ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Кооперативна гра жанру rogue-like на основі Unreal Engine 5»
на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Технології цифрового розвитку»

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

_____ Михайло Бахмут
(підпис)

Виконав: здобувач вищої освіти групи ТЦР-44
Михайло Бахмут

Керівник: Іван Олейніков

Рецензент:

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Технології цифрового розвитку»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Технологій цифрового розвитку
_____ Вікторія ЖЕБКА
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бахмуту Михайлу Сергійовичу

1. Тема кваліфікаційної роботи: «Кооперативна гра жанру rogue-like на основі Unreal Engine 5»
керівник кваліфікаційної роботи ст. викл. Іван Олейніков,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.
2. Строк подання кваліфікаційної роботи «18» травня 2026 року.
3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи та засоби розробки комп'ютерних ігор, принципи побудови roguelike/roguelite-ігор,

архітектурні особливості Unreal Engine 5, технічна документація з Unreal Engine 5, матеріали щодо Blueprint-системи, систем штучного інтелекту, бойової взаємодії, генерації ігрових подій та тестування програмного забезпечення.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд існуючих методів, технологій та програмних засобів розробки комп'ютерних ігор жанру roguelike/roguelite.
2. Проєктування архітектури та основних підсистем кооперативної гри жанру rogue-like на основі Unreal Engine 5.
3. Програмна реалізація та опис функціонування основних механік гри: керування персонажем, бойової системи, взаємодії з ворогами, системи предметів, нагород і прогресії забігу.
4. Тестування програмної системи та аналіз результатів розробки.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Загальна архітектура програмної системи.
4. Структурна схема взаємодії основних підсистем гри.
5. Діаграма класів / Blueprint-структури.
6. Алгоритм роботи основних ігрових механік.
7. Екрани користувацького інтерфейсу та приклади ігрового процесу.
8. Результати тестування та апробація розробленої системи.

6. Дата видачі завдання «20» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	01.03.–03.03.2026	Виконано
2	Огляд аналогів та побудова вимог на їх основі	04.03.–06.03.2026	Виконано
3	Проєктування архітектури гри	07.03.–10.03.2026	Виконано
4	Програмна реалізація базових механік гри в Unreal Engine 5	11.03.–15.03.2026	Виконано
5	Запуск та перевірка основного функціоналу	16.03.–22.03.2026	Виконано
6	Тестування гри та налаштування ігрового інтерфейсу	23.03.–05.04.2026	Виконано
7	Вирішення основних помилок і доопрацювання систем прогресії	06.04.–12.04.2026	Виконано
8	Розробка демонстраційних матеріалів	13.04.–25.04.2026	Виконано
9	Попередній захист роботи	26.04.–09.05.2026	Виконано

Здобувач вищої освіти _____ Бахмут Михайло

(підпис)

Керівник

кваліфікаційної роботи _____ Олейніков Іван

(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 66 стор., 5 табл., 23 рис., 20 джерел.

Мета роботи – удосконалення процесу проєктування та реалізації базового ігрового циклу кооперативної гри жанру rogue-like шляхом розробки MVP-прототипу на основі Unreal Engine 5 з використанням Blueprints Visual Scripting, компонентної організації gameplay-систем, системи предметів і прогресії забігу.

Об'єкт дослідження – процес розробки комп'ютерної гри жанру rogue-like у середовищі Unreal Engine 5.

Предмет дослідження – методи та засоби реалізації базових gameplay-систем кооперативної rogue-like гри за допомогою Blueprints Visual Scripting та інструментів Unreal Engine 5.

Короткий зміст роботи: У роботі досліджено специфіку розробки комп'ютерних ігор жанру rogue-like/roguelite, акцентуючи увагу на таких аспектах, як структура ігрового циклу, система випадкових нагород, прогресія під час забігів, механіка бойової взаємодії та принципи повторюваності ігрових сесій. Також проаналізовано можливості Unreal Engine 5 для реалізації подібних проєктів, зокрема використання Blueprint Visual Scripting, Gameplay Framework, Actor Components, системи введення Enhanced Input, інструментів створення користувацького інтерфейсу та механізмів обробки ігрових даних. Розроблено архітектуру гри та програмно реалізовано ключові функціональні можливості: керування персонажем, систему характеристик, бойову взаємодію, ворогів, систему спавну, випадкові нагороди, предмети, скрині, інтерфейс гравця, завершення забігу та відображення результатів. Для підвищення масштабованості проєкту частина ігрової логіки винесена в окремі Blueprint-класи та Actor Components, а параметри предметів і характеристик організовані у data-driven форматі. Проведено тестування основних механік гри, перевірено коректність

роботи бойової системи, отримання нагород, зміни характеристик персонажа та завершення ігрової сесії.

Сферою використання розробленого програмного забезпечення є створення розважального ігрового застосунку, а також використання проєкту як основи для подальшого розвитку roguelike/roguelite-гри з розширеними механіками, новими персонажами, ворогами, предметами та кооперативним режимом.

КЛЮЧОВІ СЛОВА: ROGUE-LIKE, ROGUELITE, UNREAL ENGINE 5, BLUEPRINTS, GAMEPLAY FRAMEWORK, ACTOR COMPONENTS, ІГРОВИЙ ЦИКЛ, ІГРОВА ПРОГРЕСІЯ.

ЗМІСТ

ВСТУП

Актуальність теми. Комп'ютерні ігри є одним із найбільш динамічних напрямів розвитку сучасного програмного забезпечення, оскільки поєднують у собі графічні технології, системи штучного інтелекту, інтерактивну взаємодію користувача, мережеві можливості, обробку подій у реальному часі та складну внутрішню архітектуру [11]. Особливе місце серед ігрових жанрів займають rogue-like та roguelite-ігри, для яких характерні повторювані ігрові сесії, випадкові нагороди, прогресія персонажа, поступове ускладнення ігрового процесу та висока варіативність проходження [12].

Розробка гри такого жанру є доволі складним інженерним завданням, оскільки програмна система повинна гарантовано забезпечувати стабільну роботу численних взаємопов'язаних підсистем. До таких належать: управління персонажем, бойова механіка, система характеристик, модулі ворогів і предметів, випадкові нагороди, прогресія забігу, інтерфейс користувача та процедура завершення ігрової сесії. У разі створення кооперативної гри рівень складності зростає ще більше, адже це вимагає врахування взаємодії між кількома гравцями, синхронізації ігрового стану та правильної організації даних, що стосуються кожного учасника ігрового процесу.

Unreal Engine 5 є сучасним ігровим рушієм, який надає великі можливості для розробки інтерактивних застосунків і комп'ютерних ігор [1]. Однією з його переваг є Blueprint Visual Scripting – система візуального програмування, що дозволяє швидко реалізовувати gameplay-логіку без обов'язкового використання мови C++ [2]. Водночас Blueprint-first підхід потребує правильної архітектурної організації, оскільки надмірно великі та жорстко пов'язані Blueprint-графи ускладнюють подальшу підтримку, тестування та розширення проєкту [9].

Саме тому актуальним є розроблення кооперативної гри жанру rogue-like на основі Unreal Engine 5 із використанням модульної архітектури, Actor Components, окремих Blueprint-класів для основних ігрових сутностей та data-driven підходу до

опису предметів, характеристик і нагород [5]. Такий підхід дозволяє зменшити дублювання логіки, спростити балансування ігрових параметрів і створити основу для подальшого розширення гри.

Мета роботи – удосконалення процесу проєктування та реалізації базового ігрового циклу кооперативної гри жанру rogue-like шляхом розробки MVP-прототипу на основі Unreal Engine 5 з використанням Blueprints Visual Scripting, компонентної організації gameplay-систем, системи предметів і прогресії забігу.

Для досягнення поставленої мети в кваліфікаційній роботі слід реалізувати такі завдання:

1. Дослідити особливості ігор жанру rogue-like та roguelite.
2. Розглянути та розібрати існуючі аналоги та визначити основні вимоги до нашої гри.
3. Розглянути можливості Unreal Engine 5 для створення ігрових застосунків.
4. Спроєктувати архітектуру гри та структуру основних Blueprint-класів.
5. Реалізувати систему керування персонажем.
6. Реалізувати систему характеристик, здоров’я, шкоди та бойової взаємодії.
7. Реалізувати базову логіку ворогів і систему їх появи на ігровій арені.
8. Розробити систему предметів, скринь, випадкових нагород і прогресії забігу.
9. Створити користувацький інтерфейс для відображення основних параметрів гри.
10. Провести тестування основних механік та проаналізувати результати роботи програмної системи.

Об’єкт дослідження – процес розробки комп’ютерної гри жанру rogue-like.

Предмет дослідження – методи та засоби реалізації кооперативної гри жанру rogue-like на основі Unreal Engine 5.

Методи дослідження. Під час виконання кваліфікаційної роботи використано методи теоретичного аналізу для дослідження жанрових

особливостей rogue-like/roguelite-ігор, методи порівняльного аналізу для огляду існуючих аналогів, методи об’єктно-орієнтованого проєктування для побудови архітектури програмної системи, методи компонентного проєктування для винесення повторюваної gameplay-логіки в Actor Components, а також методи функціонального тестування для перевірки працездатності реалізованих механік.

Наукова новизна одержаних результатів. У роботі запропоновано підхід до побудови модульної архітектури rogue-like гри в Unreal Engine 5 з використанням Blueprint-first розробки, розподілу відповідальності між основними класами Gameplay Framework та винесенням повторюваної логіки в Actor Components. Особливістю підходу є поєднання швидкої розробки засобами Blueprints із data-driven організацією ігрових параметрів, що дозволяє змінювати баланс предметів, характеристик і нагород без суттєвої зміни основної логіки гри.

Практичне значення отриманих результатів полягає в тому, що розроблений прототип може слугувати основою для подальшого створення повноцінної гри у жанрі rogue-like або roguelite. Запропонована архітектура забезпечує гнучкість у розвитку проєкту, дозволяючи легко додавати нових персонажів, ворогів, предмети, нагороди, ігрові режими та кооперативні механіки. Також важливою є можливість використання результатів роботи як зразка організації проєктів за підходом Blueprint-first у середовищі Unreal Engine 5.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Бахмут М.С. Проєктування архітектури гри типу roguelike на основі Unreal Engine 5. Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація». 1-3 травня 2026 року, Київ, ДУІКТ. Збірник тез (подано в друк).
2. Бахмут М.С. Прототип адаптивної системи формування складності та нагород у roguelike-грі на базі Unreal Engine 5. VI Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». 15 травня 2026 року, Київ, ДУІКТ. Збірник тез (подано в друк).

1 ДОСЛІДЖЕННЯ ОБЛАСТІ РОЗРОБКИ ІГОР ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Вивчення предметної області

Комп’ютерні ігри представляють собою складні інтерактивні програмні системи, що поєднують у собі графічне відображення, обробку дій користувача, моделювання поведінки ігрових об’єктів, штучний інтелект, фізику, елементи інтерфейсу користувача, аудіо-супровід та логіку ігрового процесу. На відміну від багатьох традиційних програмних застосунків, ігри працюють у режимі реального часу та потребують постійного оновлення стану ігрового світу відповідно до дій гравця, поведінки ворогів, внутрішніх правил гри та випадкових подій.

Предметною областю даної кваліфікаційної роботи є розробка комп’ютерної гри жанру rogue-like на основі Unreal Engine 5 [14]. Ігри цього жанру характеризуються повторюваними ігровими сесіями, високою варіативністю проходження, випадковими нагородами, поступовим розвитком персонажа та необхідністю починати новий забіг після поразки [12]. У сучасних проєктах часто використовується також поняття roguelite, яке передбачає збереження основних ознак rogue-like, але допускає більш гнучку структуру ігрового процесу, бойову систему в реальному часі та часткову прогресію між забігами [12].

Основою ігрового процесу в rogue-like/roguelite-іграх є забіг. Забіг являє собою окрему ігрову сесію, під час якої гравець керує персонажем, бореться з ворогами, отримує досвід, золото, предмети або інші нагороди, а також поступово посилює свого персонажа [12]. У разі смерті персонажа або виконання кінцевої умови забіг завершується. Такий підхід створює циклічну структуру гри, де кожна нова спроба може відрізнятися від попередньої через випадковий вибір предметів, іншу послідовність подій, зміну складності або появу різних типів ворогів.

У rogue-like та roguelite-іграх розвиток персонажа зазвичай відбувається не одразу, а поступово, під час самої спроби проходження. На початку гравець має базові можливості, але після боїв, відкриття скринь або отримання нагород

ситуація змінюється. Персонаж може стати витривалішим, швидше атакувати, завдавати більше шкоди або краще відновлювати здоров’я. Через це одна й та сама арена не завжди відчувається однаково, навіть якщо вороги й загальні правила залишаються схожими. В одному забігу гравець більше покладається на швидкість і шкоду, в іншому – на запас здоров’я та виживання. Для MVP-версії такої гри цього вже достатньо, бо основна ідея жанру стає зрозумілою без складної системи постійного розвитку між забігами.

Окремо треба враховувати роль випадкових нагород. Саме вони роблять проходження менш передбачуваним і дають гравцю причину продовжувати забіг після кожної хвилі ворогів. Предмет може випасти після бою, бути отриманий зі скрині або стати нагородою за певну подію. При цьому важливо, щоб такі нагороди не виглядали повністю випадковими і несправедливими. Якщо сильні предмети з’являються занадто часто, гра швидко втрачає складність. Якщо ж корисні покращення майже не трапляються, гравець може не відчути розвитку персонажа. Тому предмети доцільно поділяти за рідкістю та впливом на характеристики. Такий підхід не ускладнює MVP надмірно, але дозволяє показати важливу рису жанру – кожен забіг може мати свій темп і трохи інший стиль проходження.

З технічної точки зору розробка гри такого типу потребує чіткого поділу на окремі підсистеми. До основних підсистем належать: система керування персонажем, бойова система, система характеристик, логіка ворогів, система появи ворогів, система предметів, нагороди, користувацький інтерфейс, умови перемоги та поразки. Якщо ці елементи реалізовувати як жорстко пов’язані частини одного Blueprint-класу, проєкт швидко стає складним для підтримки та розширення. Тому застосовувати модульний підхід, де кожна система чітко виконує свою визначену функцію.

Unreal Engine 5 є одним із сучасних ігрових рушіїв, який надає засоби для створення інтерактивних 3D-проєктів [1]. Для реалізації gameplay-логіки в Unreal Engine 5 може використовуватися як C++, так і Blueprint Visual Scripting. Blueprint-first підхід є доцільним для MVP та студентських проєктів, оскільки дозволяє швидко створювати, перевіряти та змінювати ігрові механіки [2]. Водночас

ефективне використання Blueprints потребує правильної архітектури, оскільки великі та неструктуровані Blueprint-графи ускладнюють тестування, налагодження та подальший розвиток гри.

У межах даної роботи предметна область охоплює створення прототипу кооперативної rogue-like гри з бойовою взаємодією в реальному часі, системою характеристик, ворогами, випадковими предметами, скринями, прогресією забігу та інтерфейсом користувача. Основна увага приділяється не лише реалізації окремих ігрових механік, а й побудові такої архітектури, яка дозволить надалі розширювати гру шляхом додавання нових ворогів, предметів, здібностей, ігрових подій та кооперативних можливостей.

1.2 Аналіз існуючих аналогів

Для визначення функціональних особливостей майбутньої гри доцільно проаналізувати існуючі проєкти жанру rogue-like/roguelite. Аналіз аналогів дозволяє визначити основні механіки, які формують ігровий цикл, впливають на повторюваність проходження та забезпечують зацікавленість гравця протягом багатьох ігрових сесій.

У межах роботи розглянуто декілька ігор, які мають різні підходи до реалізації roguelite-механік: Risk of Rain 2, Hades та Soulstone Survivors [17]**.** Ці проєкти обрано тому, що вони демонструють різні варіанти побудови ігрового процесу: тривимірний action roguelite з кооперативними можливостями, сюжетно орієнтований roguelite з системою нагород і аренно-хвильовий roguelite з великою кількістю ворогів.

1.2.1 Risk of Rain 2

Risk of Rain 2 є одним із найбільш релевантних аналогів для даної кваліфікаційної роботи, оскільки гра поєднує тривимірний ігровий простір, бойову систему в реальному часі, випадкові предмети, масштабування складності та кооперативний режим [15]. Основою ігрового процесу є проходження окремих

забігів, під час яких гравець знищує ворогів, отримує золото, відкриває скрині, збирає предмети та поступово посилює персонажа.

Важливою особливістю Risk of Rain 2 є система предметів. Предмети не просто збільшують окремі характеристики, а можуть суттєво змінювати стиль гри. Наприклад, одні предмети підвищують шкоду, інші – швидкість атаки, виживаність, мобільність або додають додаткові ефекти під час атаки. Завдяки цьому кожен забіг має іншу динаміку, навіть якщо гравець використовує одного й того самого персонажа.

Ще однією важливою механікою є масштабування складності з часом. Чим довше триває забіг, тим сильнішими стають вороги. Це створює постійний тиск на гравця та змушує його швидко приймати рішення: досліджувати рівень для пошуку додаткових предметів або швидше активувати подію з босом і переходити далі. Такий підхід добре підтримує темп гри та не дозволяє гравцю залишатися в безпечному стані занадто довго.

Для даної роботи найбільш корисними ідеями з Risk of Rain 2 є:

- система скринь як джерела випадкових предметів;
- отримання золота за перемогу над ворогами;
- предмети як основний спосіб посилення персонажа;
- поступове ускладнення гри;
- структура забігу з фінальною ціллю або босом;
- можливість подальшого розширення гри до кооперативного формату.

Рис. 1.1 Ігровий процес у Risk of Rain 2

Рис. 1.2 Система предметів та скринь у Risk of Rain 2

У розроблюваному проєкті ці ідеї можуть бути адаптовані у спрощеному вигляді: гравець перемагає ворогів, отримує золото, відкриває скрині, отримує випадкові предмети, посилює характеристики та завершує забіг після виконання кінцевої умови.

1.2.2 Hades

Hades є прикладом roguelite-гри, у якій значну роль відіграють повторювані забіги, вибір нагород, розвиток персонажа та поступове відкриття нового контенту [16]. На відміну від Risk of Rain 2, Hades має більш структурований поділ на окремі кімнати, у яких гравець перемагає ворогів і після цього отримує нагороду або обирає подальший шлях.

Однією з ключових особливостей Hades є система вибору покращень. Після проходження певних етапів гравець отримує тимчасові бонуси, які діють у межах поточного забігу. Ці бонуси можуть змінювати атаки, додавати додаткові ефекти, покращувати захист або впливати на інші аспекти бойової системи. Завдяки цьому кожен забіг має власний набір можливостей, а гравець може адаптувати стиль гри до отриманих покращень.

Також важливою є наявність метапрогресії. Навіть якщо гравець програє, він поступово відкриває нові можливості, покращення або сюжетні елементи. Це зменшує негативне відчуття від поразки та мотивує починати нові спроби. Для

MVP-версії розроблюваної гри повна система метапрогресії може бути надмірною, однак її можна розглядати як перспективу подальшого розвитку.

З Hades для розроблюваної гри можна використати такі ідеї:

- забіг як окрема завершена ігрова сесія;
- нагороди, що змінюють стиль проходження;
- поступове посилення персонажа в межах забігу;
- зрозумілу структуру перемоги та поразки;
- мотивацію до повторного проходження.

