку

Завдяки використанню одного шаблонного класу персонажа, який адаптується під гравця або ворога, забезпечується модульність і висока повторюваність логіки. Це дозволяє легко масштабувати функціонал, повторно використовувати компоненти та знижує час на розробку.

3.5 Розробка системи прогресії

У межах розробки системи прогресії було реалізовано функціональну архітектуру, засновану на Actor Component-підході. Кожна з ключових підсистем – інвентар, екіпірування, витратні матеріали, характеристики персонажа та система прокачування – реалізована як окремий Actor Component, що забезпечує гнучкість, модульність і зручне масштабування.

Компонент BPAC_Inventory відповідає за обробку предметів в інвентарі та реалізує універсальну структуру з такими функціями:

- CreateStack, AddToStack, AddToInventory, RemoveFromInventoryByStruct, RemoveFromInventoryById – основна логіка роботи з предметами;
- ToggleInventory – показ / приховування UI інвентаря;
- макрос HasPartialStack – для дозаповнення вже створених стеків предметів.

Усі предмети зберігаються у вигляді масиву структур, що дає змогу гнучко обробляти різні типи даних і об’єднувати схожі предмети. При додаванні в

інвентар перевіряється, чи можливо дозаповнити існуючий стек. Якщо ні – створюється новий.

Компонент BPAC_PlayerEquipment відповідає за екіпірування:

– AddWeapon, RemoveWeapon, EquipWeapon, EquipNextWeapon – обробка зброї;
– AddAmulet, RemoveAmulet, AddConsumable, EquipNextConsumable – амулети й витратники;
– зброя зберігається у масиві структур (до 2 слотів), амулети й витратники – у масивах до 3 слотів кожен;
– усі екземпляри зброї зберігаються в масиві SpawnedWeapons з індексацією CurrentWeaponEquipIndex;
– є метод ToggleEquipment – для перемикання видимості екіпіровки.

BPAC_PlayerCrafting містить:

– масив доступних для створення предметів CraftingItems;
– функція CraftItem – проходить по ID матеріалів, знімає їх з інвентаря та додає новий предмет;
– функція CheckCanCreated – перевіряє наявність усіх необхідних ресурсів перед створенням.

Характеристики реалізовано в компоненті BPAC_PlayerStats:

– GetNextPlayerLevelXP, AddPlayerXP, PlayerLevelUp, AddSkillPoints, RemoveSkillPoints – базова логіка прокачки;
– PrevSkill, NextSkill, ApplyNewSkills, ToggleLevelUp – інтерфейс для розподілу характеристик;
– розрахунок значень відбувається динамічно: початкове значення, інтервал підвищення, максимальний рівень тощо;
– передбачено додаткові бонуси – наприклад, від екіпірованих амулетів.

Паралельно діє BPAC_InGamePlayerStats – відстежує актуальний стан персонажа під час гри:

– враховує здоров’я, витривалість, статусні ефекти;
– має Event Dispatchers: GetDamage, PlayerDeath, NoStamina, CanMove;
– статуси активуються через окремі події – зручний контроль на рівні логіки гри.

Для управління прокачуванням створено актор BPA_PlayerHub – це інтерактивний об’єкт з візуальним представленням і колізією. Він має наступну логіку:

– CanWithHubInteraction – перевіряє відсутність ворогів поблизу;
– StartWithHubInteraction – знищує ворогів, відкриває UI меню прокачування;
– EndWithHubInteraction – повертає управління гравцю, респавнить ворогів.

Таке рішення дозволяє органічно інтегрувати систему розвитку персонажа у світ гри, надаючи гравцеві безпечну зону для прокачування.

Спавнер ворогів реалізовано через BPA_EnemySpawner:

– зберігає типи ворогів у масиві структур EnemyTypes;
– має методи Spawn та DestroyAll, масив SpawnedEnemies – для керування інстансами.

Окремо створено BPA_MasterWeapon – шаблон зброї:

– має змінні характеристик, логіку візуального розташування;
– реалізовано точну систему детекції попадань – через побудову spline по мешу й Sphere Trace між точками (рис. 3.6);
– удари фіксуються лише при активації спеціального івенту з анімаційного монтажу;
– методи: CalcDamage, CalcNewDamage, HitsRegistration – відповідають за нанесення шкоди, спавн ефектів, визначення джерела атаки (гравець/ворог).

