БАКАЛАВРСЬКА РОБОТА

БР.ІП-09.00.00.000 ПЗ

Група ІП-22-1

Тріщ Ростислав

2026

Івано-Франківський національний технічний університет нафти і газу
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

Тріщ Ростислав Васильович

(прізвище, ім'я, по батькові)

УДК 004.4

БАКАЛАВРСЬКА РОБОТА

Розробка комп'ютерної гри Etemenamki в жанрі Roguelite, використовуючи серидовище Unity

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня Тріщ Р. В.
(підпис, ініціали та прізвище здобувача)

Науковий керівник Шекета Василь Іванович, д.т.н, професор
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звесорання керівника)

Допущено до захисту

Завідувач кафедри

доц. _____ _____ _____ Бандура В.В.
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ - 2026

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 - Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доцент. В.В. Бандура
“ ____ ” _____2026 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Тріщу Ростиславу Васильовичу
(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) " Розробка комп’ютерної гри Etemenamki в жанрі Roguelite, використовуючи серидовище Uniti "

затверджені наказом вищого навчального закладу від **“** _21_ ” _квітня 2026 р. № 179/7

2. Строк подання студентом проекту (роботи) 10 червня 2026р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

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

1 Проведення та аналіз предметної області та особливостей жанру Roguelite.

2 Виконати вибір технологій та засобів розробки програмного продукту

3 Розробка архітектуру та реалізувати основні ігрові підсистеми гри

4 Проведення тестування програмного продукту та оцінити результати розробки

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1 Папка скриптів проекту (рис. 2.1, ст. 27)

2 Схема роботи дистанційної атаки (рис. 3.1, ст. 35)

3 Машина станів анімації противника (рис 3.4, ст 38)

4 Діаграма росту складності (рис. 3.6 ст. 41)

5 Таблиця результатів тестування функціональності (рис 5.3 ст 55)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

2. Дата видачі завдання **2026 р.**

Керівник Шекета В. І. _____
(підпис)

Завдання прийняв до виконання Тріщ Р. В. _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Визначення та обґрунтування теми роботи	04.02.2026	виконано
2	Аналіз предметної області та існуючих рішень	21.03.2026	виконано
3	Розробка архітектури та реалізація програмного продукту	28.04.2026	виконано
4	Тестування та документування результатів розробки	03.06.2026	виконано
5	Оформлення пояснювальної записки бакалаврської роботи	10.06.2026	виконано

Студент _____ Тріщ Р. В.
(підпис)

Керівник роботи _____ Шекета В. І.
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 72 сторінок, 15 рисунків, 2 таблиці, список використаних джерел із 35 найменувань та додаток.

Метою бакалаврської роботи є розробка комп'ютерної гри «Etemenamki» жанру Roguelite з використанням ігрового рушія Unity та мови програмування C#.

Об'єктом дослідження є процес розробки комп'ютерних ігор на пла Unity.

Предметом дослідження є програмні засоби та алгоритми реалізації ігрових механік жанру Roguelite

У першому розділі проведено аналіз предметної області та особливостей жанру Roguelite, розглянуто існуючі програмні рішення та сформульовано вимоги до програмного продукту.

У другому розділі обґрунтовано вибір ігрового рушія Unity, мови програмування C#, середовища розробки Microsoft Visual Studio та описано архітектуру програмного забезпечення.

У третьому розділі наведено реалізацію основних підсистем гри, зокрема системи керування персонажем, бойової системи, штучного інтелекту ворогів, процедурної генерації рівнів, системи прогресії та масштабування складності.

У четвертому розділі розглянуто архітектуру програмного забезпечення гри та організацію взаємодії ігрових систем

У п'ятому розділі описано користувацький інтерфейс, особливості ігрового процесу та результати тестування програмного продукту.

Висновки роботи містять результати розробки та оцінку працездатності створеного програмного продукту. Проведене тестування підтвердило коректність роботи реалізованих механік та відсутність критичних помилок.

Отримані результати включають розроблену комп'ютерну гру «Etemenamki», реалізовані механіки жанру Roguelite, систему процедурного формування рівнів, прогресії персонажа та масштабування складності

КЛЮЧОВІ СЛОВА: UNITY, C#, ROGUELITE, КОМП'ЮТЕРНА ГРА, ПРОЦЕДУРНА ГЕНЕРАЦІЯ РІВНІВ, ІГРОВІ МЕХАНІКИ, МАСШТАБУВАННЯ СКЛАДНОСТІ, СИСТЕМА ПРОГРЕСІЇ ПЕРСОНАЖА.