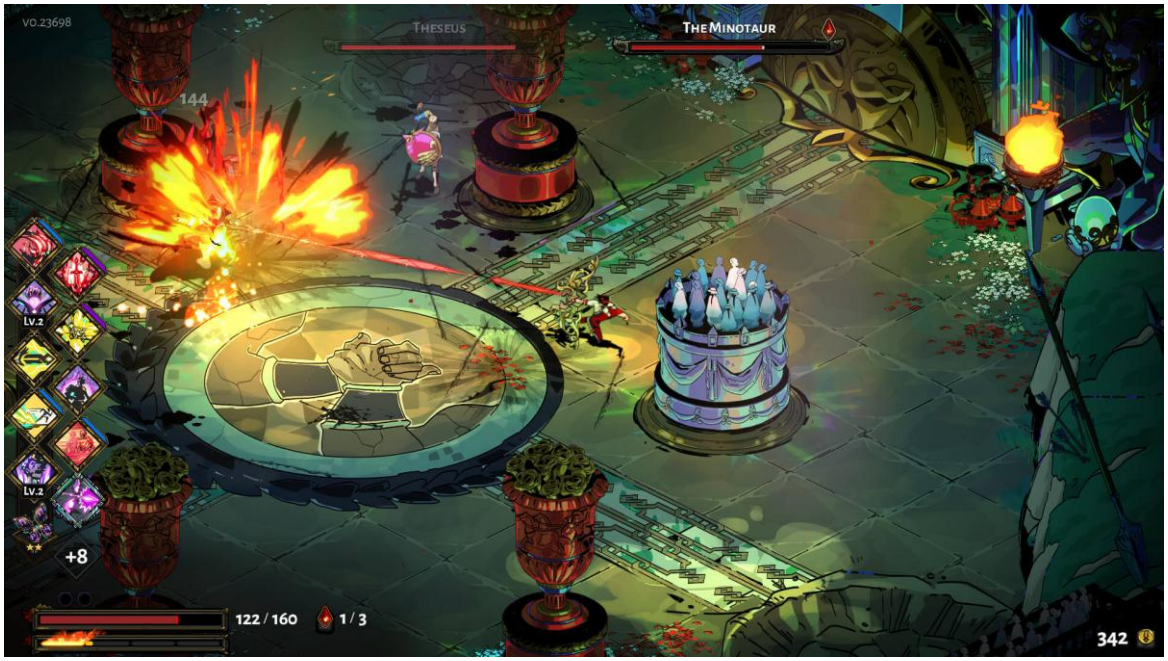


Рис. 1.3 Приклад бойової сцени та інтерфейсу в Hades

У межах даної роботи ці принципи можуть бути реалізовані через систему предметів, які змінюють характеристики персонажа, та через завершення забігу після перемоги над босом або смерті гравця.

1.2.3 Soulstone Survivors

Soulstone Survivors є прикладом action roguelite-гри з аренною структурою, великою кількістю ворогів, хвильовим ігровим процесом і постійним розвитком персонажа під час бою [18]. На відміну від ігор із кімнатною структурою, тут основна дія відбувається на відкритій арені, де гравець постійно переміщується, уникає атак і знищує велику кількість противників.

Основною особливістю Soulstone Survivors є інтенсивний бойовий цикл. Вороги з’являються хвилями або постійним потоком, а гравець повинен виживати, покращувати свої здібності та адаптуватися до зростання складності. Такий підхід добре підходить для MVP, оскільки не потребує складної процедурної генерації рівнів і дозволяє сфокусуватися на бойовій системі, характеристиках, ворогах, предметах та балансі.

Ще однією важливою рисою є поступове зростання сили персонажа. У процесі гри гравець отримує нові покращення, які впливають на шкоду, швидкість, область ураження, частоту атак або інші параметри. Це створює відчуття розвитку та дозволяє гравцю поступово переходити від слабкого початкового стану до сильного персонажа, здатного знищувати великі групи ворогів.

Для розроблюваної гри з Soulstone Survivors доцільно використати такі ідеї:

- арена як основний простір гри;
- хвильова або поступова поява ворогів;
- швидкий бойовий цикл;
- розвиток персонажа під час забігу;
- поступове підвищення складності;
- фокус на виживанні та контролі простору.

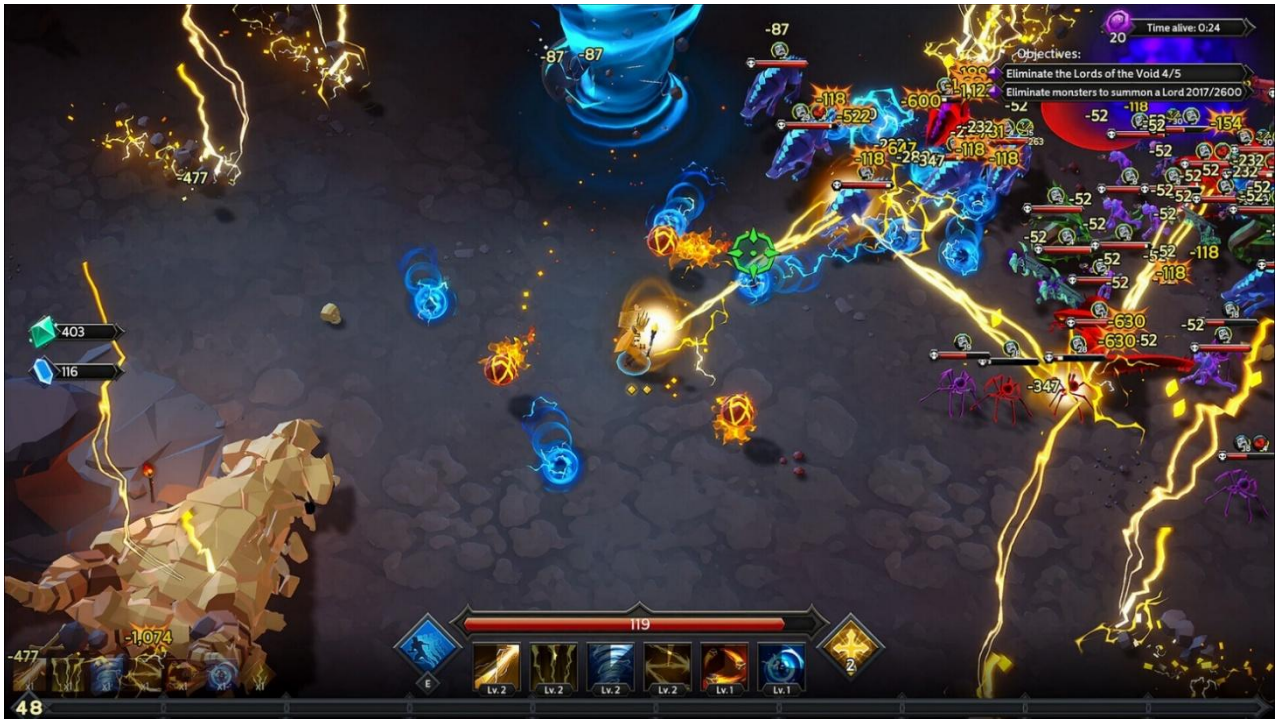


Рис. 1.4 Ігровий процес у Soulstone Survivors

Цей аналог є корисним саме з точки зору MVP, оскільки арена дозволяє швидше створити працездатний ігровий цикл без необхідності реалізовувати складну систему процедурних рівнів.

1.2.4 Постановка задачі розробки

На основі аналізу ігор Risk of Rain 2, Hades та Soulstone Survivors можна визначити ключові механіки, які є доцільними для використання під час розробки власного проєкту. Кожна з розглянутих ігор реалізує різний підхід до побудови roguelike/roguelite-геймплею, однак усі вони мають спільні риси: повторюваність забігів, розвиток персонажа в межах ігрової сесії, випадкові або варіативні нагороди, бойову взаємодію та поступове підвищення складності.

Для порівняння основних особливостей розглянутих аналогів сформовано таблицю 1.1

Таблиця 1.1

Порівняльна характеристика ігор-аналогів

Критерій порівняння	Risk of Rain 2	Hades	Soulstone Survivors
Жанровий напрям	3D action roguelite	Action roguelite	Arena action roguelite
Основна структура гри	Забіг через рівні з босами та предметами	Послідовність бойових кімнат	Арена з хвилями ворогів
Тип бойової системи	Бій у реальному часі від третьої особи	Бій у реальному часі з ізометричною камерою	Бій у реальному часі на арені
Основний спосіб розвитку персонажа	Предмети, що змінюють характеристики та ефекти	Тимчасові покращення та здібності	Покращення характеристик і здібностей
Система випадковості	Випадкові предмети, скрині, події	Випадкові нагороди після кімнат	Випадкові покращення та хвилі ворогів
Масштабування складності	Зростання складності з часом	Поступове ускладнення через етапи забігу	Зростання кількості та сили ворогів
Наявність кооперативу	Передбачений кооперативний режим	Відсутній	Переважно singleplayer
Корисність для власного проєкту	Основний референс для предметів, скринь, золота, босів і кооперативного потенціалу	Референс для повторюваності забігів і системи нагород	Референс для арени, хвиль ворогів і швидкого бойового циклу

Risk of Rain 2 є найближчим аналогом до розроблюваної гри, оскільки поєднує тривимірний простір, бойову систему в реальному часі, отримання золота, відкриття скринь, випадкові предмети та поступове ускладнення гри. Особливо важливою для даної роботи є система предметів, оскільки саме вона формує варіативність проходження та дозволяє створювати різні комбінації покращень у межах одного забігу.

Hades демонструє інший підхід до структури забігу. Гра побудована на послідовному проходженні окремих бойових кімнат, після яких гравець отримує нагороди або покращення. Для розроблюваного проєкту корисною є не сама кімнатна структура, а принцип поступового посилення персонажа в межах забігу, а також ідея повторюваності проходжень із різними наборами покращень.

Soulstone Survivors є корисним прикладом аренного roguelite-підходу, де основна увага приділяється виживанню, великій кількості ворогів, хвилям противників і швидкому темпу бойової взаємодії. Цей підхід є зручним для MVP-версії гри, оскільки дозволяє створити повноцінний ігровий цикл без складної процедурної генерації рівнів.

1.3 Визначення вимог до гри

Після аналізу предметної області та існуючих аналогів було визначили основні вимоги до розроблюваного проєкту. Оскільки проєкт створюється як MVP-версія гри жанру rogue-like/roguelite, вимоги мають бути сформовані з пріоритетом на реалізацію основного циклу гри, а вже після можна фокусуватись на комерційні питання та збільшення контенту. Основна увага приділяється працездатності ключових механік ігрового циклу: керуванню персонажем, бойовій взаємодії, системі характеристик, ворогам, предметам, нагородам, інтерфейсу користувача та завершенню забігу.

Кінцевий варіант проєкту повинна мати завершений ігровий цикл, у якому гравець починає гру, бореться з ворогами, отримує нагороди, посилює свого персонажа за допомогою предметів, рівня і завершує забіг перемогою або

поразкою. Таким чином ми можем перевірити основні принципи жанру rogue-like/roguelite у межах обмеженого за обсягом проєкту.

Вимоги до гри можна поділити на нефункціональні а також функціональні. Вимоги навиваються функціональними тому що характеризують конкретні можливості, які повинна мати гра як наприклад базовий цикл гри. Якщо казати про нефункціональні вимоги то вони зосереджуються на більш якісні аспекти програмних систем: здібність і зручність розширення, підтримуваність, стабільність, зрозумілість структури та можливість подальшого розвитку.

Функціональні вимоги до розроблюваної гри наведено в таблиці 1.2

Таблиця 1.2

Функціональні вимоги до розроблюваної гри

№	Вимога	Опис
1	Керування персонажем	Гравець повинен мати можливість переміщувати персонажа ігровою ареною та виконувати базові дії.
2	Бойова система	Гравець повинен мати можливість атакувати ворогів і завдавати їм шкоди.
3	Система здоров’я	Персонаж і вороги повинні мати показники здоров’я, отримувати шкоду та помирати при досягненні нульового значення здоров’я.
4	Система характеристик	У грі повинні бути реалізовані основні характеристики, зокрема здоров’я, шкода, броня, швидкість руху, швидкість атаки та регенерація.
5	Вороги	У грі повинні бути вороги, які з’являються на арені, переслідують гравця та можуть завдавати йому шкоди.
6	Система появи ворогів	Гра повинна підтримувати появу ворогів через точки спавну або систему хвиль.
7	Нагороди за перемогу над ворогами	Після знищення ворогів гравець повинен отримувати ігрові ресурси, наприклад золото або досвід.
8	Система рівня	Гравець повинен мати можливість отримувати досвід і підвищувати рівень персонажа.
9	Система скринь	У грі повинні бути інтерактивні скрині, які гравець може відкривати за ігрову валюту.
10	Система предметів	Після відкриття скрині гравець повинен отримувати випадковий предмет, який змінює характеристики персонажа.
11	Рідкість предметів	Предмети повинні мати різну рідкість, наприклад Common, Rare та Legendary.
12	Інтерфейс користувача	Гравець повинен бачити основні показники: здоров’я, золото, рівень, характеристики та отримані предмети.
13	Завершення забігу	Забіг повинен завершуватися у разі смерті гравця або виконання умови перемоги.
14	Перезапуск гри	Після завершення забігу гравець повинен мати можливість почати нову спробу.

Основною вимогою є наявність повного циклу забігу. Для MVP достатньо реалізувати одну ігрову арену, базового персонажа, декілька типів ворогів, систему скринь, випадкові предмети та екран завершення гри. Розширення кількості персонажів, здібностей, ворогів, босів і рівнів може бути виконане на наступних етапах розробки.

Нефункціональні вимоги до гри наведено в таблиці 1.3

Таблиця 1.3

Нефункціональні вимоги до розроблюваної гри

№	Вимога	Опис
1	Модульність	Основні gameplay-системи повинні бути розділені між окремими Blueprint-класами та компонентами.
2	Підтримуваність	Структура проєкту повинна бути зрозумілою для подальшого редагування та виправлення помилок.
3	Масштабованість	Проєкт повинен дозволяти додавати нові предмети, ворогів, характеристики та ігрові події без повної переробки архітектури.
4	Data-driven підхід	Параметри предметів, характеристик і нагород бажано зберігати у структурах даних, таблицях або data assets.
5	Мінімізація дублювання логіки	Повторювана логіка, наприклад робота зі здоров'ям і характеристиками, повинна бути винесена в окремі компоненти.
6	Стабільність роботи	Основні механіки гри повинні працювати без критичних помилок під час проходження забігу.
7	Зручність тестування	Реалізовані системи повинні бути достатньо розділеними, щоб їх можна було перевіряти окремо.
8	Можливість подальшого розвитку	Архітектура гри повинна враховувати можливість додавання кооперативного режиму, нових класів персонажів і нових типів контенту.
9	Оптимізація Blueprint-логіки	Необхідно уникати зайвої логіки на Tick там, де можна використати події, таймери, overlap-події або dispatcher-и.
10	Відокремлення UI від gameplay-логіки	Інтерфейс користувача повинен відображати дані, але не керувати основними правилами гри.

Особливо важливою нефункціональною вимогою є модульність. У випадку Blueprint-first розробки надмірне накопичення логіки в одному Blueprint-класі ускладнює підтримку проєкту.

Для системи характеристик доцільно використовувати окремий Actor Component, який відповідає за здоров'я, шкоду, броню, швидкість руху, швидкість атаки, регенерацію та обробку отриманої шкоди. Такий підхід дозволяє використовувати одну й ту саму логіку як для персонажа гравця, так і для ворогів.

Окремою вимогою є використання data-driven підходу для системи предметів. Предмети повинні описуватися через структури, таблиці або data assets, де вказується назва предмета, його рідкість, тип ефекту та числові значення. Це дозволяє змінювати баланс гри без зміни основної логіки Blueprint-класів.

Також важливо відокремити gameplay-логіку від користувацького інтерфейсу. Інтерфейс повинен лише отримувати актуальні значення здоров’я, золота, рівня, характеристик та предметів і відображати їх гравцю. Основні правила гри, такі як отримання нагород, застосування предметів або завершення забігу, не повинні реалізовуватися безпосередньо у Widget Blueprint.

Таким чином, визначені вимоги формують основу для подальшого проєктування та реалізації гри. Функціональні вимоги задають перелік механік, які необхідно реалізувати в межах MVP, а нефункціональні вимоги визначають архітектурні обмеження, необхідні для підтримуваності та подальшого розвитку проєкту.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ГРИ НА ОСНОВІ UNREAL ENGINE 5

2.1 Формування функціональних та нефункціональних вимог до гри

Перед початком реалізації гри необхідно визначити вимоги до програмної системи. Це дозволяє краще зрозуміти, які основні функції має виконувати гра, які підсистеми потрібно спроєктувати та які обмеження слід враховувати під час розробки.

Розроблювана гра належить до жанру rogue-like/roguelite. Для таких ігор характерні короткі ігрові сесії, поступове посилення персонажа, випадкові нагороди, висока повторюваність проходження та наявність ризику програшу під час забігу [12]. Основна ідея проєкту полягає у створенні прототипу гри, де гравець керує персонажем, бореться з ворогами, отримує ресурси, відкриває нагороди, посилює характеристики та завершує забіг перемогою або поразкою.

Функціональні вимоги описують можливості, які повинна мати гра.

Функціональні вимоги до гри:

- керування персонажем у реальному часі;
- можливість атаки ворогів;
- отримання шкоди персонажем і ворогами;
- система здоров’я, шкоди, броні та інших характеристик;
- поява ворогів на ігровій арені;
- отримання золота або досвіду за знищення ворогів;
- відкриття інтерактивних об’єктів, наприклад скринь;
- отримання випадкових предметів;
- зміна характеристик персонажа через предмети;
- відображення здоров’я, золота, рівня та інших параметрів в інтерфейсі;
- можливість виклику фінальної події або боса;
- завершення забігу через перемогу або поразку;
- можливість перезапуску забігу.

Нефункціональні вимоги описують не окремі функції, а загальні властивості системи. Для гри важливо, щоб основні механіки працювали стабільно, були зрозумілими для гравця та могли розширюватися в майбутньому.

Нефункціональні вимоги:

- стабільна робота основних gameplay-систем;
- зрозуміла структура проєкту;
- можливість розширення гри новими ворогами, предметами та подіями;
- відокремлення логіки персонажа, ворогів, предметів, інтерфейсу та правил забігу;
- зручність налагодження Blueprint-логіки;
- мінімізація дублювання однакової логіки;
- підготовка архітектури до можливого кооперативного режиму;
- коректне оновлення інтерфейсу під час зміни параметрів гравця.

Для реалізації таких вимог доцільно використовувати модульну структуру. У грі не варто розміщувати всю логіку в одному Blueprint-класі, оскільки це ускладнює підтримку, тестування та подальший розвиток проєкту. Тому основні частини гри мають бути розділені між персонажами, компонентами, системою забігу, інтерфейсом і таблицями даних.

2.2 Обґрунтування вибору середовища та засобів розробки

Для розробки гри було обрано Unreal Engine 5. Даний рушій призначений для створення інтерактивних тривимірних застосунків і комп’ютерних ігор, а також містить готові інструменти для роботи з персонажами, фізикою, анімаціями, інтерфейсом, колізіями та ігровою логікою [20].

Використання готового рушія є доцільним, оскільки метою роботи є не створення власного ігрового рушія, а розробка працездатного прототипу гри. Unreal Engine 5 дозволяє зосередитися на побудові ігрових механік: керуванні персонажем, бойовій системі, ворогах, предметах, прогресії та завершенні забігу.

Основні засоби розробки:

- Unreal Engine 5 – основне середовище створення гри;
- Blueprint Visual Scripting – інструмент реалізації gameplay-логіки;
- Gameplay Framework – основа для поділу ролей між персонажем, контролером, режимом гри та станом гравця;
- Enhanced Input – система обробки введення;
- Actor Components – засіб винесення повторюваної логіки в окремі модулі;
- UMG – система створення користувацького інтерфейсу;
- Data Tables або Data Assets – засіб зберігання параметрів предметів, ворогів та інших ігрових даних.

Однією з причин вибору Unreal Engine 5 є наявність Blueprint Visual Scripting. Blueprints дозволяють реалізовувати логіку гри без обов'язкового написання коду мовою C++, що є зручним для швидкого створення MVP та перевірки ігрових механік [2].

При цьому Blueprint-first підхід потребує правильної організації. Якщо всю логіку розміщувати лише в персонажі або в одному великому Blueprint-графі, проєкт швидко стає складним для розуміння. Тому під час проєктування передбачено розділення відповідальності між окремими класами, компонентами та структурами даних.

Enhanced Input використовується для налаштування дій гравця. Через нього можна описати рух, атаку, взаємодію з об'єктами та подальше розширення керування здібностями. Такий підхід є зручним, оскільки дозволяє гнучко змінювати схему введення без повної переробки логіки персонажа [4].

UMG використовується для створення інтерфейсу користувача. За допомогою віджетів можна відображати здоров'я, золото, рівень, предмети та результат завершення забігу [7].

Для опису параметрів предметів і нагород доцільно використовувати data-driven підхід. Це означає, що значення предметів, рідкість, тип ефекту та числові параметри зберігаються в окремих структурах або таблицях, а не прописуються вручну в логіці кожного об'єкта [5].