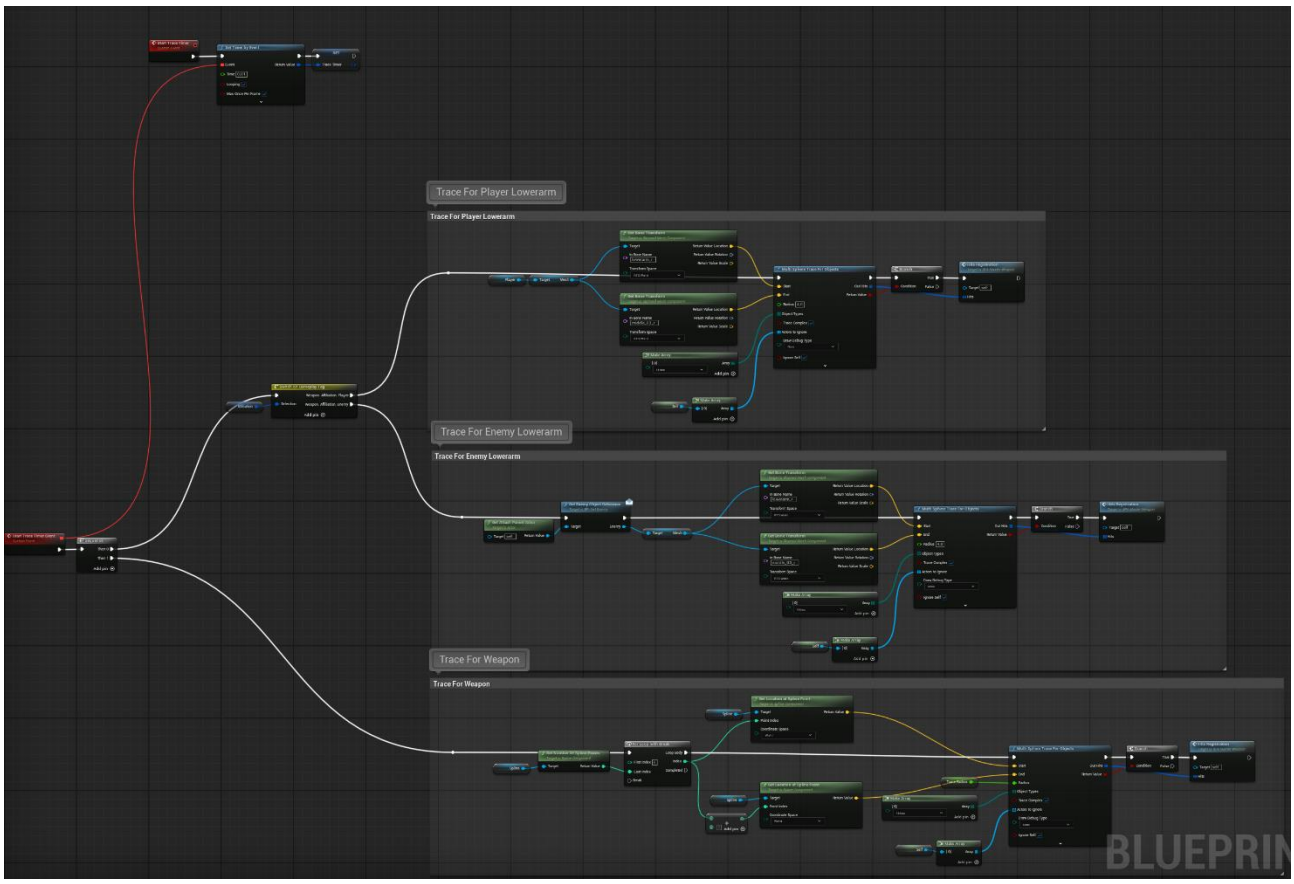


Рисунок 3.6 – Blueprint BPA_MasterWeapon. Детекція попадань зброї

Для AI використано ті самі компоненти, але з адаптацією: видалено логіку витривалості, змінено власника зброї – ворог замість гравця. Це дало змогу забезпечити спільну бойову систему як для гравця, так і для супротивників з мінімальним дублюванням коду.

3.6 Побудова структури рівнів

У проєкті реалізовано три послідовні рівні, кожен із яких має власну архітектуру, логіку та механізми переходу. Перемикання між рівнями здійснюється через взаємодію з визначеними акторами, що супроводжується анімаційними чи функціональними подіями.

Перший рівень (M_Dunge) – це лінійне підземелля, створене на основі готових ассетів.

– у межах рівня налаштовано патрулювання ворогів зі шляхами, побудованими за допомогою сплайнів;
– для переходу на другий рівень використовується актор BPA_OpenedGate (рис. 3.7). Він має зону активації, яка реагує на вхід гравця;
– після активації запускається таймлайн тривалістю 5 секунд, що поступово змінює обертання воріт і створює ефект відкриття;
– по завершенню анімації перевіряється, чи живий гравець;
– якщо так – усі вороги на рівні знищуються, активується завантажувальний екран, відбувається переміщення на другий рівень і вивантаження першого з пам’яті.

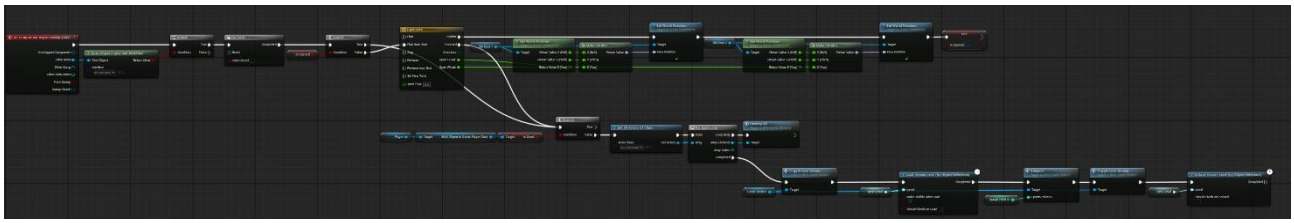


Рисунок 3.7 – Blueprint BPA_OpenedGate. Перехід на новий рівень

Другий рівень (M_Forest) є відкритим світом, реалізованим на базі ландшафтного актору. Основна мета цього сегменту – дослідження території та збір ресурсів.

– ворожі актори тут відсутні;
– через систему процедурної генерації на рівні розміщується актор BPA_HealingHerb, представлений у вигляді Static Mesh. Гравець може збирати ці об’єкти як матеріали для крафту;
– для переходу на третій рівень використовується актор BPA_Capel. Він має тригер-зону, при вході в яку гравець автоматично переміщується на наступний рівень. У цьому випадку завантаження здійснюється миттєво і без вивантаження другого рівня з пам’яті.

Третій рівень (M_BossArena) – це ізольована арена для фінального бою з босом. Він також створений із використанням готових ассетів.

– перехід на нього відбувається через актор BPA_Capel;

– арена повністю замкнута – після переміщення гравець не має можливості повернутись, якщо не помре. При смерті гравець переміщкеться до останьої точки збереження на 2 рівні;
– уся логіка, пов’язана з боєм проти боса, реалізується окремо на цьому рівні.

Побудована структура рівнів охоплює три послідовні середовища з чітким поділом функцій: бойове підземелля, дослідницький відкритий світ та ізольована арена для фінального протистояння. Такий підхід дозволяє поступово ускладнювати ігровий процес, а також ефективно керувати ресурсами пам’яті завдяки поетапному завантаженню та вивантаженню рівнів.

3.7 Реалізація AI та бойової поведінки

Штучний інтелект ворогів реалізовано за допомогою систем Behavior Tree і Blackboard у поєднанні з компонентом AI Perception. Основна логіка реагування на гравця та управління станами ворогів зосереджена в окремому AI-контролері, створеному для обробки поведінки як звичайних ботів, так і босів.

Базовий контролер BPAIC_MasterController містить набір функцій, які визначають реакцію бота на зорові та бойові стимули. Зокрема, функції:

– RememberSpawnLocation;
– SetIsRandomPointSecurity;
– ProcessVision;
– ProcessDamage.

використовуються для збереження точки спавну, активації випадкового патрулювання та обробки подій виявлення або отримання ушкодження.

Компонент AI Perception налаштований на сприйняття візуальних стимулів та впливів від ушкодження, з відповідними реакціями в залежності від типу джерела.