2.3 Проєктування загальної структури гри

Загальна структура гри повинна забезпечувати поділ систем за призначенням. Це знадобиться коли треба буде змінювати різні частини гри без повної переробки всього проєкту. Для узагальнення взаємодії основних gameplay-систем було побудовано IDEF0-модель, яка показує вхідні дані, керуючі правила, механізми реалізації та результати виконання ігрового забігу.

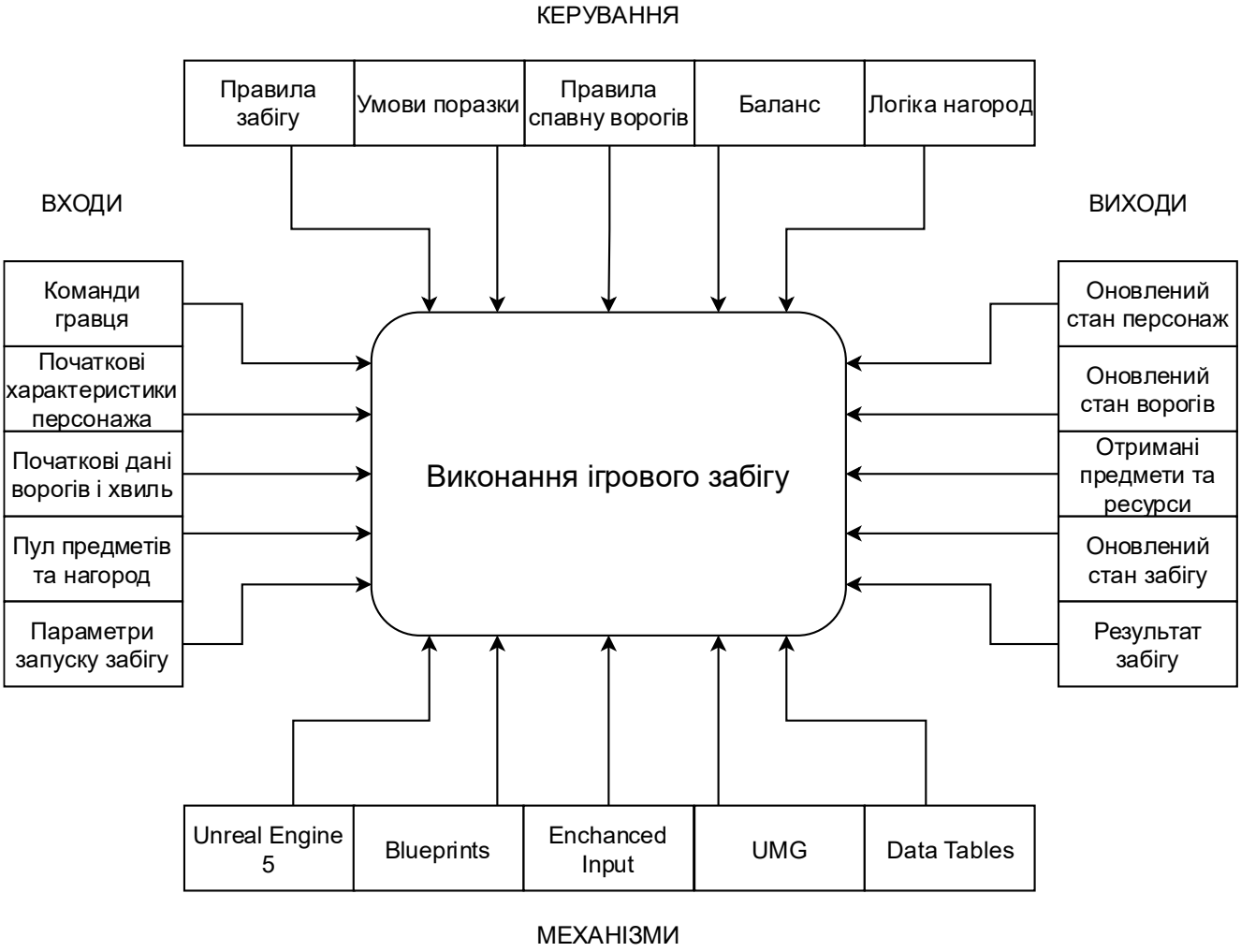


Рис. 2.1 IDEF0-модель загальної структури ігрового процесу

З наведеної моделі видно, що виконання ігрового забігу залежить від дій гравця, параметрів персонажа, даних ворогів, предметів і налаштувань забігу. Керування процесом здійснюється через правила забігу, умови смерті, баланс і

логіку нагород. Механізмами реалізації виступають Unreal Engine 5, Blueprints Visual Scripting, Enhanced Input, UMG та таблиці даних.

Для деталізації внутрішньої структури було побудовано діаграму Blueprint-класів основних gameplay-систем. Вона показує, які класи відповідають за керування забігом, персонажа гравця, ворогів, характеристики, систему появи ворогів, інтерфейс, інтерактивні об’єкти та data-driven частину предметів.

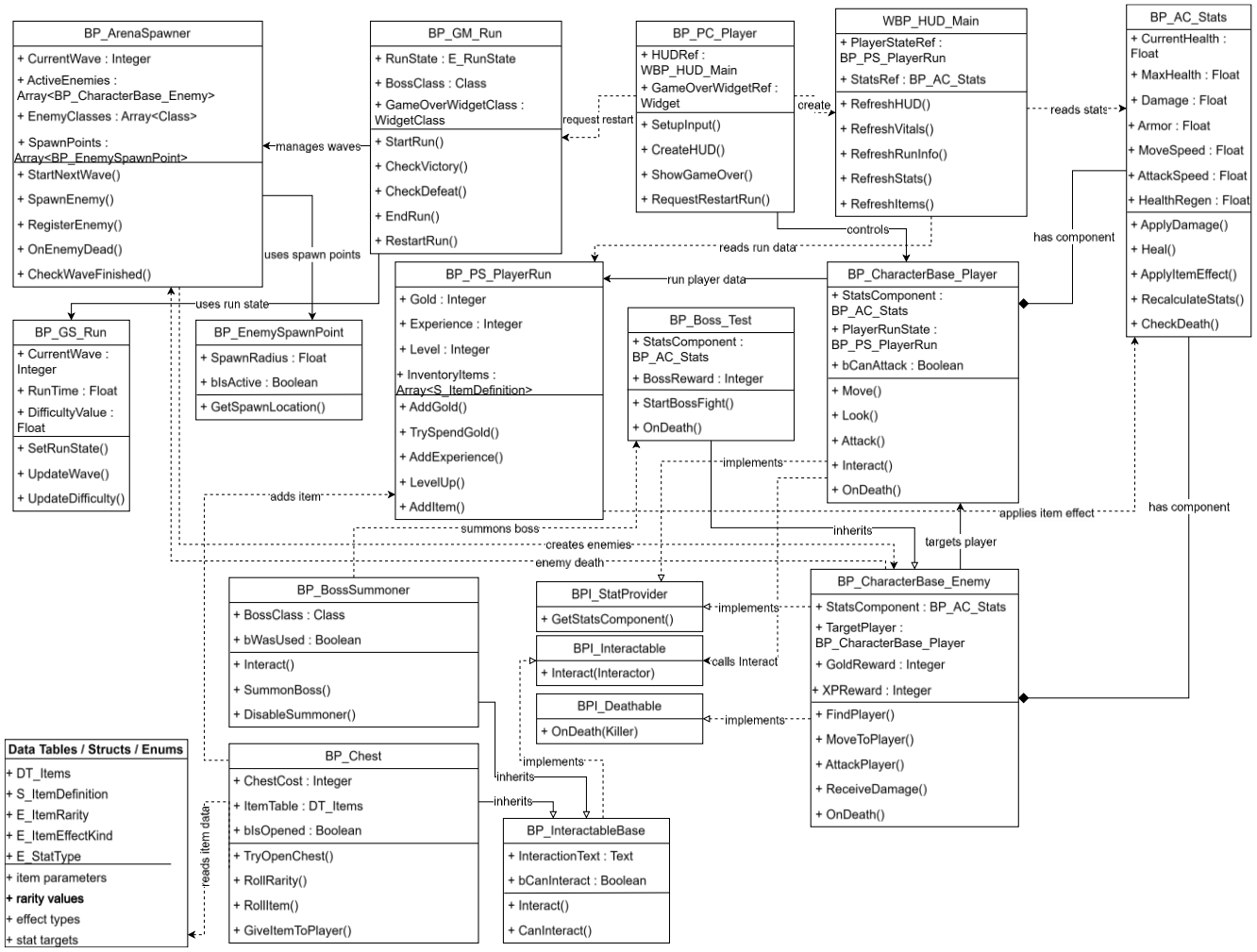


Рис. 2.2 UML-діаграма Blueprint-класів основних ігрових систем

Згідно з діаграмою, загальні правила забігу винесені в BP_GM_Run. Цей клас відповідає за запуск гри, перевірку перемоги або поразки, завершення забігу та перезапуск. Спільний стан забігу може зберігатися в BP_GS_Run, що є корисним для подальшого розвитку кооперативного режиму.

Персонаж гравця реалізується через BP_CharacterBase_Player. Він відповідає за переміщення, отримання введення, виконання атак і взаємодію з об’єктами.

Вороги реалізуються через BP_CharacterBase_Enemy, який відповідає за пошук гравця, переслідування, атаку, отримання шкоди та смерть. Окремий BP_Boss_Test може наслідувати базову логіку ворога, але використовуватися для фінальної події забігу.

Спільні характеристики гравця, ворогів і боса винесені в компонент BP_AC_Stats. Він зберігає основні бойові параметри, зокрема здоров'я, шкоду, броню, швидкість руху, швидкість атаки та регенерацію. Такий підхід зменшує дублювання, оскільки не потрібно створювати окрему систему здоров'я для кожного типу ігрової сутності.

Дані конкретного гравця в межах забігу зберігаються в BP_PS_PlayerRun. До них належать золото, досвід, рівень та список отриманих предметів. Інтерфейс WBP_HUD_Main не керує ігровою логікою напряму, а лише читає дані з BP_PS_PlayerRun і BP_AC_Stats для відображення здоров'я, характеристик, золота, досвіду та рівня.

Для створення ворогів використовується BP_ArenaSpawner, який працює з BP_EnemySpawnPoint, створює ворогів і контролює список активних противників. Інтерактивні об'єкти, такі як BP_Chest і BP_BossSummoner, можуть наслідувати BP_InteractableBase та використовувати спільний принцип взаємодії.

Для зменшення жорстких залежностей між класами доцільно використовувати Blueprint Interface. Інтерфейси дозволяють різним об'єктам взаємодіяти через спільний набір функцій, не прив'язуючись до конкретного Blueprint-класу [6]. У структурі передбачено використання BPI_Interactable, BPI_StatProvider та BPI_Deathable, які відповідають за взаємодію з об'єктами, отримання компонента характеристик і обробку смерті.У такій структурі скриня, призиватель боса або інший інтерактивний об'єкт можуть використовувати однаковий принцип взаємодії. Це дозволяє додавати нові об'єкти без повної зміни логіки персонажа.

2.4 Проєктування бойової системи та системи характеристик

Бойова система є однією з головних частин гри. Вона визначає, як персонаж атакує ворогів, як вороги завдають шкоди гравцю, як змінюється здоров’я та що відбувається після смерті ігрової сутності.

Для MVP достатньо реалізувати просту, але розширювану систему. Атака може працювати через перевірку області перед персонажем або через колізію. Головне, щоб шкода не була жорстко прописана в одному місці, а бралася з системи характеристик. Це дозволяє використовувати однаковий принцип для гравця, ворогів і боса.

Загальний алгоритм бойової взаємодії наведено на рис. 2.3

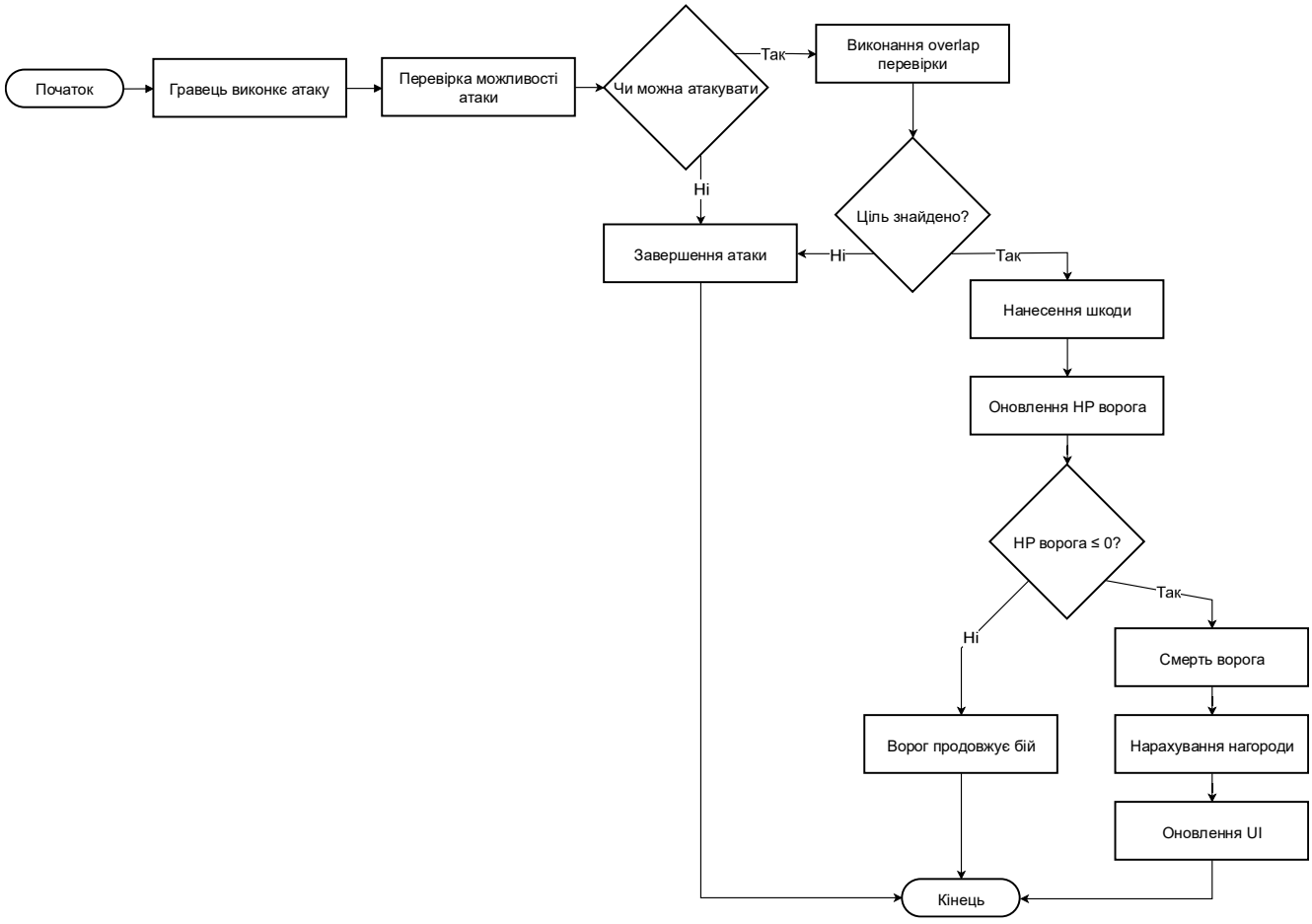


Рис. 2.3 Блок схема бойової системи

Відповідно до наведеної діаграми, бойова взаємодія починається з виконання атаки. Після цього система перевіряє можливість виконання атаки, визначає наявність цілі, наносить шкоду та оновлює значення здоров’я. Якщо здоров’я ворога стає меншим або дорівнює нулю, запускається логіка смерті, нарахування нагороди та оновлення інтерфейсу.

Компонент характеристик відповідає за зберігання та зміну основних бойових параметрів: поточного здоров’я, максимального здоров’я, шкоди, броні, швидкості руху, швидкості атаки та регенерації. Саме цей компонент повинен обробляти отримання шкоди, оновлення здоров’я та перевірку умови смерті.

Швидкість атаки використовується для розрахунку інтервалу між атаками. Наприклад, якщо швидкість атаки дорівнює 1, персонаж може атакувати приблизно один раз на секунду. Якщо значення збільшується, інтервал між атаками зменшується.

Для стабільності системи доцільно передбачити мінімальний інтервал атаки. Це потрібно не для штучного обмеження гравця, а для захисту від технічних ситуацій, коли надто велике значення швидкості атаки може створити надмірне навантаження.

Смерть гравця не повинна напряму завершувати гру всередині персонажа. Краще, щоб персонаж або компонент характеристик повідомляв систему забігу, а вже вона перевіряла умову поразки. Це важливо для майбутнього кооперативного режиму, де смерть одного гравця не завжди означає завершення всієї сесії.

2.5 Проєктування системи предметів, нагород і прогресії

Система предметів і нагород створює основну варіативність забігу. У roguelite-грі гравець повинен поступово посилюватися, отримуючи випадкові покращення, ресурси або інші переваги.

Золото використовується як ресурс поточного забігу. Гравець може отримувати його після знищення ворогів і витрачати на відкриття скринь. Досвід використовується для підвищення рівня персонажа. Рівень може збільшувати

базові характеристики, а в майбутньому може бути використаний для відкриття додаткових можливостей або складніших варіантів розвитку.

Загальний алгоритм роботи системи прогресії та нагород наведено на рис. 2.4

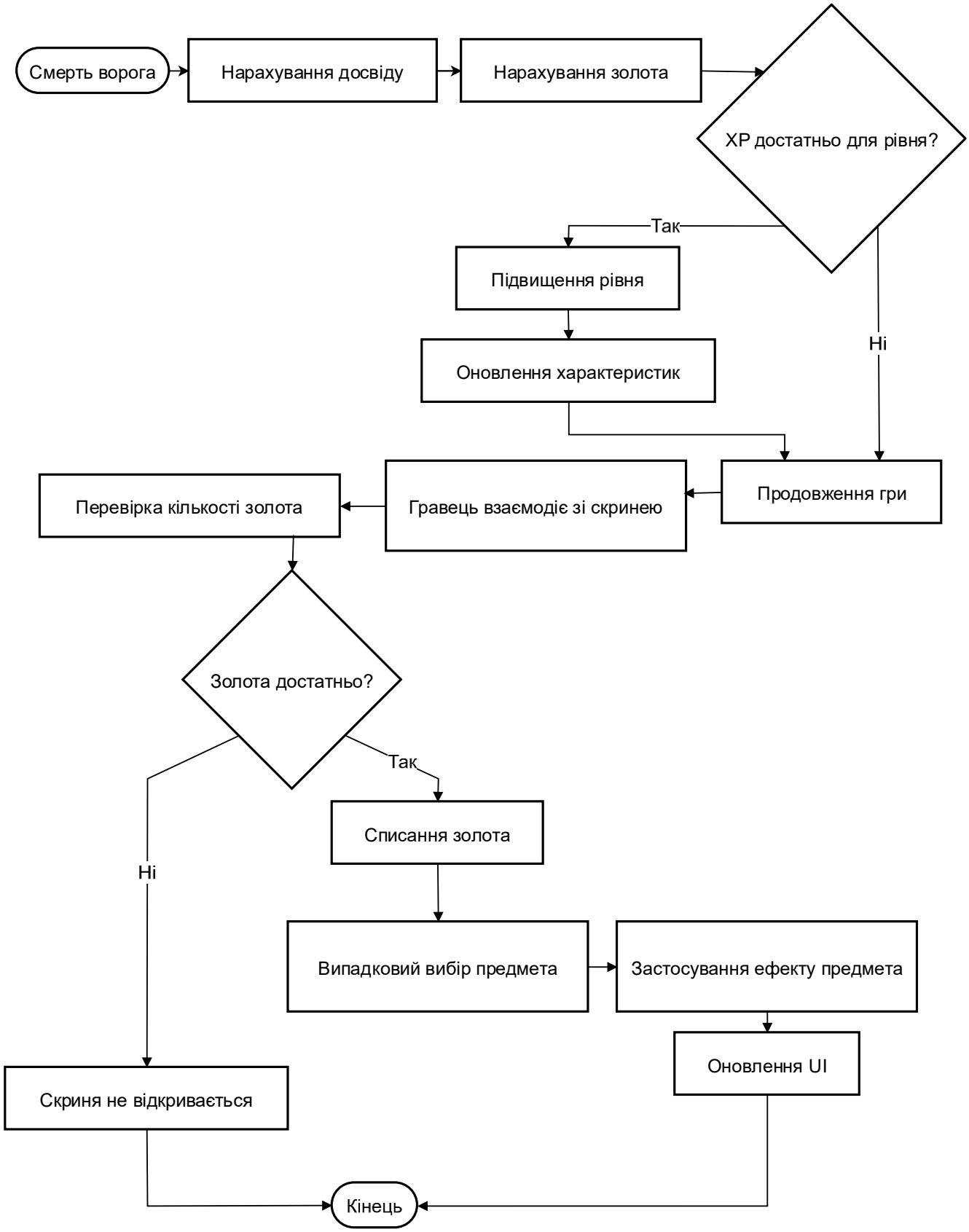


Рис. 2.4 Блок схема системи прогресії та нагород

З діаграми видно, що після смерті ворога гравець отримує досвід і золото. За достатньої кількості досвіду підвищується рівень персонажа та оновлюються його характеристики. Також передбачено відкриття скрині: система перевіряє наявність золота, списує його, випадково обирає предмет, застосовує його ефект і оновлює інтерфейс.