У випадку виявлення актора зору, функція ProcessVision визначає, чи є ціль гравцем. Якщо виявлений об’єкт не є гравцем, перевіряється його стан. Якщо об’єкт мертвий, його остання позиція записується в Blackboard як

LastKnownLocation. Якщо ж виявлений саме гравець, поведінка залежить від його стану: якщо гравець мертвий, змінні EnemyTarget і LastKnownLocation очищуються; якщо живий – ціль фіксується в EnemyTarget, а також очищуються змінна LastKnownLocation. При втраті візуального контакту останнє положення гравця записується в LastKnownLocation.

Поведінка бота динамічно змінюється в Behavior Tree (рис. 3.8) на основі значень у Blackboard. Ворог переходить між станами пасивності, настороженості або атаки, реагуючи на ситуацію у просторі гри.

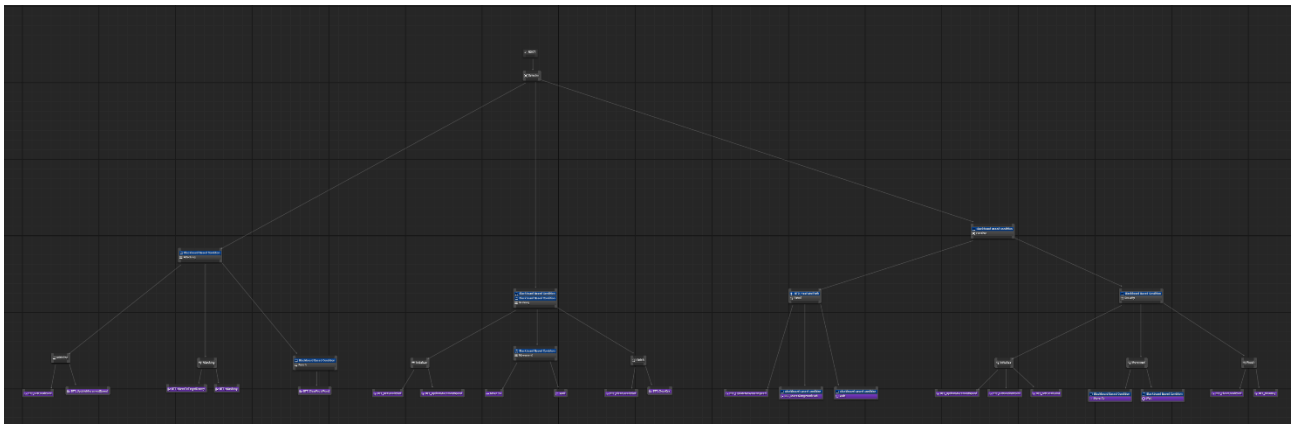


Рисунок 3.8 – Behavior Tree BT_Skeleton. Реалізація поведінки

Для боса реалізовано окрему логіку з власним контролером BPAIC_BossSkeleton, який також наслідує базовий контролер. На відміну від звичайних ворогів, бос не використовує стан настороженості – він постійно орієнтований на гравця та одразу переходить у бойовий режим. Для нього застосовуються окремі Behavior Tree та Blackboard, що дозволяє реалізувати унікальну поведінку.

Звичайні вороги керуються через контролер BPAIC_Skeleton. Обидва типи використовують навігаційні сітки (NavMesh), які попередньо встановлені на першому та третьому рівнях для забезпечення коректної навігації та руху ворогів по складній геометрії рівнів.

У результаті реалізовано гнучку та масштабовану систему AI, що охоплює як звичайних ворогів, так і унікальних босів, з глибокою логікою поведінки. Це дозволяє легко розширювати бойову систему, створювати складні сценарії та забезпечувати динамічний геймплей із адаптивними викликами для гравця.

3.8 Побудова інтерфейсу користувача UMG

Інтерфейс користувача реалізовано за допомогою системи UMG у межах кількох функціональних виджетів, які відповідають за управління інвентарем, екіпіруванням, характеристиками персонажа, повідомленнями, екраном завантаження та системними налаштуваннями.

Всі ключові виджети створюються, додаються до вьюпорту або знищуються за допомогою функцій Toggle-, які викликаються з актор-компонентів відповідно до ігрового контексту. Це дозволяє централізовано контролювати появу інтерфейсних елементів залежно від дій гравця (наприклад, відкриття інвентаря, отримання ушкоджень або завантаження рівня).

Для зручності взаємодії з інтерфейсом у грі створено головний виджет WBP_MenuInGame, який виконує роль центрального хаба доступу до всіх підсистем. У цьому меню реалізовані кнопки переходу до таких підменю:

– екіпірування;
– крафт;
– інвентар;
– покращення характеристик;
– налаштування.

Інтерфейс налаштувань реалізований у вигляді окремого виджета. Базові налаштування (наприклад, гучність або яскравість) задаються через слайдери з діапазоном значень від 0 до 4. Вибір роздільної здатності екрану та режиму вікна реалізований за допомогою інтерфейсного елемента з кнопками переходу "вліво/вправо" і поточним значенням у центрі – подібно до спискового перемикача.

Для збереження користувацьких параметрів застосовується вбудована система Settings [15]. Через неї інтерфейс отримує, змінює та застосовує системні налаштування, що гарантує їхнє збереження між сесіями.

Крім того, у проєкті реалізовано:

– повідомлення про смерть гравця з подальшим поверненням до останньої точки збереження;
– підказки контекстної дії (наприклад, “Натисніть E для взаємодії”);
– завантажувальний екран, який відображається під час переходу між рівнями. Цей виджет активується перед початком завантаження та автоматично приховується після його завершення.

Уся взаємодія з UMG-інтерфейсом побудована таким чином, щоб забезпечити максимальну гнучкість і контроль через функціональні виклики в логіці акторів, без жорсткої прив’язки до конкретних рівнів або ситуацій.

3.9 Тестування, оптимізація, фінальна збірка

Завершальний етап розробки передбачав комплексне тестування, оптимізацію логіки та продуктивності, а також формування фінальної збірки проєкту для демонстрації.

У процесі фіналізації було реалізовано низку заходів для підвищення продуктивності, зменшення затримок при завантаженні та забезпечення стабільної роботи на всіх рівнях. Основним інструментом оптимізації виступила система Level Streaming, яка дозволяє динамічно завантажувати та вивантажувати частини ігрового світу залежно від прогресу гравця.

На відміну від прямого завантаження рівнів (Open Level) або завчасного завантаження всіх елементів у пам’ять, Level Streaming дозволив:

а) уникнути значних затримок на старті гри;
б) зменшити блокування основного потоку;
в) забезпечити плавне завантаження без зависань.

Кожен рівень гри реалізований як окремий підрівень:

г) після завершення першого рівня (підземелля) – другий рівень (відкритий світ) завантажується, а перший вивантажується;
д) перехід до третього рівня (бос-арена) здійснюється після входу до каплиці з відповідним завантаженням тільки потрібного підрівня.

Цей підхід значно зменшив навантаження на оперативну пам’ять і забезпечив стабільну частоту кадрів навіть на середніх конфігураціях ПК.

Додатково було оптимізовано складні механіки: трасування колізій при атаках виконується лише в потрібні моменти через події (Events) у Montage, що знижує навантаження на фізичний рушій.

Проведено всебічне тестування основних ігрових систем:

е) інвентар – додавання, видалення, сортування, використання предметів;
ж) екіпірування – робота слотів, вплив на характеристики, інтеграція з боєм;
з) крафт – перевірка логіки доступності, списання ресурсів, результат;
и) прокачування – досвід, рівні, очки характеристик, вплив на параметри;
к) системи збереження – перевірено сесійну модель та виклик чекпойнтів на всіх рівнях;
л) AI та бойова система – реакції на гравця, стабільність поведінки, бойові фази.

Окрема увага приділена логіці переходів між рівнями, роботі UI при зміні контекстів, а також коректності логіки збереження та повернення до чекпойнтів.

Виправлено критичні помилки, зокрема: некоректне збереження станів, зависання при переході фаз бою, збої при втраті цілі, неправильній роботі колізії на початку гри.