Предмети доцільно зберігати в таблиці або окремому наборі даних. Для кожного предмета можна визначити назву, опис, рідкість, тип ефекту, характеристику для зміни, числове значення та іконку. Це спрощує балансування і дозволяє змінювати предмети без переробки логіки скрині.

Рідкість предметів потрібна для керування шансами випадіння. Наприклад, звичайні предмети можуть випадати часто, рідкісні – рідше, а легендарні – з мінімальною ймовірністю. Це дозволяє створити просту систему випадкових нагород.

Для rogue-like/roguelite гри важливо, щоб предмети не були повністю зашиті в логіку скрині. Скриня повинна лише запускати процес видачі нагороди. Самі параметри предметів мають зберігатися окремо. Такий підхід спрощує балансування і дозволяє додавати нові предмети без переписування основної логіки.

Окремою частиною прогресії є фінальна подія. У межах MVP це може бути виклик боса або іншої складної події. Після перемоги над фінальним ворогом система забігу може завершити гру перемогою.

2.6 Проєктування базового інтерфейсу гравця

Інтерфейс гравця є основним джерелом інформації для відображення стану гравця під час забігу. У грі реального часу гравець повинен швидко бачити основні параметри, не відкриваючи додаткових меню.

Основні елементи інтерфейсу:

- шкала здоров’я;

- поточний рівень;
- кількість золота;
- досвід або прогрес рівня;
- список отриманих предметів;
- панель характеристик;
- екран завершення гри.

Інтерфейс не повинен бути основним місцем зберігання даних. Його задача – отримувати актуальні значення з інших систем і показувати їх гравцю. Наприклад, здоров’я має зберігатися в компоненті характеристик, золото і досвід – у стані гравця, а результат забігу – у системі правил гри.

Оновлення інтерфейсу може відбуватися після зміни відповідного значення. Для MVP достатньо створити окремі функції оновлення здоров’я, золота, досвіду та предметів. Надалі це можна покращити через події або диспетчери подій, щоб UI оновлювався тільки тоді, коли значення справді змінилося.

Логіка завершення забігу також має бути відокремлена від UI. Інтерфейс лише показує результат, але не повинен сам вирішувати, коли гра завершилася.

Для майбутнього кооперативного режиму важливо, щоб система завершення не була прив’язана тільки до одного персонажа. Якщо один гравець загинув, система повинна перевірити стан інших учасників. Поразка має наставати лише тоді, коли всі активні гравці більше не можуть продовжувати гру або виконана інша глобальна умова поразки [8].

Таким чином, у другому розділі було спроєктовано загальну архітектуру гри, визначено основні функціональні та нефункціональні вимоги, обґрунтовано вибір Unreal Engine 5, описано структуру gameplay-систем, систему характеристик, предметів, інтерфейсу та завершення забігу. Детальна реалізація конкретних Blueprint-класів, створених функцій, змінних, інтерфейсів і взаємодій буде розглянута в наступному розділі.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ GAMEPLAY-СИСТЕМ ГРИ В UNREAL ENGINE 5

3.1 Створення структури проєкту та основних Blueprint-класів

Після етапу проєктування було виконано практичну реалізацію MVP гри в середовищі Unreal Engine 5. Основною задачею цього розділу є опис того, як спроєктовані в попередньому розділі системи були реалізовані на практиці.

Розробка виконувалась за Blueprint-first підходом. Це означає, що основна gameplay-логіка реалізовувалась за допомогою Blueprint Visual Scripting, а не через написання C++ коду [19]. Такий підхід був обраний через те, що для MVP важливо швидко перевіряти механіки, змінювати параметри, налагоджувати поведінку об’єктів і поступово розширювати систему без складної компіляції проєкту [2].

На початку реалізації була створена базова структура папок у проєкті. Вона потрібна для того, щоб Blueprint-класи, інтерфейси, віджити, таблиці даних і рівні не були змішані в одному місці.

Основна структура проєкту має такий вигляд:

1. Blueprints/Core – основні Blueprint-класи ігрової логіки.
2. Blueprints/Core/Characters – персонаж гравця, вороги та бос.
3. Blueprints/Core/Components – компоненти характеристик та іншої повторюваної логіки.
4. Blueprints/Core/Run – об’єкти, пов’язані із забігом, ареною та спавном.
5. Blueprints/Core/Interactables – інтерактивні об’єкти.
6. Blueprints/Core/Progression – GameMode, GameState та логіка прогресії.
7. Blueprints/Interfaces – Blueprint Interface для взаємодії між системами.
8. Data – структури, enum-и та таблиці даних.
9. UI – віджети інтерфейсу користувача.
10. Input – налаштування Enhanced Input.
11. Maps – рівні гри.

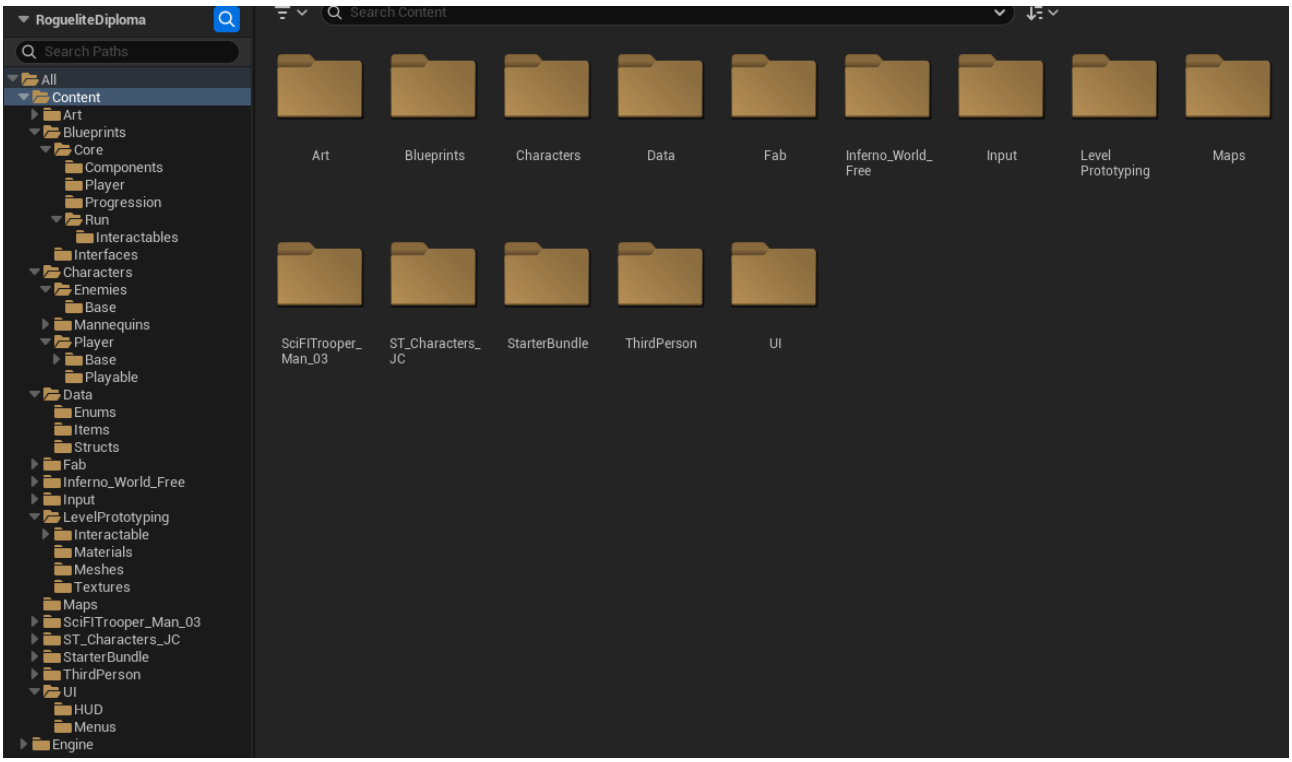


Рис. 3.1 Структура папок проєкту в Unreal Engine 5

Для реалізації основного gameplay було створено декілька ключових Blueprint-класів. Вони виконують різні ролі, але працюють як частини однієї системи.

Основні Blueprint-класи проєкту:

1. BP_CharacterBase_Player – базовий персонаж гравця.
2. BP_CharacterBase_Enemy – базовий клас ворога.
3. BP_AC_Stats – компонент характеристик.
4. BP_PC_Player – контролер гравця.
5. BP_PS_PlayerRun – стан гравця в межах забігу.
6. BP_GM_Run – режим гри для керування забігом.
7. BP_GS_Run – загальний стан гри.
8. BP_ArenaSpawner – система появи ворогів.
9. BP_EnemySpawnPoint – точка появи ворогів.
10. BP_InteractableBase – базовий інтерактивний об’єкт.
11. BP_Chest – скриня з випадковою нагородою.
12. BP_BossSummoner – об’єкт виклику боса.

Окремо були створені інтерфейси, які дозволяють зменшити жорстку залежність між Blueprint-класами. Для цього використано BPI_Interactable, BPI_StatProvider та BPI_Deathable. Наприклад, персонаж гравця може взаємодіяти не з конкретним класом скрині, а з будь-яким об’єктом, який реалізує інтерфейс взаємодії. Це робить систему більш гнучкою, бо в майбутньому можна додати нові інтерактивні об’єкти без повної переробки логіки персонажа [6].

3.2 Реалізація персонажа гравця та системи керування

Персонаж гравця є центральним об’єктом гри, оскільки саме через нього користувач взаємодіє з ігровим світом. У проєкті для цього використовується Blueprint-клас BP_CharacterBase_Player, створений на основі стандартного класу Character. Це рішення є зручним, оскільки клас Character вже має готову підтримку пересування, колізії, капсули зіткнення та Character Movement Component.

BP_CharacterBase_Player відповідає за такі функції:

- переміщення персонажа по арені;
- обробку введення гравця;
- поворот персонажа у напрямку атаки або руху;
- виконання базової атаки;
- взаємодію з інтерактивними об’єктами;
- отримання доступу до PlayerState та компонента характеристик;
- передачу подій у системи забігу.

Для керування використовується Enhanced Input. Було створено Input Mapping Context та окремі Input Action для основних дій гравця. Такий спосіб зручніший за стару систему введення, оскільки дозволяє окремо описувати дії гравця та прив’язувати їх до клавіш або кнопок [4].

Основні дії введення:

1. IA_Move – переміщення персонажа.
2. IA_Attack – базова атака.
3. IA_Interact – взаємодія з об’єктами.

4. IA_ToggleStatsPanel – відкриття або закриття панелі характеристик.

Під час запуску гри Player Controller додає Mapping Context для локального гравця. Після цього персонаж може отримувати введення і викликати відповідні події. Переміщення реалізовано через стандартні вузли руху Character, а напрямок руху залежить від налаштувань камери та осей введення.

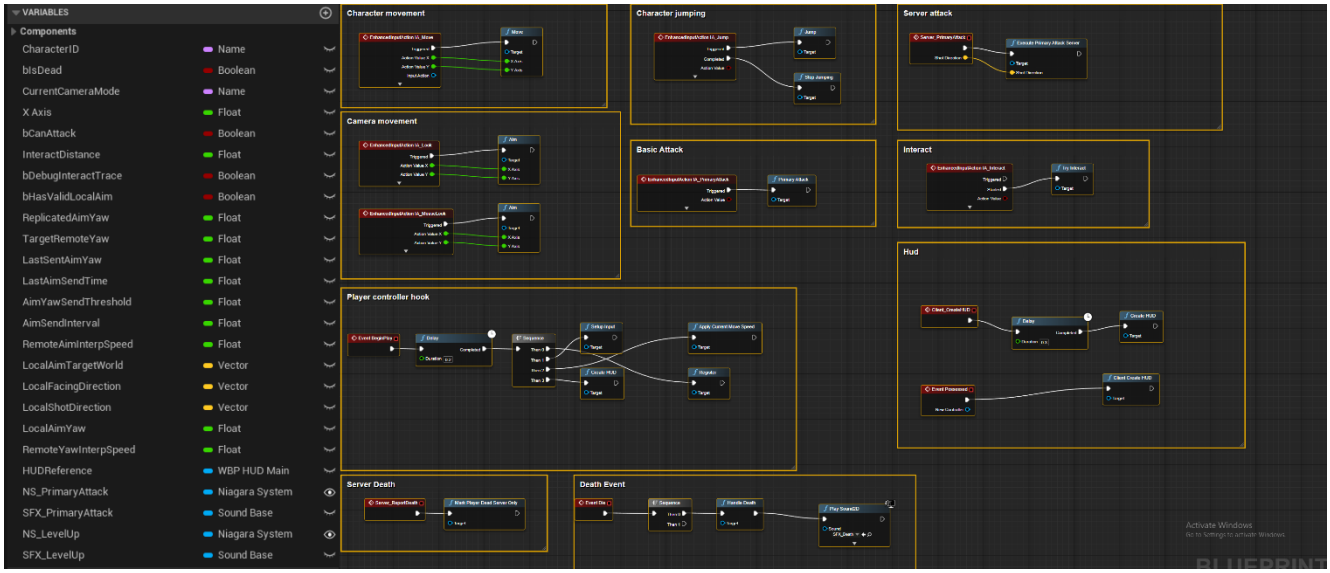


Рис. 3.2 Реалізація персонажа гравця в Blueprint

Окрему роль виконує BP_PC_Player. Player Controller використовується як проміжний рівень між гравцем, персонажем та інтерфейсом. Це потрібно для того, щоб персонаж не відповідав напряму за всю логіку UI. У подальшому це також зручніше для кооперативного режиму, оскільки кожен гравець матиме власний Player Controller і власний локальний інтерфейс.

У межах MVP система керування зроблена достатньо простою. Основна увага була приділена тому, щоб персонаж міг стабільно рухатися, атакувати, взаємодіяти з об’єктами і передавати інформацію іншим системам. Більш складні здібності, альтернативні атаки або різні персонажі можуть бути додані пізніше на основі цієї ж структури.

3.3 Реалізація системи характеристик, шкоди та смерті

Для зберігання і зміни характеристик було створено Actor Component BP_AC_Stats. Його задача полягає в тому, щоб відокремити числові параметри персонажа або ворога від основного Blueprint-класу. Завдяки цьому одна й та сама система характеристик може використовуватися як для гравця, так і для ворогів.

У компоненті BP_AC_Stats були реалізовані такі основні змінні:

1. CurrentHealth – поточне здоров’я.
2. MaxHealth – максимальне здоров’я.
3. Damage – базова шкода.
4. Armor – броня.
5. MoveSpeed – швидкість руху.
6. AttackSpeed – швидкість атаки.
7. HealthRegen – регенерація здоров’я.

Такий набір характеристик є достатнім для MVP. Він дозволяє реалізувати базовий бойовий цикл, предмети, які змінюють параметри, і подальше балансування персонажа. У майбутньому систему можна розширити додатковими параметрами, наприклад шансом критичного удару, силою критичного удару, опором до ефектів або тривалістю бафів.

Основна логіка отримання шкоди також розміщена в BP_AC_Stats. Це зроблено для того, щоб не дублювати однакову обробку шкоди в гравці, ворогах і босі. Коли об’єкт отримує шкоду, компонент зменшує CurrentHealth, враховує потрібні модифікатори та перевіряє, чи не досягло здоров’я нуля.

Загальний процес отримання шкоди має такий вигляд:

- об’єкт отримує запит на нанесення шкоди;
- значення шкоди передається в BP_AC_Stats;
- за потреби враховується броня;
- CurrentHealth зменшується;
- викликається оновлення HUD або інших систем;

– якщо CurrentHealth менше або дорівнює нулю, запускається логіка смерті.

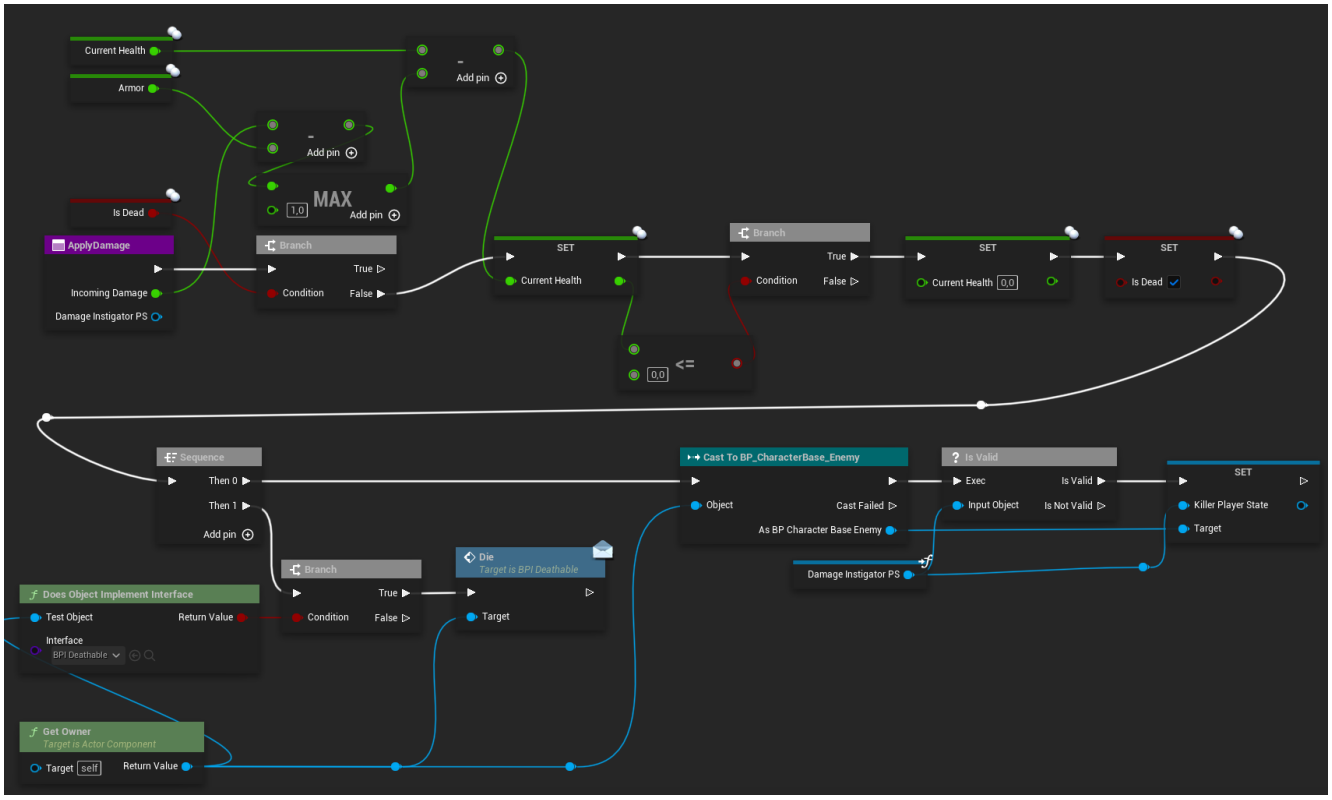


Рис. 3.3 Обробка отримання шкоди в компоненті характеристик

Для атаки використовується параметр AttackSpeed. Він впливає на інтервал між атаками. У спрощеному вигляді інтервал атаки розраховується як обернене значення до швидкості атаки. При цьому використовується мінімальне обмеження інтервалу, щоб уникнути ситуації, коли дуже велике значення AttackSpeed створює технічні проблеми або надмірну кількість викликів атаки.