Під час тестування бойової системи були виявлені та усунені критичні помилки, пов’язані з роботою колізій:

м) виявилося, що при дуже близькому розташуванні атакуючого до противника урон іноді не проходив. Причиною стало те, що для реєстрації урону використовувалась лише колізія зброї. У деяких типах атак, особливо з широкими дугами, між плечем персонажа та початковою точкою зброї залишався зазор без активної колізії. У таких випадках противник міг фактично "опинитися всередині" атакуючого, минаючи зону урону. Для вирішення цієї ситуації до зброї було додано допоміжну колізію на руці атакуючого. Це дозволило розширити зону нанесення урону в ближньому бою та забезпечити очікувану поведінку без збільшення кількості перевірок;

н) спочатку передбачалась класична фронтальна система блоку, але під час тестів виникли проблеми з точкою ураження. Наприклад, коли гравець і противник стояли один перед одним, анімації ударів (із замахами та випадами) могли змістити точку ураження в бік, у зону поза фронтом блоку – у такому разі блок не спрацьовував, хоча з точки зору гравця атака була очевидно заблокована. Ще одна проблема виникала при ударах ззаду: через специфіку гри (зброя може проходити крізь об’єкти), удар міг "зачепити" спершу модель гравця, а потім зброю. Це порушувало логіку – гравець активував блок, але урон все одно проходив або, навпаки, не проходив там, де мав.

Щоб уникнути неоднозначних ситуацій і зберегти динамічність бою, було прийнято рішення реалізувати всенаправлений блок: якщо блок активний у момент атаки – урон не проходить, незалежно від напрямку. Баланс досягався за рахунок витрат витривалості у режимі блоку, а також її часткового поглинання атакою. У цьому режимі стаміна не відновлюється, що змушує гравця стратегічно використовувати захист.

Фінальна збірка пакована під платформу Windows з використанням системи Unreal Engine 5.5 через інструмент Package Project, що дозволяє створити самостійну версію гри з усіма залежностями (включно з .exe-файлом) без потреби встановлення Unreal Engine.

Готова збірка включає:

о) усі три рівні, інтегровані через Level Streaming;

п) повноцінний ігровий цикл: старт → відкритий світ → бос-арена;

р) повний інтерфейс (UMG), крафт, екіпірування, прокачування, бойову логіку;

с) системи збереження, чекпойнтів, повідомлень, меню.

У результаті:

т) гра після збірки займає 3,27 ГБ;

у) не потребує спеціальних умов для запуску;

ф) може бути запущена на будь-якому ПК з Windows 11 та підтримкою DirectX 12.

Це дозволяє зручно демонструвати проєкт на широкому спектрі систем без необхідності додаткової конфігурації.

ВИСНОВКИ

В результаті виконаної роботи розроблена 3D гра з використанням ігрового рушія Unreal Engine 5.

У процесі розробки було реалізовано повноцінний ігровий прототип у жанрі Soulslike, зосереджений на побудові базових механік: бою, інвентаря, крафту, прокачування, збереження прогресу, переміщення між локаціями та взаємодії з об’єктами світу. Гра охоплює три повноцінні рівні, які логічно пов’язані між собою, а також реалізує повний цикл взаємодії гравця зі світом – від старту до бою з фінальним босом.

Було розроблено понад 30 керуючих скриптів на візуальній мові програмування Blueprints, без використання мови C++. Це забезпечило високу гнучкість та швидкість створення прототипу, а також зробило проєкт доступним для подальшого вдосконалення студентами чи інди-розробниками, які не мають глибоких знань у програмуванні.

Під час розробки використано такі технології та плагіни Unreal Engine 5:

х) Unreal Motion Graphics (UMG) – для створення багаторівневого інтерфейсу користувача;
ц) Level Streaming – для динамічного завантаження та вивантаження рівнів без затримок;
ч) Behavior Trees та AI Perception – для створення інтелектуальної поведінки супротивників;
ш) Animation Montages – для організації бойової анімації;
щ) Blueprint Interfaces – для передачі даних між окремими компонентами системи;
ы) FAB Plugin (Sunset – Modular Medieval Brick Buildings) – для побудови середовища;
э) Procedural Content Generation Framework – для оптимізації контенту та створення унікальних елементів;
ю) Water System – для створення візуально реалістичних водних поверхонь;

я) Settings API – для налаштування графічних параметрів у реальному часі.

Проєкт також включає адаптивну бойову систему з елементами тактичної взаємодії, систему екіпірування з впливом на характеристики персонажа, інвентар з фільтрацією та кастомізовану систему крафту, яка працює на основі доступних ресурсів і враховує характеристики гравця. Прокачування реалізоване через очки навичок, що розподіляються у спеціальних точках збереження.