Приклад логіки атаки:

– гравець натискає кнопку атаки;
– система перевіряє, чи може персонаж атакувати;
– виконується пошук цілі через область або колізію;
– ціль отримує шкоду;
– після атаки запускається затримка відповідно до AttackSpeed;
– після завершення затримки персонаж знову може атакувати.

Смерть об’єкта реалізована через окрему подію. Якщо помирає ворог, після цього може бути видана нагорода, оновлений стан хвилі або викликаний спавн наступних ворогів. Якщо помирає гравець, інформація передається в BP_GM_Run, який перевіряє, чи потрібно завершити забіг.

Для майбутньої кооперативної гри це важливо, тому що смерть одного гравця не повинна автоматично завершувати гру. GameMode повинен перевірити, чи залишилися інші живі гравці. У поточному MVP ця логіка вже враховується на рівні структури, навіть якщо повний кооператив ще не є фінально реалізованим.

3.4 Реалізація ворогів, арени та системи появи противників

Вороги реалізовані на основі Blueprint-класу BP_CharacterBase_Enemy. Він також створений на базі Character, оскільки ворогам потрібні колізія, переміщення та можливість взаємодіяти з іншими об’єктами. До ворога додається компонент BP_AC_Stats, завдяки чому він має здоров’я, шкоду, швидкість руху та інші параметри.

Основні функції базового ворога:

- поява на арені;
- пошук гравця або цілі;
- рух у напрямку гравця;
- нанесення шкоди при наближенні або контакті;
- отримання шкоди від гравця;
- смерть після втрати здоров’я;
- передача події смерті в інші системи.

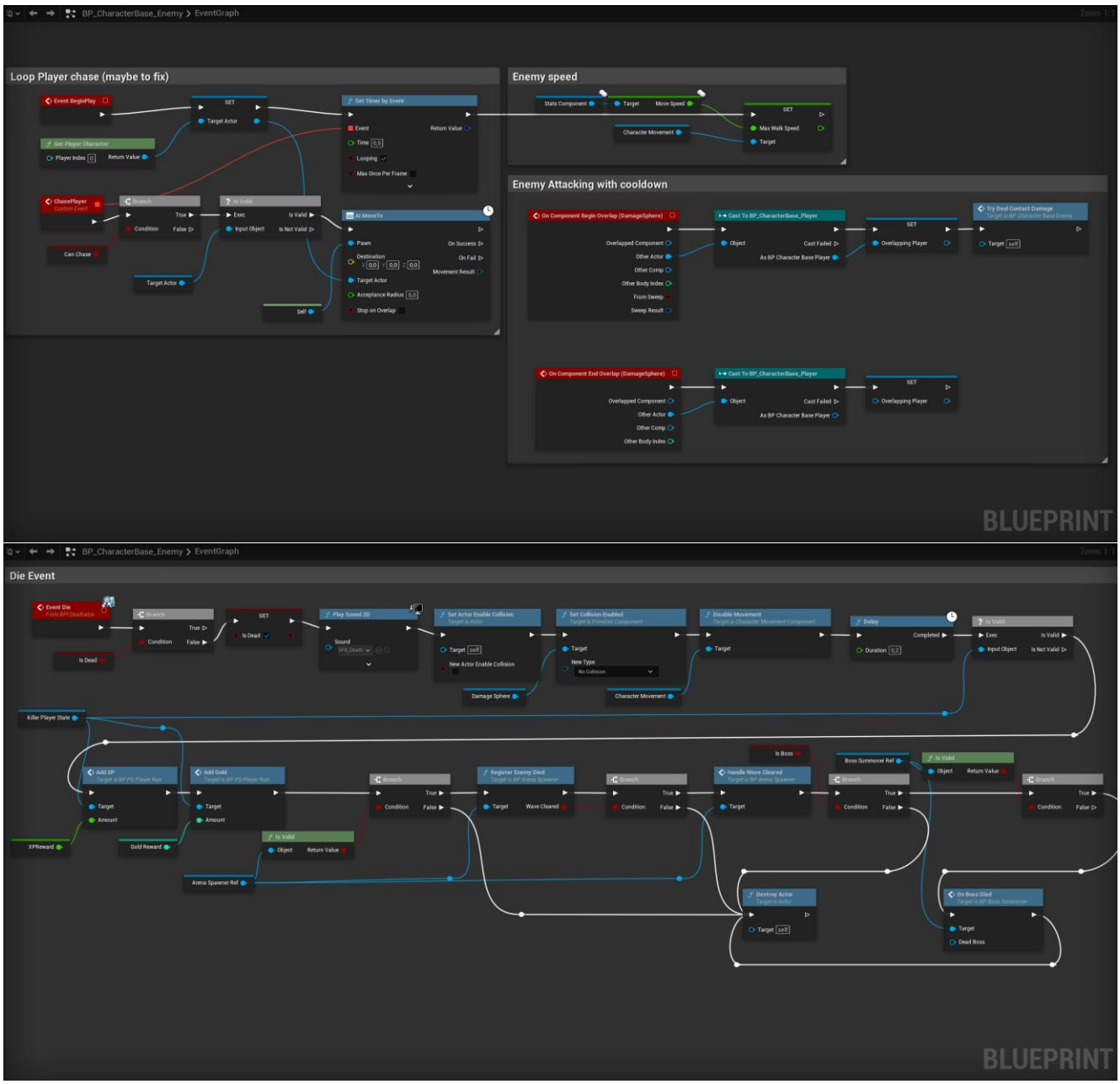


Рис. 3.4 Реалізація базового ворога

Рис. 3.5 Базові вороги

Для появи ворогів використовується BP_ArenaSpawner. Цей Blueprint відповідає за керування хвилями або групами противників. Він не створює ворогів у випадковому місці без контролю, а використовує BP_EnemySpawnPoint. Це дозволяє задавати конкретні точки на арені, де можуть з’являтися противники.

Система появи ворогів складається з таких елементів:

1. BP_ArenaSpawner – керує процесом спавну;
2. BP_EnemySpawnPoint – визначає місце появи ворога;
3. Таблиця або структура записів ворогів;
4. Параметри хвилі;
5. Бюджет або кількість ворогів;
6. Логіка завершення хвилі.

У межах MVP можна використовувати просту систему хвиль. Наприклад, спавнер створює певну кількість ворогів, після чого чекає, поки вони будуть знищені. Коли всі вороги хвилі мертві, можна перейти до наступної хвилі або відкрити можливість виклику боса.

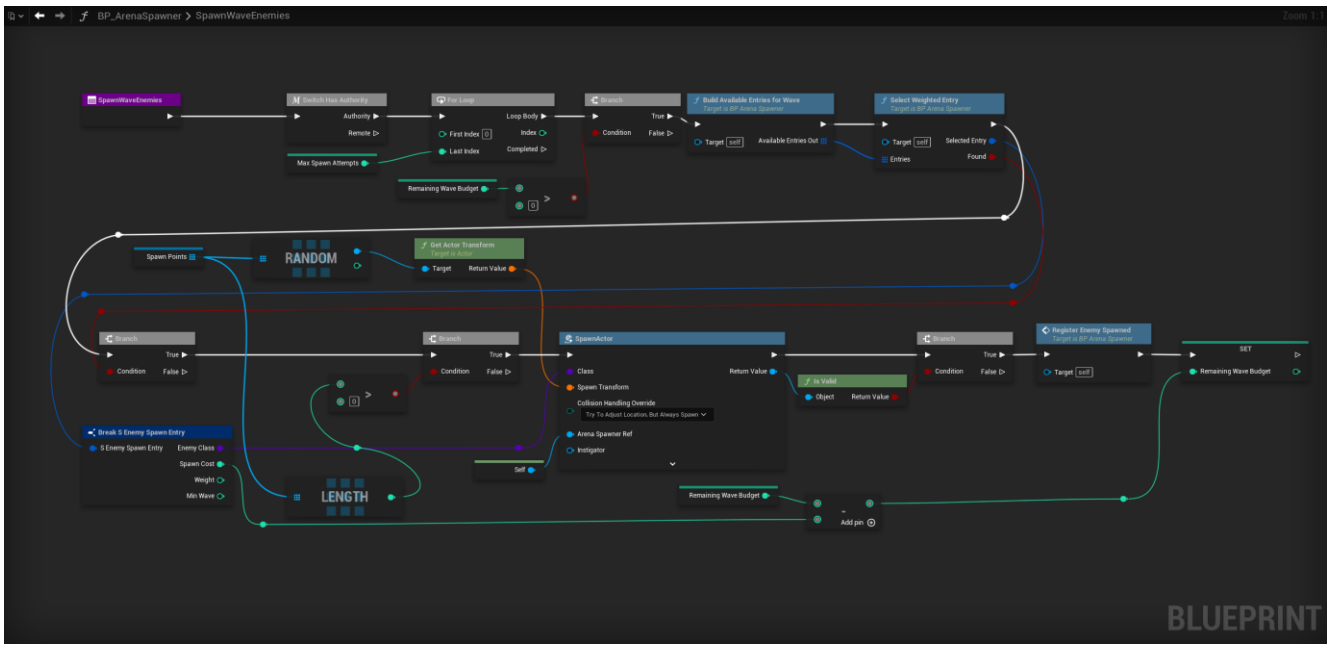


Рис. 3.6 Система появи ворогів на арені

Загальна логіка хвилі:

- запускається StartNextWave;
- визначається кількість або бюджет ворогів;
- вибираються доступні точки появи;
- створюються вороги;
- вороги додаються до списку активних;
- після смерті ворога він видаляється зі списку;
- якщо активних ворогів не залишилось, хвиля вважається завершеною.

Окремо було реалізовано тестового боса BP_Boss_Test. Його основна задача – перевірити фінальну подію забігу. Бос може бути сильнішою версією ворога або окремим Blueprint-класом, який використовує ті самі базові принципи: компонент характеристик, отримання шкоди, смерть і повідомлення GameMode.

Для виклику боса використовується інтерактивний об’єкт BP_BossSummoner. Це дозволяє не запускати фінальну подію автоматично, а дати гравцю можливість активувати її вручну. Такий підхід підходить для загальної бази гравців, оскільки він простий у розумінні та дає гравцю змогу мати можливість віддалено керувати темпом гри.

3.5 Реалізація предметів, скринь і прогресії гравця

Однією з головних частин roguelite-гри є система предметів. У реалізованому MVP предмети використовуються для зміни характеристик гравця під час забігу. Це створює відчуття прогресії, тому що після отримання предметів персонаж стає сильнішим або змінює свій стиль гри.

Для опису предметів використовується структура S_ItemDefinition. Вона володіє загальними даними про предмет, які потім використовуються скринею, PlayerState, компонентом характеристик та інтерфейсом.

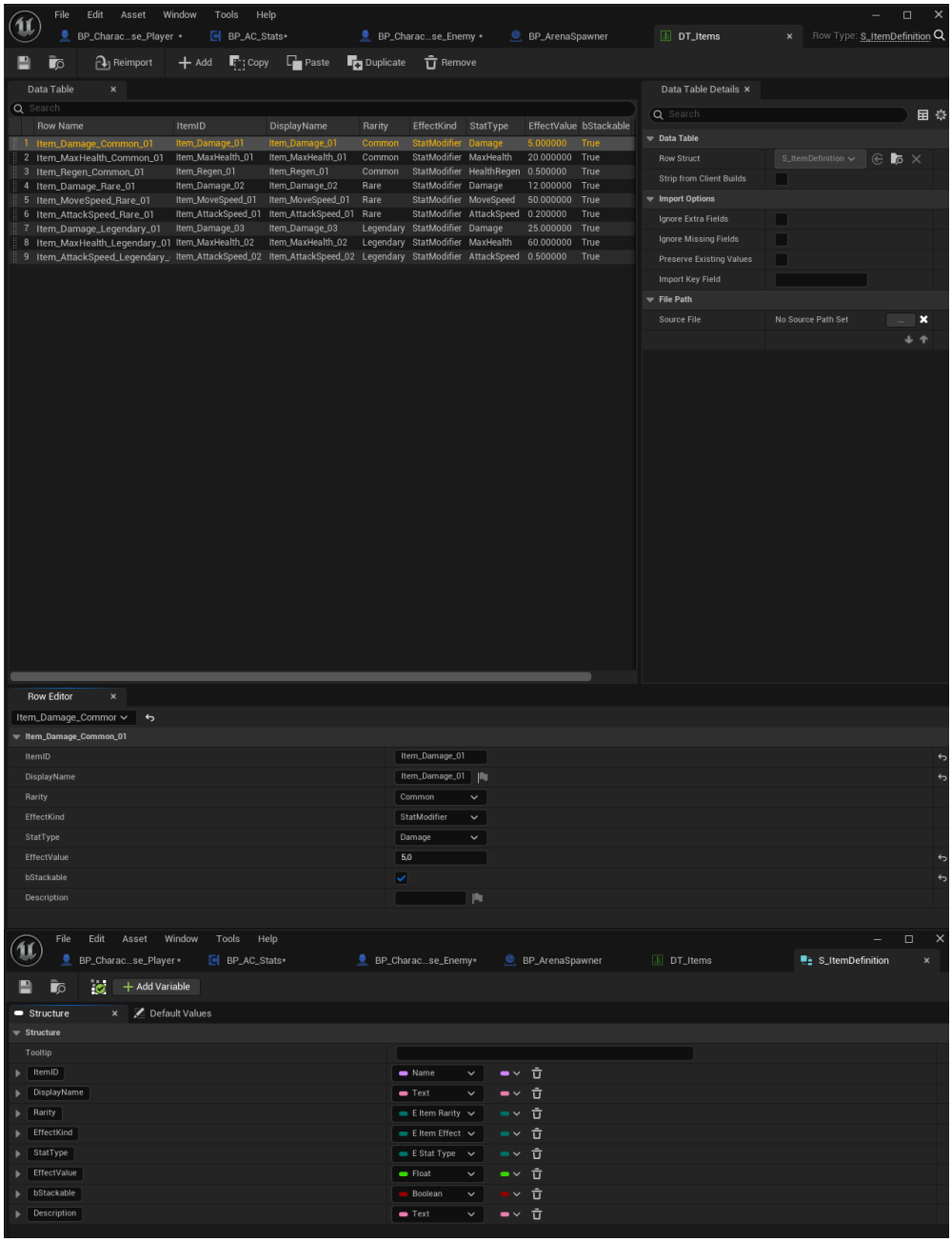


Рис. 3.7 Таблиця даних предметів

Структура предмета включає:

- назву предмета;
- опис ефекту;
- рідкість;
- тип ефекту;
- тип характеристики;
- числове значення;
- іконку для UI.

Для класифікації предметів використовуються enum-и:

- E_ItemRarity – рідкість предмета;
- E_ItemEffectKind – тип ефекту предмета;
- E_StatType – характеристика, на яку впливає предмет.

Предмети зберігаються в таблиці даних DT_Items. Це дозволяє додавати або змінювати предмети без повної зміни Blueprint-логіки. Наприклад, можна змінити значення бонусу до шкоди або шанс випадіння предмета прямо в таблиці, не переписуючи логіку скрині.

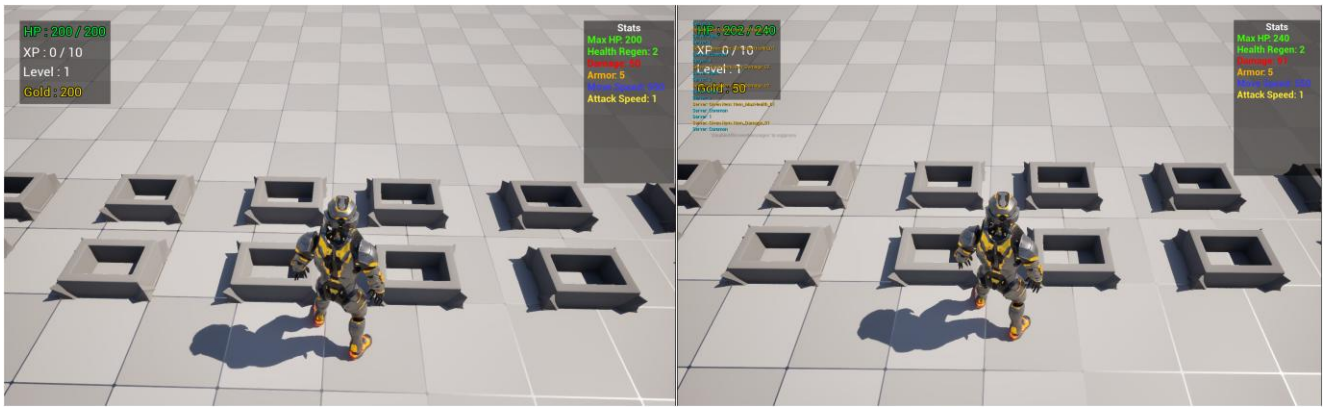


Рис. 3.8 Отримання предмета зі скрині

Для отримання предметів реалізовано BP_Chest. Це інтерактивний об’єкт, з яким гравець може взаємодіяти через систему BPI_Interactable. Скриня перевіряє, чи може гравець її відкрити, списує золото, обирає випадковий предмет і передає його в систему прогресії.

Загальна логіка відкриття скрині:

- гравець підходить до скрині;
- скриня визначається як доступний інтерактивний об’єкт;
- гравець натискає кнопку взаємодії;
- викликається логіка BP_Chest;
- перевіряється кількість золота в BP_PS_PlayerRun;
- якщо золота достатньо, воно списується;
- система обирає випадковий предмет з DT_Items;
- предмет додається до списку отриманих;
- ефект предмета застосовується до BP_AC_Stats;
- HUD оновлює значення характеристик.

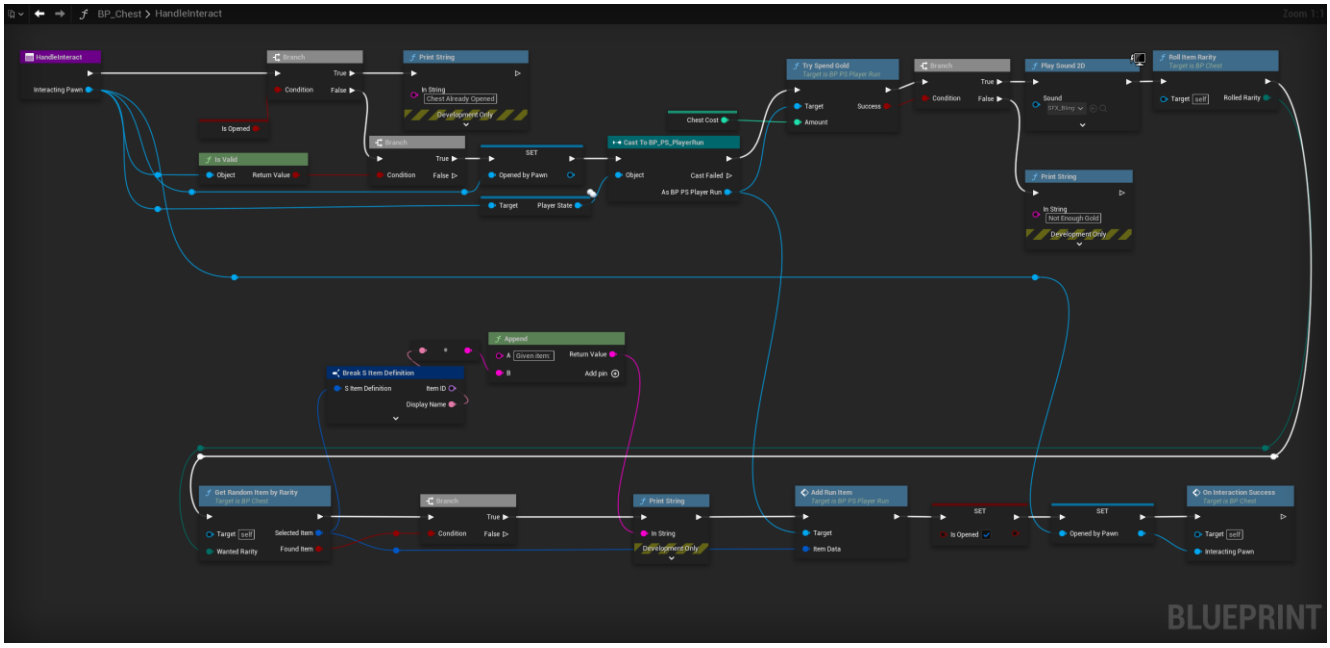


Рис. 3.9 Реалізація відкриття скрині та видачі предмета

Прогресія гравця в межах забігу зберігається в BP_PS_PlayerRun. Там розміщуються значення золота, досвіду, рівня і список отриманих предметів. Це краще, ніж зберігати такі дані лише в персонажі або інтерфейсі. PlayerState логічно описує стан конкретного гравця, тому підходить для таких параметрів, особливо з урахуванням можливого кооперативного режиму.

Основні дані BP_PS_PlayerRun:

1. Gold – кількість золота.
2. Experience – поточний досвід.
3. Level – рівень гравця.
4. InventoryItems – список отриманих предметів.
5. RunStats – статистика поточного забігу.

Після смерті ворога гравець може отримати золото або досвід. Якщо досвіду достатньо для нового рівня, виконується підвищення рівня. У межах MVP рівень може просто збільшувати базові характеристики. У майбутньому цю систему можна розширити вибором покращень, новими здібностями або метапрогресією.

3.6 Реалізація базового інтерфейсу гравця

Базовий інтерфейс гравця реалізовано за допомогою UMG. Основна задача UI полягає в тому, щоб показувати гравцю актуальний стан персонажа і забігу. При цьому інтерфейс не зберігає основну gameplay-логіку, а лише отримує дані з PlayerState, компонента характеристик або GameMode.

У проєкті використовуються такі основні віджети:

1. WBP_HUD_Main – основний HUD гравця.
2. WBP_StatsPanel – панель характеристик.
3. WBP_GameOver – екран поразки.
4. WBP_EndRun – екран завершення забігу.

WBP_HUD_Main показує найважливішу інформацію під час гри. До неї належать здоров’я, золото, рівень, досвід і, за потреби, список предметів. Ці значення оновлюються після зміни відповідних даних у BP_AC_Stats або BP_PS_PlayerRun.

Основні елементи HUD:

- шкала здоров’я;
- значення поточного і максимального здоров’я;
- кількість золота;
- рівень гравця;

- досвід;
- отримані предмети;
- підказка взаємодії з об’єктом.

Рис. 3.10 Відображення основного HUD під час забігу

Панель WBP_StatsPanel використовується для детальнішого перегляду характеристик. Вона може показувати шкоду, броню, швидкість руху, швидкість атаки, регенерацію та інші параметри. Для відкриття або закриття цієї панелі використовується окрема дія введення IA_ToggleStatsPanel.

Оновлення UI виконується через окремі функції, наприклад RefreshHUD, RefreshVitals або RefreshRunInfo. Такий підхід дозволяє не оновлювати весь інтерфейс постійно через Tick. Значення оновлюються тоді, коли це справді потрібно: після отримання шкоди, зміни золота, отримання досвіду або нового предмета.

Загальна схема оновлення HUD:

- змінюється значення в BP_AC_Stats або BP_PS_PlayerRun;
- викликається функція оновлення;
- HUD отримує актуальні дані;
- текст, шкали або іконки оновлюються на екрані.

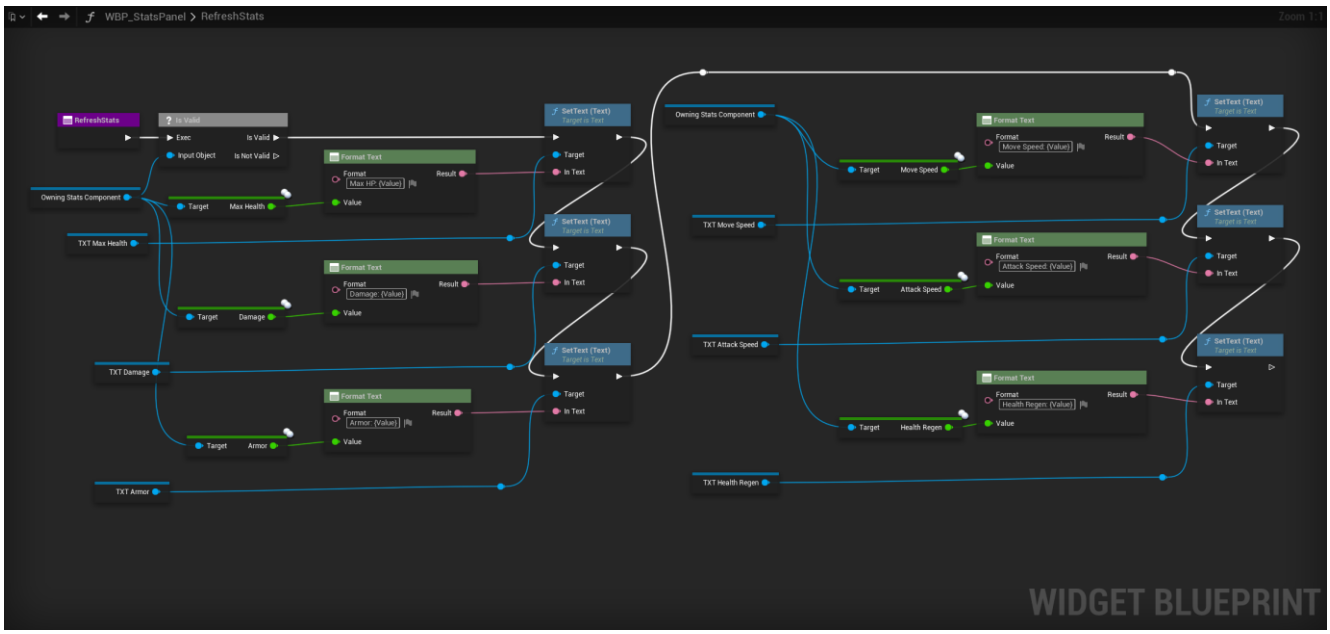


Рис. 3.11 Оновлення інтерфейсу гравця

Екрани завершення гри реалізовані окремо. WBP_GameOver використовується для поразки, а WBP_EndRun може використовуватися для перемоги або загального підсумку забігу. Важливо, що ці віджети не приймають рішення про завершення гри. Вони лише показуються після того, як BP_GM_Run визначив результат забігу.

3.7 Реалізація завершення забігу та підготовка до кооперативного режиму

Завершення забігу реалізується через BP_GM_Run. Цей Blueprint відповідає за основні правила гри: початок сесії, перевірку поразки, перевірку перемоги, показ відповідного екрана та перезапуск забігу. Такий підхід відповідає структурі Gameplay Framework, де GameMode використовується для опису правил гри [3].

У грі передбачено два основні варіанти завершення:

- поразка, якщо гравець або вся команда загинула;
- перемога, якщо був переможений бос або виконана фінальна умова.

Коли гравець помирає, BP_AC_Stats або персонаж повідомляє про це BP_GM_Run. Після цього GameMode перевіряє, чи залишилися живі гравці. Для

одиночного режиму смерть одного г равця означає поразку. Але структура зроблена так, щоб у майбутньому можна було перевіряти стан кількох гравців.

Логіка поразки:

- здоров’я гравця стає меншим або рівним нулю;
- викликається подія смерті;
- BP_GM_Run отримує повідомлення;
- перевіряється кількість живих гравців;
- якщо живих гравців не залишилось, викликається EndRun;
- показується WBP_GameOver;
- гравець може перезапустити забіг.

Перемога реалізована через тестового боса. Гравець активує BP_BossSummoner, після чого з’являється BP_Boss_Test. Якщо бос помирає, GameMode отримує повідомлення про виконання фінальної умови і завершує забіг перемогою.

Логіка перемоги:

- гравець активує об’єкт виклику боса;
- у світі з’являється бос;
- гравець завдає йому шкоди;
- після смерті боса викликається подія перемоги;
- BP_GM_Run запускає EndRun;
- показується екран завершення забігу.

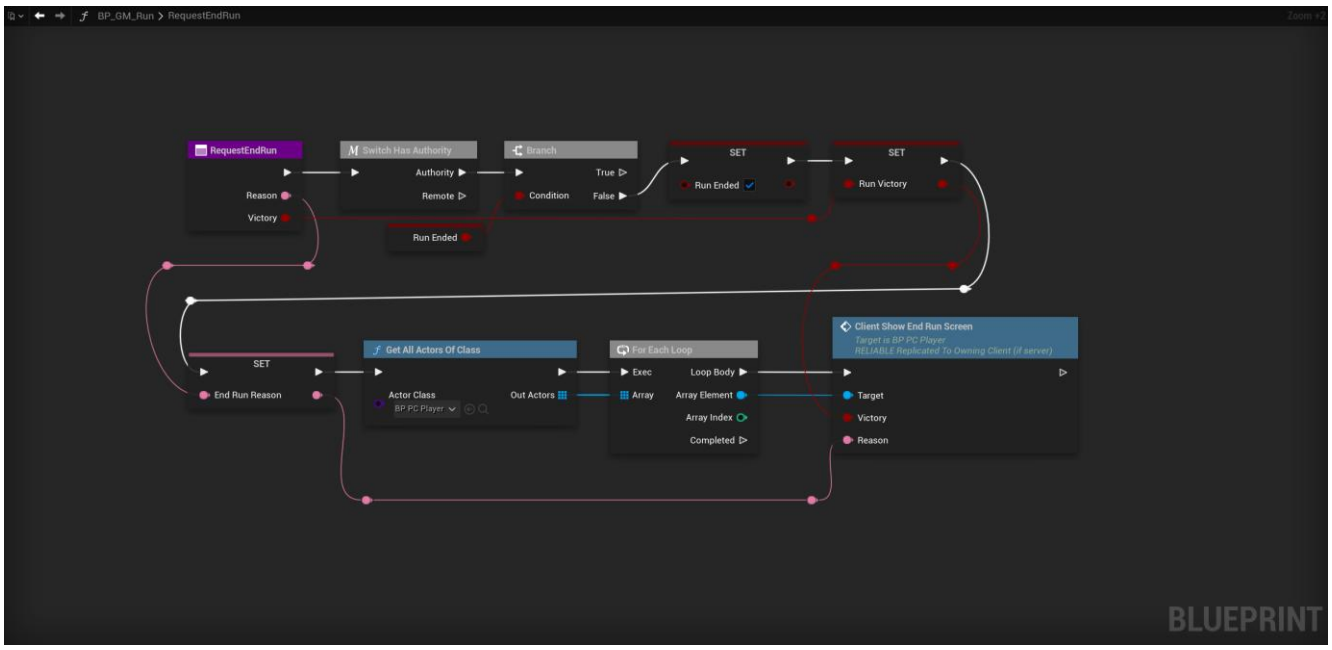


Рис. 3.12 Реалізація завершення забігу

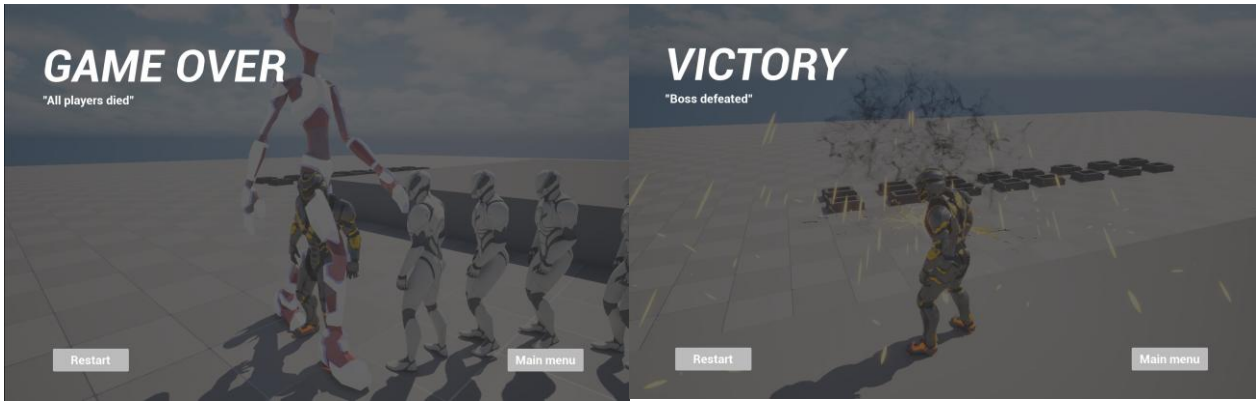


Рис. 3.13 Варіанти екрану завершення забігу

Під час завершення забігу важливо зупинити активну gameplay-логіку. Наприклад, спавнер не повинен продовжувати створювати ворогів після Game Over або перемоги. Також потрібно заблокувати зайві бойові дії, щоб після завершення гри не виникали помилки з активними об’єктами.

Окремо у проєкті враховано можливість майбутнього кооперативного режиму. Повноцінна мережева реалізація потребує додаткового тестування, однак частина архітектури вже побудована так, щоб її було легше розширювати.

Для цього використовуються:

1. PlayerState для індивідуальних даних гравця.

2. PlayerController для локального введення та UI.
3. GameMode для правил забігу.
4. GameState для загального стану сесії.
5. компоненти для повторюваної логіки.
6. інтерфейси для зменшення залежності між класами.

Такий підхід не робить гру автоматично повністю мережевою, але він зменшує кількість логіки, яка прив’язана тільки до одного локального персонажа. Це важливо, тому що при майбутньому додаванні кооперативу основні дані гравців, правила забігу та інтерфейс уже мають більш правильне місце в архітектурі [8].

4 ТЕСТУВАННЯ, АНАЛІЗ ТА ПЕРЕВІРКА РЕЗУЛЬТАТУ

4.1 Організація тестування гри

Після реалізації основних gameplay-систем було проведено їх тестування у самому середовищі гри. Основною метою тестування було перевірити не тільки запуск проєкту, а й працездатність повного ігрового циклу. Для гри такого жанру важливо, щоб гравець міг почати забіг, керувати персонажем, боротися з ворогами, отримувати нагороди, відкривати скрині, посилювати характеристики та завершувати ігрову сесію перемогою або поразкою.

Тестування виконувалося у середовищі Unreal Engine 5 за допомогою режиму Play In Editor. Даний режим є зручним для перевірки Blueprint-логіки, оскільки дозволяє швидко запускати гру після змін, перевіряти окремі механіки та використовувати налагоджувальні повідомлення через Print String. Також Unreal Engine 5 має засоби для локальної перевірки гри у режимі кількох гравців, що є корисним при підготовці проєкту до кооперативної структури [8].

Під час тестування перевірялися такі основні частини проєкту:

- запуск ігрового рівня;
- створення персонажа гравця;
- робота переміщення та камери;
- базова атака гравця;
- отримання шкоди гравцем і ворогами;
- смерть ворогів;
- отримання золота та досвіду;
- підвищення рівня;
- відкриття скрині;
- отримання випадкового предмета;
- зміна характеристик через предмети;
- оновлення користувацького інтерфейсу;
- виклик боса;

- завершення забігу;
- перезапуск гри.

Окремо зверталась увага на те, щоб логіка гри не була повністю прив’язана до одного Blueprint-класу. Наприклад, характеристики зберігаються в окремому компоненті, прогресія поточного гравця – у PlayerState, а правила завершення забігу – у GameMode. Така структура дозволяє краще перевіряти окремі частини системи і зменшує ризик того, що зміна однієї механіки зламає інші.

Для тестування використовувалися такі методи:

- ручне проходження ігрової сесії;
- перевірка окремих Blueprint-графів;
- виведення проміжних значень через Print String;
- перевірка зміни характеристик після отримання предметів;
- перевірка оновлення HUD після зміни даних;
- запуск гри в одиночному режимі;
- локальна перевірка listen server з додатковим клієнтом.

Такий підхід дозволив перевірити не тільки видиму частину гри, але й внутрішню логіку. Наприклад, при відкритті скрині потрібно було переконатися, що гравець має достатньо золота, золото списується, предмет вибирається з таблиці, ефект застосовується до характеристик, а інтерфейс показує оновлені значення.

4.2 Перевірка основного ігрового циклу

Першим етапом була перевірка основного ігрового циклу. У межах MVP важливо було отримати не повністю завершену комерційну гру, а працездатний прототип, у якому основні системи вже взаємодіють між собою. Саме тому тестування починалося з перевірки повного проходження забігу.

Основний цикл гри складається з таких етапів:

- запуск рівня арени;
- поява персонажа гравця;

- поява ворогів;
- бойова взаємодія;
- отримання нагород;
- відкриття скринь;
- посилення характеристик;
- виклик боса;
- завершення забігу.

Під час перевірки було встановлено, що персонаж гравця коректно створюється на рівні, отримує керування від PlayerController і реагує на введення. Гравець може переміщуватися по арені, атакувати ворогів і взаємодіяти з об’єктами. Камера дозволяє контролювати ситуацію під час бою, хоча її фінальне налаштування ще може бути покращене.

Система появи ворогів також була перевірена. Вороги створюються через спавнер на визначених точках рівня. Кожен ворог має власні характеристики, може отримувати шкоду та помирає після зменшення здоров’я до нуля. Після смерті ворога гравець отримує нагороду, що підтверджує зв’язок між системою бою, прогресією і PlayerState.

Результати перевірки основного ігрового циклу подано у таблиці 4.1.

Таблиця 4.1

Перевірка основного ігрового циклу

Елемент перевірки	Запланована поведінка системи	Отриманий результат під час тестування
Ініціалізація ігрової сцени	Рівень відкривається без критичних помилок	Виконується
Створення гравця	Персонаж з’являється на арені	Виконується
Керування	Гравець може рухатися та атакувати	Виконується
Поява ворогів	Вороги створюються через систему спавну	Виконується
Смерть ворогів	Ворог зникає або переходить у стан смерті	Виконується
Отримання нагороди	Гравець отримує золото або досвід	Виконується
Завершення забігу	Показується відповідний результат	Виконується після перевірки умов

У процесі перевірки основного циклу було важливо простежити, щоб події відбувалися у правильній послідовності. Наприклад, ворог не повинен видавати нагороду кілька разів після смерті, а гравець не повинен отримувати предмет зі скрині без достатньої кількості золота. Такі помилки можуть бути не одразу помітні візуально, але вони впливають на баланс і логіку гри.

За результатами цього етапу можна зробити висновок, що базовий gameplay-цикл працює. Гравець може пройти основну послідовність дій, яка є необхідною для MVP rogue-like/roguelite гри.

4.3 Перевірка системи характеристик і бойової взаємодії

Система характеристик є однією з головних частин проєкту, оскільки вона використовується і гравцем, і ворогами. У грі перевірялися такі параметри: поточне здоров’я, максимальне здоров’я, шкода, броня, швидкість руху, швидкість атаки та регенерація. Ці параметри зберігаються не в інтерфейсі, а в окремому Actor Component, що відповідає модульній структурі проєкту.

Під час тестування було перевірено отримання шкоди. Коли ворог або гравець отримує шкоду, значення здоров’я зменшується. Якщо здоров’я стає меншим або рівним нулю, викликається логіка смерті. Для ворогів це означає завершення їхньої участі в бою та видачу нагороди гравцю. Для гравця це може означати завершення забігу, але рішення про це приймає не сам персонаж і не HUD, а логіка GameMode.