Фінальна збірка гри була сформована під Windows з урахуванням підтримки DirectX 12 і сучасних графічних API, що дозволяє запускати проєкт навіть на середньостатистичних комп’ютерах без потреби у високопродуктивному апаратному забезпеченні.

Розробку можна використовувати як технічну демонстрацію, навчальний матеріал або основу для комерційного проєкту в жанрі RPG або Soulslike. Також її можливо впровадити в освітні програми з вивчення Unreal Engine, розробки ігрових інтерфейсів, систем управління, логіки AI та побудови ігрового світу.

Таким чином, дана кваліфікаційна робота продемонструвала можливість повноцінної реалізації ключових ігрових механік виключно на основі Blueprint-системи Unreal Engine 5, із залученням сучасних технологій та практик оптимізації, що робить проєкт ефективним прикладом розробки 3D-гри в сучасних умовах.

ПЕРЕЛІК ПОСИЛАНЬ

1. Unreal Engine 5 Features. Unreal Engine. URL : https://www.unrealengine.com/en-US/unreal-engine-5 (дата звернення : 21.05.2025).

2. Soulslike. Wikipedia. URL : https://ru.wikipedia.org/wiki/Soulslike (дата звернення : 21.05.2025).

3. Nanite Virtualized Geometry. Unreal Engine Documentation. URL: https://docs.unrealengine.com/5.0/en-US/nanite-virtualized-geometry-in-unreal-engine (дата звернення: 21.05.2025).

4. Lumen Global Illumination and Reflections. Unreal Engine Documentation. URL: https://docs.unrealengine.com/5.0/en-US/lumen-global-illumination-and-reflections-in-unreal-engine (дата звернення: 21.05.2025).

5. Programming with C++. Unreal Engine Documentation. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/programming-with-cplusplus-in-unreal-engine (дата звернення: 21.05.2025).

6. Blueprint Visual Scripting. Unreal Engine Documentation. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/blueprints-visual-scripting-in-unreal-engine (дата звернення: 21.05.2025).

7. Global games market outlook: Key growth drivers and challenges for 2025–2027. Newzoo. URL: https://newzoo.com/insights/articles/3d-games-market-trends-2025 (дата звернення: 21.05.2025).

8. Lies of P Review. IGN. URL: https://www.ign.com/articles/lies-of-p-review (дата звернення: 21.05.2025).

9. CD Projekt RED. The Witcher 3: Wild Hunt – Game Mechanics. The Witcher. URL: https://thewitcher.com/en/witcher3 (дата звернення: 21.05.2025).

10. FAB. URL: https://www.fab.com/ (дата звернення: 21.05.2025).

11. Nativizing Blueprints. Unreal Engine Documentation. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/nativizing-blueprints?application_version=4.27 (дата звернення: 21.05.2025).

12. Sunset – Modular Medieval Brick Buildings. FAB. URL: https://www.fab.com/listings/3b168493-061b-45fa-9693-0c7334a9b6c1 (дата звернення: 21.05.2025).

13. Procedural Content Generation Framework. Unreal Engine Documentation. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/procedural-content-generation--framework-in-unreal-engine?application_version=5.4 (дата звернення: 21.05.2025).

14. Water System. Unreal Engine Documentation. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/water-system-in-unreal-engine (дата звернення: 21.05.2025).

15. Settings. Unreal Engine Documentation. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/BlueprintAPI/Settings (дата звернення: 21.05.2025).

ДОДАТОК А

Посилання на скомпільовану версію гри

Скомпільована версія гри "Souls Like Template" доступна для завантаження за наступним посиланням: https://drive.google.com/file/d/1XyZAbCDefgHIJKlmnoPqrStUvWXyz/view?usp=sharing

Технічна інформація:

а) назва архіву: Windows.zip;

б) каталог всередині архіву: Game Build;

в) розмір архіву: 3,13 ГБ;

г) формат: ZIP-архів;

д) інструкція щодо запуску гри:

- перейти за посиланням;
- перейти до папки Game Build;
- завантажити архів;
- розпакувати вміст у зручне місце на диску;
- запустити виконуваний файл SoulsLikeTemplate.exe.