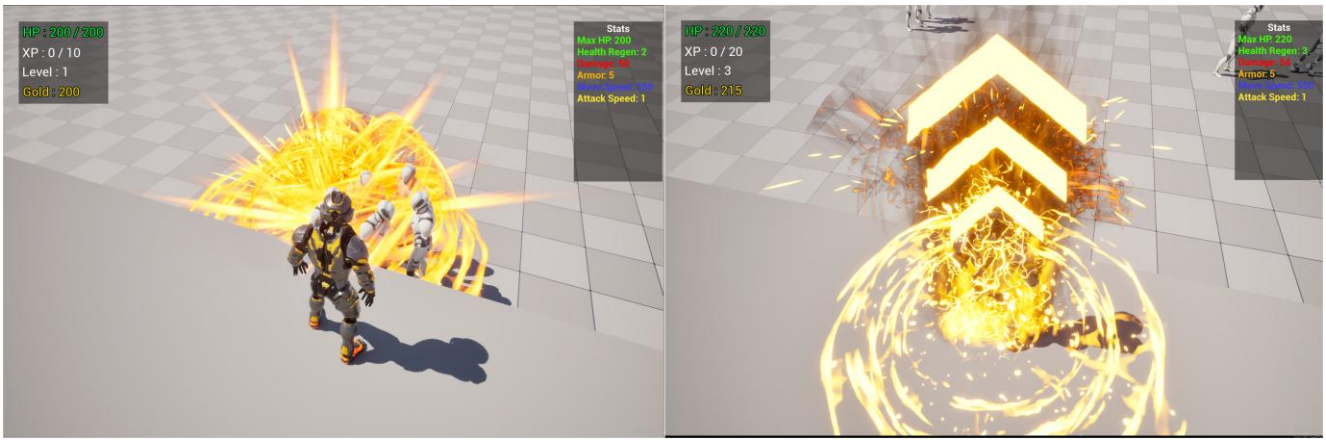


Рис. 4.1 Перевірка бойової взаємодії з ворогами

На рис. 4.1 показано перевірку бойової взаємодії в ігровому середовищі. Ворог переслідує гравця, гравець виконує атаку, після чого значення здоров’я ворога зменшується. Це підтверджує роботу зв’язку між атакою, компонентом характеристик і логікою смерті.

Окремо перевірялась швидкість атаки. Вона використовується для визначення інтервалу між атаками. Якщо швидкість атаки збільшується, персонаж може атакувати частіше. При цьому в системі має бути мінімальне технічне обмеження, щоб дуже великі значення не створювали надмірне навантаження. Це важливо, бо в roguelite-іграх предмети можуть сильно змінювати параметри персонажа.

Також була перевірена броня. Її задача полягає у зменшенні отриманої шкоди. Для MVP не обов’язково робити дуже складну формулу, але сама наявність цього параметра дає можливість у майбутньому створювати предмети, які підвищують виживання персонажа. Те саме стосується регенерації здоров’я, яка може використовуватися як базова характеристика або як ефект окремого предмета.

Перевагою винесення характеристик в окремий компонент є те, що одна й та сама логіка може використовуватись різними сутностями. Гравець, звичайний ворог і бос можуть мати різні значення параметрів, але сама система обробки здоров’я і шкоди залишається спільною. Це зменшує дублювання Blueprint-логіки і робить проєкт зручнішим для подальшого розширення.

4.4 Перевірка системи предметів, нагород і прогресії

Система предметів і нагород перевірялась окремо, оскільки вона є важливою для rogue-like/roguelite жанру. Саме випадкові предмети і поступове посилення персонажа роблять забіг більш варіативним. Якщо предмети не змінюють gameplay, то гра швидко стає одноманітною.

У межах MVP прогресія гравця складається з таких елементів:

- золото;
- досвід;
- рівень;
- предмети;
- зміна характеристик.

Золото використовується як ресурс поточного забігу. Гравець отримує його після знищення ворогів і витрачає на відкриття скринь. Під час тестування було перевірено, що кількість золота змінюється в PlayerState, а не тільки в інтерфейсі. Це важливо, тому що HUD повинен лише показувати дані, але не бути місцем їх основного зберігання.

Досвід і рівень також зберігаються в PlayerState. Після отримання достатньої кількості досвіду рівень гравця підвищується. У межах MVP рівень може впливати на базові характеристики персонажа або використовуватися як основа для подальшого розвитку системи прогресії. Навіть якщо механіка рівнів ще не є фінальною, її наявність дозволяє показати поступове посилення персонажа під час забігу.

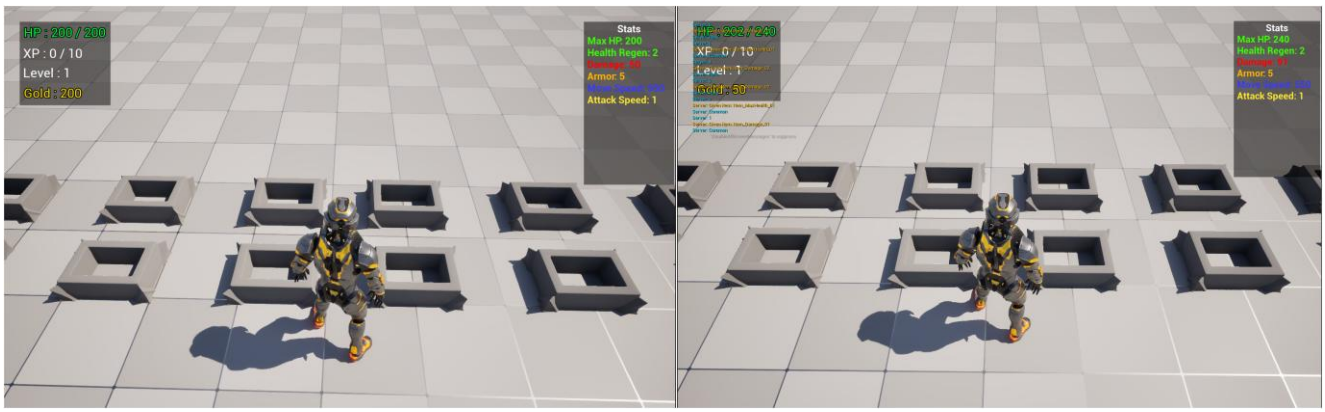


Рис. 4.2 Перевірка отримання предмета зі скрині

Скрині перевірялися через систему взаємодії. Гравець підходить до скрині, натискає кнопку взаємодії, після чого система перевіряє кількість золота. Якщо золота достатньо, воно списується, а гравець отримує випадковий предмет. Якщо золота недостатньо, предмет не видається.

Предмети описуються через data-driven підхід. Для цього використовуються структура предмета, enum-и рідкості, типи ефектів і таблиця даних. Такий спосіб є зручним, тому що нові предмети можна додавати через дані, а не переписувати кожного разу логіку скрині. Data-driven підхід часто використовується в Unreal Engine для зберігання ігрових параметрів окремо від логіки [5].

Під час перевірки предметів було встановлено, що прості ефекти, які змінюють характеристики, працюють коректно. Наприклад, предмет може збільшити максимальне здоров'я, шкоду, броню, швидкість руху або швидкість атаки. Це означає, що система предметів уже впливає на gameplay, а не є тільки декоративним елементом.

Разом з цим система предметів ще має потенціал для розвитку. У майбутньому можна додати предмети з умовними ефектами, активні предмети, синергії, ефекти після вбивства ворога або ефекти, які спрацьовують з певним шансом. Для MVP достатньо реалізувати базову систему, яка доводить, що випадкові нагороди можуть змінювати характеристики персонажа.

4.5 Перевірка користувацького інтерфейсу та завершення забігу

Користувацький інтерфейс перевірявся з точки зору правильного відображення даних. У грі такого типу гравець має швидко бачити основні параметри, оскільки gameplay відбувається в реальному часі. До таких параметрів належать здоров'я, золото, рівень, досвід, характеристики та отримані предмети.

Основний HUD створений за допомогою UMG. Він не зберігає основні gameplay-дані, а тільки отримує їх з інших систем. Це є правильним підходом, оскільки інтерфейс не повинен керувати логікою гри. UMG використовується саме для створення візуального представлення інформації, меню та інших елементів інтерфейсу [7].

Під час тестування перевірялися такі елементи UI:

- відображення здоров'я;
- відображення золота;

- відображення досвіду;
- відображення рівня;
- відкриття панелі характеристик;
- оновлення даних після отримання предмета;
- показ екрана завершення гри.

Оновлення HUD перевірялося після різних подій. Після отримання шкоди має змінюватися здоров’я. Після смерті ворога має оновлюватися золото або досвід. Після відкриття скрині мають оновитися золото, список предметів або характеристики. У процесі перевірки було важливо переконатися, що UI показує актуальні дані, а не старі значення.

Завершення забігу перевірялося у двох основних випадках: поразка після смерті гравця та перемога після знищення боса. У першому випадку компонент характеристик визначає, що здоров’я гравця дорівнює нулю, після чого GameMode перевіряє умови поразки. У другому випадку смерть боса передає сигнал про виконання умови перемоги.

Важливо, що GameOver, Victory і EndRun не повинні бути однією й тією самою дією. EndRun є загальним завершенням забігу, а GameOver або Victory тільки описують причину завершення. Такий поділ робить систему зрозумілішою і дозволяє у майбутньому додати інші варіанти завершення, наприклад вихід через портал або завершення після виконання певної задачі.

Окремо перевірявся перезапуск забігу. Після завершення гри гравець повинен мати можливість почати нову сесію. При цьому старі вороги, таймери або події не повинні продовжувати працювати. Для цього логіка завершення має блокувати активні gameplay-процеси, які вже не потрібні після кінця забігу.

4.6 Виявлені недоліки та їх усунення

Під час тестування було виявлено декілька проблем, які є типовими для MVP-проєкту на Blueprints. Частина з них була пов’язана з неправильним порядком

виклику подій, частина – з оновленням інтерфейсу, а частина – з підготовкою логіки до кооперативного режиму.

Основні виявлені недоліки:

- не всі події взаємодії були правильно підключені;
- HUD не завжди оновлювався після зміни даних;
- деякі значення потрібно було переносити з персонажа у PlayerState;
- логіка завершення забігу спочатку була занадто прив’язана до окремих об’єктів;
- після смерті одного гравця у тестовій кооперативній перевірці гра могла завершуватися для всіх;
- спавнер міг продовжувати роботу після завершення забігу;
- частина Blueprint-графів потребувала впорядкування.

Таблиця 4.2

Основні недоліки та способи їх усунення

Проблема	Спосіб усунення
Взаємодія зі скринею не запускала видачу предмета	Підключено правильний виклик обробки взаємодії
HUD не завжди оновлювався	Додано окремі функції оновлення
GameOver викликався не в тому місці	Логіку перенесено в GameMode
Смерть одного гравця завершувала сесію	Додано перевірку стану активних гравців
Спавнер працював після завершення забігу	Додано перевірку стану забігу
Частина логіки була жорстко прив’язана до класів	Використано компоненти та інтерфейси

Однією з проблем була взаємодія зі скринею. Запит на взаємодію проходив, але сама логіка видачі предмета не запускалась через те, що потрібна функція не була підключена до основної події інтеракції. Після перевірки через Print String було встановлено, на якому етапі ланцюг викликів зупиняється. Після правильного підключення події скриня почала працювати коректно.

Також виникали проблеми з оновленням HUD. У деяких випадках значення змінювалися в логіці гри, але не відображались на екрані. Для виправлення були додані окремі функції оновлення інтерфейсу після зміни здоров’я, золота, досвіду

або предметів. Це краще, ніж постійно оновлювати всі значення через Tick, оскільки такий спосіб є простішим для контролю.

Ще одним важливим виправленням було уточнення логіки завершення забігу. Було розділено поняття GameOver, Victory, EndRun і RestartRun. Це дозволило зробити систему більш зрозумілою. Тепер GameMode відповідає за рішення про завершення гри, а віджети тільки показують відповідний результат.

Після виправлень проєкт став стабільнішим та більш організованим для подальшої роботи. Основний цикл працює, а головні системи гри взаємодіють між собою менш навантажено. Проте цей проєкт все ще потребує покращення в майбутньому. Насамперед це стосується повної мережевої синхронізації, фінального оформлення користувацького інтерфейсу, балансу предметів і складнішої поведінки ворогів.

4.7 Оцінка результатів розробки

У результаті розробки було створено MVP rogue-like/roguelite гри на основі Unreal Engine 5. Проєкт не є фінальною версією гри, але він демонструє основний ігровий цикл і модульну архітектуру gameplay-систем. Гравець може почати забіг, переміщуватися ареною, атакувати ворогів, отримувати шкоду, збирати нагороди, відкривати скрині, отримувати предмети, посилювати персонажа, викликати боса і завершувати ігрову сесію.

До реалізованих результатів можна віднести:

- створення базової структури проєкту Unreal Engine 5;
- реалізацію персонажа гравця;
- реалізацію базових ворогів;
- створення системи характеристик;
- реалізацію бойової взаємодії;
- створення системи золота, досвіду і рівнів;
- реалізацію предметів через дані;
- створення інтерактивної скрині;

- реалізацію виклику боса;
- створення HUD;
- реалізацію завершення забігу.

Головною перевагою реалізованого проєкту є модульність. Основна логіка не знаходиться в одному Blueprint-класі. Характеристики винесені в Actor Component, прогрес гравця зберігається в PlayerState, загальні правила гри знаходяться в GameMode, а предмети описуються через структури та таблиці даних. Це робить проєкт зручнішим для подальшої підтримки.

Також важливо, що структура частково підготовлена до кооперативного режиму. Повноцінний кооператив ще потребує додаткової роботи, особливо в частині реплікації, авторитету сервера і локального UI. Проте використання PlayerController, PlayerState, GameMode і GameState створює правильнішу основу, ніж зберігання всіх даних тільки в персонажі або віджеті.

Подальший розвиток проєкту може включати:

- додавання різноманітних варіантів ворогів;
- реалізацію активних і пасивних здібностей;
- розширення системи предметів;
- покращення UI;
- додавання головного меню;
- реалізацію збереження прогресу;
- балансування складності;
- повноцінне тестування кооперативного режиму.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було створено MVP-прототип rogue-like/roguelite-гри на основі Unreal Engine 5. Основною задачею було не зробити повністю завершену гру, а реалізувати базовий ігровий цикл: початок забігу, керування персонажем, бій з ворогами, отримання нагород, відкриття скринь, посилення характеристик і завершення гри перемогою або поразкою.

У процесі роботи було виконано:

1. Розглянуто предметну область rogue-like та roguelite-ігор. Було визначено основні ознаки жанру: повторювані забіги, випадкові нагороди, розвиток персонажа під час сесії та поступове ускладнення гри. Також було проаналізовано Risk of Rain 2, Hades та Soulstone Survivors.
2. Сформовано вимоги до MVP-гри. Основний акцент зроблено на персонажі, керуванні, бойовій системі, ворогах, характеристиках, предметах, скринях, інтерфейсі та завершенні забігу.
3. Спроєктовано структуру гри в Unreal Engine 5. Логіку було розділено між персонажем, компонентом характеристик, системою предметів, логікою забігу, інтерактивними об’єктами та інтерфейсом.
4. Реалізовано основні gameplay-системи: персонажа гравця, ворогів, характеристики, нанесення шкоди, появу ворогів, нагороди, скрині з випадковими предметами, тестового боса, HUD і завершення гри.
5. Проведено тестування прототипу. Було перевірено запуск рівня, керування, атаку, отримання шкоди, смерть ворогів, нагороди, відкриття скринь, предмети, оновлення інтерфейсу та завершення забігу.

Отже, мету роботи було досягнуто. Створений прототип демонструє базову ігрову петлю rogue-like/roguelite-гри та має основу для подальшого розвитку: додавання нових ворогів, предметів, здібностей, покращення балансу, інтерфейсу та кооперативної логіки.

ПЕРЕЛІК ПОСИЛАНЬ

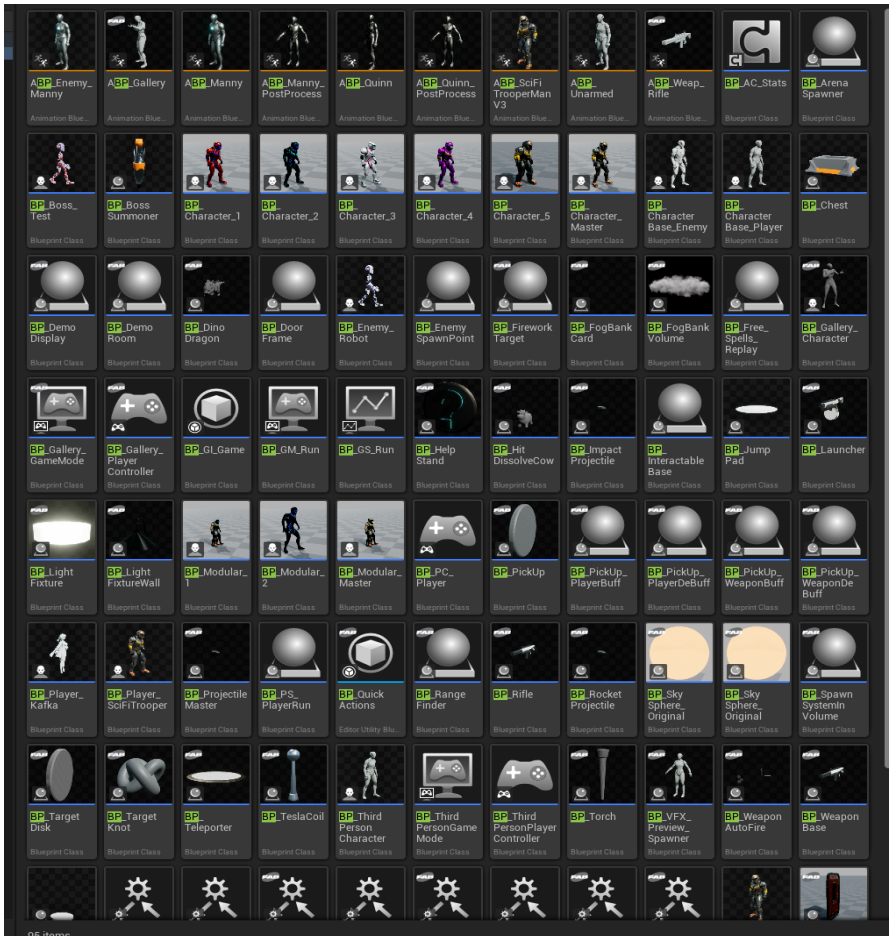
1. Unreal Engine Documentation. Unreal Engine 5 Documentation. URL: https://dev.epicgames.com/documentation/unreal-engine/unreal-engine-5-7-documentation (дата звернення: 28.04.2026)
2. Unreal Engine Documentation. Blueprints Visual Scripting in Unreal Engine. URL: https://dev.epicgames.com/documentation/unreal-engine/blueprints-visual-scripting-in-unreal-engine (дата звернення: 28.04.2026)
3. Unreal Engine Documentation. Gameplay Framework in Unreal Engine. URL: https://dev.epicgames.com/documentation/unreal-engine/gameplay-framework-in-unreal-engine (дата звернення: 28.04.2026)
4. Unreal Engine Documentation. Enhanced Input in Unreal Engine. URL: https://dev.epicgames.com/documentation/unreal-engine/enhanced-input-in-unreal-engine (дата звернення: 28.04.2026)
5. Unreal Engine Documentation. Data Driven Gameplay Elements in Unreal Engine. URL: https://dev.epicgames.com/documentation/unreal-engine/data-driven-gameplay-elements-in-unreal-engine (дата звернення: 28.04.2026)
6. Unreal Engine Documentation. Blueprint Interface in Unreal Engine. URL: https://dev.epicgames.com/documentation/unreal-engine/blueprint-interface-in-unreal-engine (дата звернення: 28.04.2026)
7. Unreal Engine Documentation. Creating User Interfaces With UMG and Slate in Unreal Engine. URL: https://dev.epicgames.com/documentation/unreal-engine/creating-user-interfaces-with-umg-and-slate-in-unreal-engine (дата звернення: 28.04.2026)
8. Unreal Engine Documentation. Networking and Multiplayer in Unreal Engine. URL: https://dev.epicgames.com/documentation/unreal-engine/networking-and-multiplayer-in-unreal-engine (дата звернення: 28.04.2026)
9. Romero M., Sewell B. Blueprints Visual Scripting for Unreal Engine 5. URL: https://www.packtpub.com/en-us/product/blueprints-visual-scripting-for-unreal-engine-5-9781801818698 (дата звернення: 28.04.2026)
10. Nystrom R. Game Programming Patterns. URL: https://gameprogrammingpatterns.com/ (дата звернення: 28.04.2026)
11. Schell J. The Art of Game Design: A Book of Lenses. URL: https://www.routledge.com/The-Art-of-Game-Design-A-Book-of-Lenses-Third-Edition/Schell/p/book/9781138632059 (дата звернення: 28.04.2026)
12. Bycer J. Game Design Deep Dive: Roguelikes. URL: https://www.routledge.com/Game-Design-Deep-Dive-Roguelikes/Bycer/p/book/9780367638191 (дата звернення: 28.04.2026)

13. Short T., Adams T. Procedural Generation in Game Design. URL: https://www.routledge.com/Procedural-Generation-in-Game-Design/Short-Adams/p/book/9781498799195 (дата звернення: 28.04.2026)
14. RogueBasin. Berlin Interpretation. URL: http://www.roguebasin.com/index.php/Berlin_Interpretation (дата звернення: 28.04.2026)
15. Risk of Rain. Official Site. URL: https://riskofrain.2k.com/ (дата звернення: 24.04.2026)
16. Supergiant Games. Hades. URL: https://www.supergiantgames.com/games/hades/ (дата звернення: 28.04.2026)
17. Dead Cells. Official Site. URL: https://dead-cells.com/ (дата звернення: 24.04.2026)
18. Soulstone Survivors. Official Site. URL: https://soulstonesurvivors.com/ (дата звернення: 24.04.2026)
19. Unreal Engine. Clair Obscur: Expedition 33: Autonomy, Creativity, and Community Key to Sandfall Interactive's Success. URL: https://www.unrealengine.com/en-US/developer-interviews/clair-obscur-expedition-33-autonomy-creativity-and-community-key-to-sandfall-interactives-success (дата звернення: 28.04.2026)
20. Epic Games. Unreal Engine 5 Features. URL: https://www.unrealengine.com/en-US/unreal-engine-5 (дата звернення: 28.04.2026)

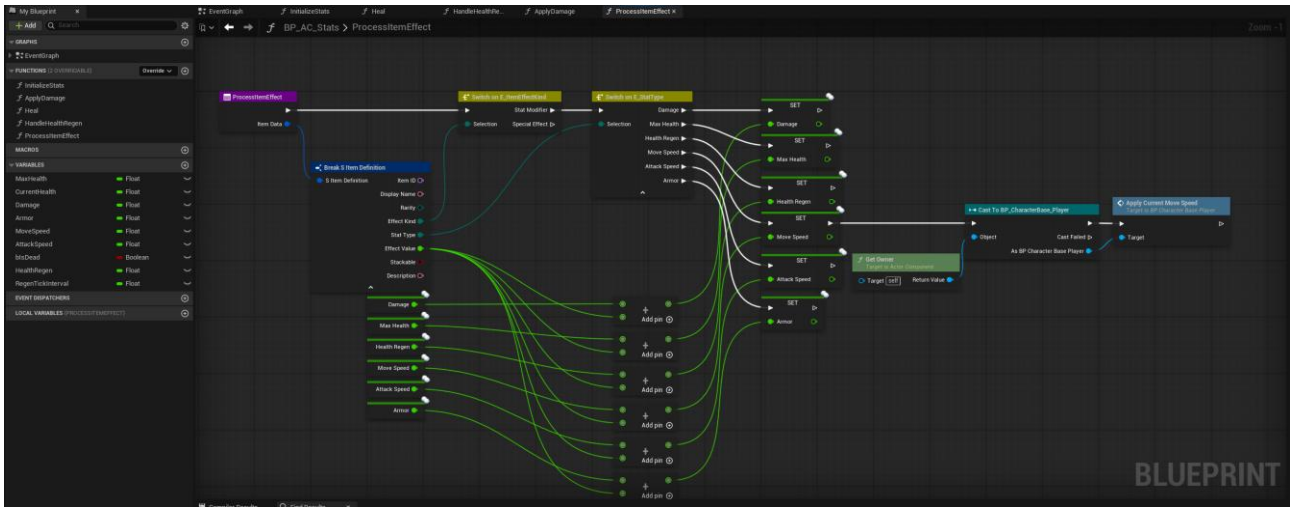
ДОДАТОК А

ТЕСТУВАННЯ КООПЕРАТИВНОГО ЗАПУСКУ ГРИ

ОГЛЯД БІЛЬШОСТІ СТВОРЕНИХ КЛАСІВ



BLUEPRINT-ГРАФ ЗАСТОСУВАННЯ ЕФЕКТУ ПРЕДМЕТА



ДОДАТОК Б. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ.

ДЕРАЖВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-ТЕЛЕКОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

КООПЕРАТИВНА ГРА ЖАНРУ ROGUE-LIKE НА ОСНОВІ UNREAL ENGINE 5

Виконав студент 4 курсу
групи ТЦР-44
Бахмут Михайло Сергійович
Науковий керівник
старший викладач Олейніков Іван Анатолійович

Київ-2026

АКТУАЛЬНІСТЬ ТЕМИ

Високий поріг входу в game development	Потреба у швидкому прототипуванні	Можливості Unreal Engine 5	Blueprints Visual Scripting
Багато ідей комп'ютерних ігор не переходять до реалізації через потребу в навичках програмування, роботі з рушієм та побудові gameplay-систем.	На початковому етапі розробки важливо швидко створити робочий MVP, щоб перевірити ігрову ідею, основний цикл та базову взаємодію гравця з грою.	Unreal Engine 5 надає готові інструменти для створення рівнів, персонажів, анімацій, інтерфейсу, фізики та логіки ігрового процесу.	Blueprints дозволяють реалізувати ігрову логіку засобами візуального скриптування без глибокого написання програмного коду.

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – удосконалення процесу проєктування та реалізації базового ігрового циклу кооперативної гри жанру rogue-like шляхом розробки MVP-прототипу на основі Unreal Engine 5 з використанням Blueprints Visual Scripting, компонентної організації gameplay-систем, системи предметів і прогресії забігу.

Об'єкт дослідження – процес розробки комп'ютерних ігор жанру rogue-like / roguelite у середовищі Unreal Engine 5.

Предмет дослідження – методи та засоби реалізації базових gameplay-систем кооперативної rogue-like гри за допомогою Blueprints Visual Scripting та інструментів Unreal Engine 5.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз особливостей rogue-like та roguelite-ігор.
2. Розглянути існуючі аналоги та визначити корисні gameplay-рішення.
3. Сформувати функціональні та нефункціональні вимоги до MVP гри.
4. Спроєктувати структуру взаємодії основних gameplay-систем.
5. Реалізувати керування персонажем, бойову взаємодію та ворогів.
6. Реалізувати ігровий забіг, нагороди, розвиток персонажа та завершення гри.
7. Створити інтерфейс користувача для відображення стану гравця.
8. Провести тестування основних сценаріїв роботи MVP.

АНАЛІЗ АНАЛОГІВ

Критерії	Risk of Rain 2	Soulstone Survivors	Hades	Розроблена гра
Повторюваний ігровий цикл	Забіг зі зростанням складності	Арена з хвилями ворогів	Серія бойових кімнат	Арена зі зростанням складності
Випадкові нагороди	Предмети	Навички та покращення	Дари	Предмети
Розвиток персонажа під час забігу	Предмети та рівні	Навички й характеристики	Дари	Предмети рівні
Бойова взаємодія	Динамічний бій	Масові бої	Швидкий ближній бій	Динамічний бій
Вороги / Хвилі / Кімнати	Вороги на рівні	Хвилі ворогів	Бойові кімнати	Хвилі ворогів у кімнаті
Реіграбельність	Висока	Висока	Середня-Висока	Середня-Висока

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги

1. Керування персонажем – переміщення, камера, атака та взаємодія з об’єктами.
2. Бойова система – перевірка попадання, нанесення шкоди, отримання шкоди та смерть ігрових сутностей.
3. Система ворогів – створення ворогів, переслідування гравця, атака та обробка смерті.
4. Ігровий забіг – запуск сесії, поява ворогів, розвиток персонажа та завершення гри.
5. Нагороди та предмети – отримання досвіду, золота, випадкових предметів і зміна характеристик.
6. Інтерфейс користувача – відображення HP, XP, рівня, золота, характеристик і результату забігу.

Нефункціональні вимоги

1. Доступність розробки – використання Blueprints Visual Scripting замість складного програмного коду.
2. Стабільність MVP – коректна робота основного ігрового циклу без критичних помилок.
3. Продуктивність – стабільна робота гри під час бою та появи ворогів.
4. Зручність – зрозуміле керування та читабельний інтерфейс.
5. Розширюваність – можливість додавання нових ворогів, предметів, рівнів і босів.
6. Структурованість – розділення логіки між Blueprint-класами, компонентами, UI та даними.

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Використані технології та їх роль у реалізації MVP кооперативної гри жанру rogue-like на основі Unreal Engine 5

Основні засоби розробки

Unreal Engine 5

Ігровий рушій для створення рівнів, акторів, компонентів, фізики та основної логіки гри.

Blueprints Visual Scripting

Візуальне скриптування gameplay-логіки: персонаж, бій, вороги, нагороди та логіка здібгу без глибокого написання коду.

Enhanced Input

Система обробки введення гравця: рух, атака, взаємодія та інші ігрові дії.

UMG

Створення інтерфейсу користувача: HUD, HP, XP, рівень, золото, налаштування та екрани завершення гри.

Animation Blueprint

Керування анімаціями персонажа і ворогів: рух, атаки, отримання шкоди та смерть.

Data Tables / Structs / Enums

Зберігання параметрів предметів, характеристик, типів ефектів та інших ігрових даних.

Niagara VFX

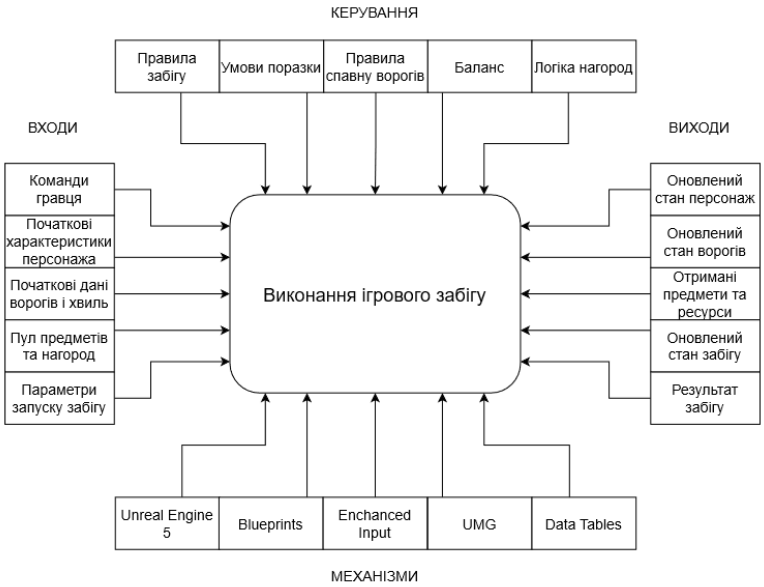
Реалізація візуальних ефектів: атака, попадання, підсилення, рівня та інші події.

Готові assets

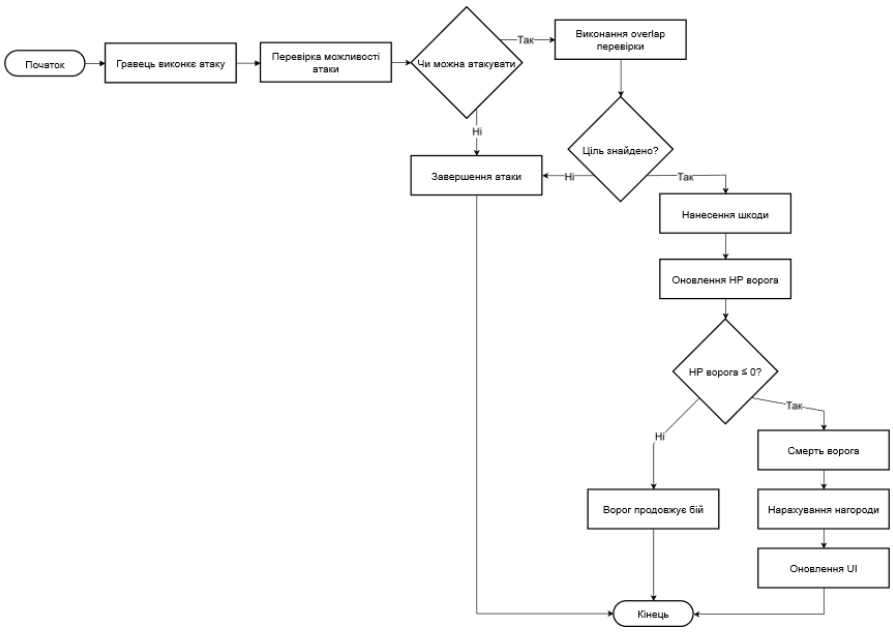
Використання моделей, матеріалів, анімацій і ресурсів для швидкого створення MVP.

Усі основні gameplay-системи реалізовано в Unreal Engine 5 із використанням Blueprint-first підходу.

IDEF0-МОДЕЛЬ ВЗАЄМОДІЇ GAMEPLAY-СИСТЕМ

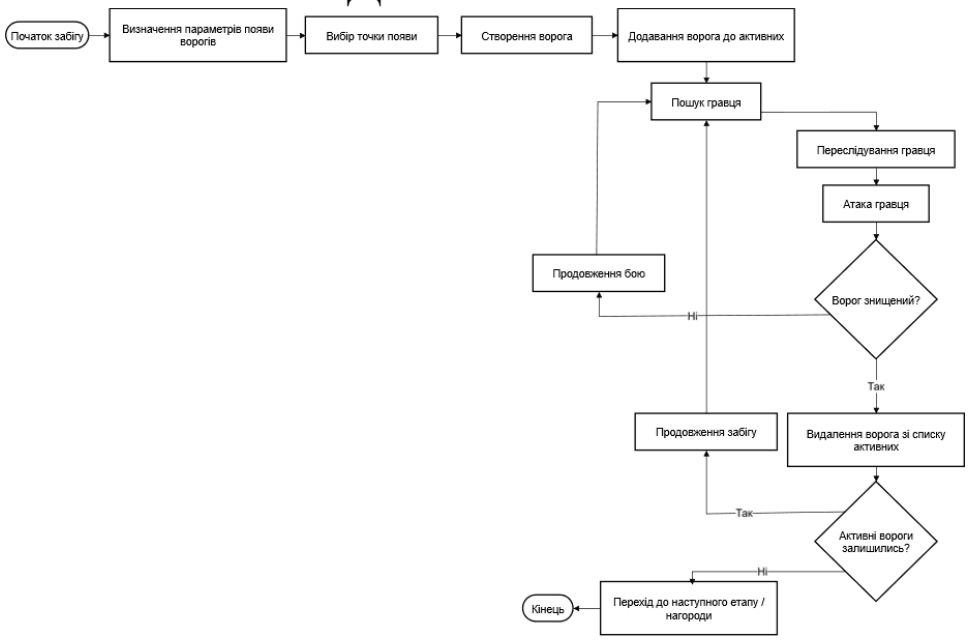


БЛОК-СХЕМА АЛГОРИТМУ БОЙОВОЇ СИСТЕМИ



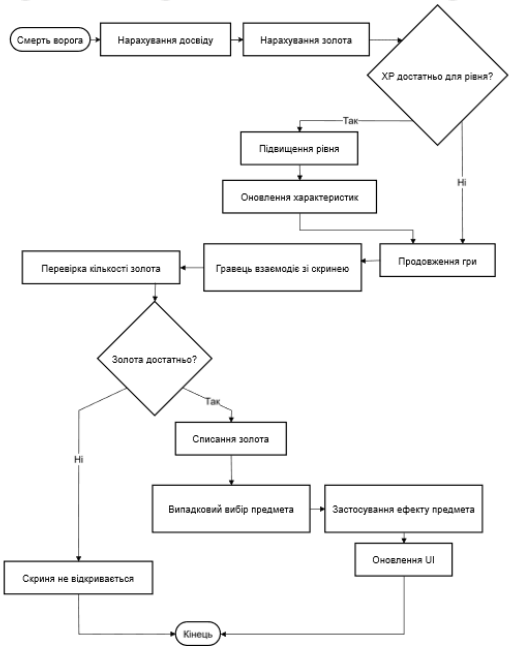
9

БЛОК-СХЕМА АЛГОРИТМУ СПАВНУ ТА ПОВЕДІНКИ ВОРОГІВ



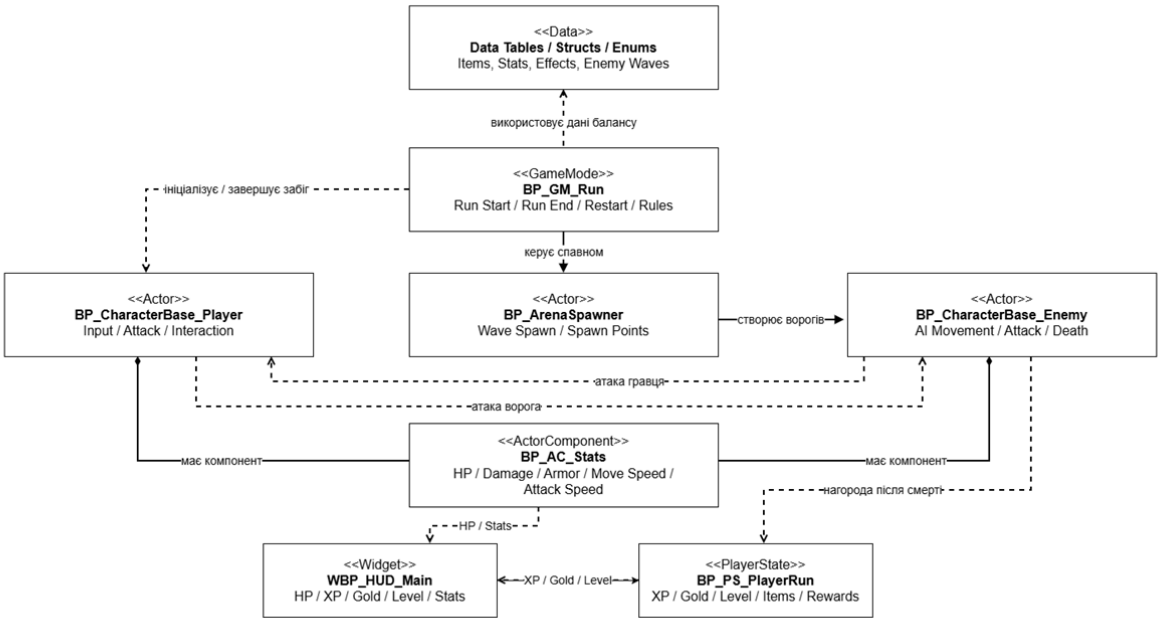
10

БЛОК-СХЕМА АЛГОРИТМУ СИСТЕМИ ПРОГРЕСІЇ ТА НАГОРОД



11

UML-ДІАГРАМА СТРУКТУРИ GAMEPLAY-СИСТЕМ



12

ЕКРАННІ ФОРМИ

Ігровий персонаж

інтерфейс користувача

ЕКРАННІ ФОРМИ

Ворог переслідує гравця

Гравець виконує атаку
та наносить шкоду

Здоров'я ворога зменшується,
смерть ворога після HP <= 0

Бойова взаємодія з ворогами

ЕКРАННІ ФОРМИ

персонаж біля скрині отримання предмета і оновлення характеристик

Нагороди, предмети та розвиток персонажа *15*

ВІДЕО ДЕМОНСТРАЦІЯ

16

АПРОБАЦІЯ

1. Бахмут М.С. Проєктування архітектури гри типу roguelike на основі Unreal Engine 5. Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація». 1-3 травня 2026 року, Київ, ДУІКТ. Збірник тез (подано в друк).
2. Бахмут М.С. Прототип адаптивної системи формування складності та нагород у roguelike-грі на базі Unreal Engine 5. VI Всеукраїнська науково- практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». 15 травня 2026 року, Київ, ДУІКТ. Збірник тез (подано в друк).

ВИСНОВКИ

1. Проведено аналіз особливостей rogue-like / roguelite-ігор та визначено вимоги до створення кооперативного MVP на основі Unreal Engine 5. Основний акцент зроблено на побудові повторюваного ігрового циклу, мережевій взаємодії гравців, розділенні gameplay-логіки та можливості подальшого розширення проєкту.
2. Розроблено MVP кооперативної rogue-like гри з використанням Blueprint-first підходу та елементів Unreal Engine Gameplay Framework. Логіку забігу, стан гравця, персонажів, інтерфейс і дані гри розділено між відповідними Blueprint-класами та компонентами.
3. Підтверджено працездатність MVP тестуванням у середовищі Unreal Engine через запуск додаткових клієнтів. Результат роботи може бути використаний як технічна основа для подальшого розвитку кооперативної rogue-like гри.