ABSTRACT

The bachelor's thesis contains 72 pages, 15 figures, 2 tables, a list of 35 references, and an appendix.

The aim of the bachelor's thesis is to develop the Roguelite computer game "Etemenamki" using the Unity game engine and the C# programming language.

The object of research is the process of game development on the Unity.

The subject of research is software tools and algorithms for implementing game mechanics of the Roguelite genre.

The first chapter presents an analysis of the subject area and the features of the Roguelite genre, reviews existing software solutions, and formulates the requirements for the software product.

The second chapter substantiates the choice of the Unity game engine, the C# programming language, and the Microsoft Visual Studio development environment, and describes the software architecture.

The third chapter presents the implementation of the game's main subsystems, including the character control system, combat system, enemy intelligence, procedural level generation, character progression system, and difficulty scaling mechanism.

The fourth chapter examines the software architecture of the game and the organization of interactions between its gameplay systems.

The fifth chapter describes the user interface, gameplay features, and the results of software product testing.

The conclusions summarize the development results and evaluate the performance of the created software product. The conducted testing confirmed the correctness of the implemented mechanics and the absence of critical errors.

The obtained results include the developed computer game "Etemenamki", implemented Roguelite gameplay mechanics, a procedural level generation system, a character progression system, and a difficulty scaling mechanism.

KEYWORDS: UNITY, C#, ROGUELITE, COMPUTER GAME, PROCEDURAL LEVEL GENERATION, GAME MECHANICS, DIFFICULTY SCALING, CHARACTER PROGRESSION SYSTEM.

ЗМІСТ

ВСТУП **9**

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ **11**

1.1. Постановка задачі 11

1.2. Аналіз жанру Roguelite 13

1.3. Аналіз існуючих рішень 16

1.4. Вимоги до програмного продукту 18

1.5. Висновки по розділу 19

РОЗДІЛ 2. МЕТОДИ ТА ЗАСОБИ РОЗРОБКИ **21**

2.1. Вибір ігрового рушія 21

2.2. Вибір мови програмування 22

2.3. Вибір середовища розробки 24

2.4. Використані пакети Unity 25

2.5. Архітектура програмного забезпечення 28

2.6. Висновки по розділу 32

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ГРИ **34**

3.1. Система керування персонажем 34

3.2. Бойова система 36

3.3. Штучний інтелект ворогів 38

3.4. Процедурна генерація рівнів 41

3.5. Система масштабування складності 42

3.7. Висновки по розділу 45

РОЗДІЛ 4. ПРОЕКТУВАННЯ ТА АРХІТЕКТУРА ГРИ **47**

4.1. Архітектура програмного забезпечення гри 47

4.2. Взаємодія компонентів 49

4.3. Організація даних та ігрових систем 51

4.4. Оптимізація та продуктивність гри 54
4.5. Висновки по розділу 56
РОЗДІЛ 5. ОПИС ТА ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ 57
5.1. Інтерфейс користувача 57
5.2. Ігровий процес 59
5.3. Тестування функціональності 62
5.4. Оцінка результатів 64
5.5. Висновки по розділу 66
ВИСНОВКИ 68
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 70
ДОДАТКИ

ВСТУП

Сучасна індустрія комп’ютерних ігор є однією з найбільш динамічних галузей інформаційних технологій. Розвиток апаратного забезпечення, програмних засобів та ігрових рушіїв сприяв появі великої кількості проєктів різних жанрів і масштабів. Особливе місце серед них займають ігри жанру Roguelite, популярність яких зростає завдяки поєднанню динамічного ігрового процесу, випадкової генерації елементів гри та високої повторної цінності проходження. Такі ігри дозволяють створювати унікальний досвід для користувача під час кожної нової ігрової сесії, що робить їх актуальним напрямком сучасної ігрової розробки.

Актуальність теми зумовлена зростанням популярності комп’ютерних ігор жанру Roguelite та активним розвитком ігрових рушіїв, які надають широкі можливості для створення інтерактивних програмних продуктів.

Метою бакалаврської роботи є розробка комп’ютерної гри «Etemenamki» в жанрі Roguelite на платформі Unity із реалізацією основних ігрових механік, системи розвитку персонажа, системи масштабування складності та формування рівнів на основі шаблонних кімнат.

Завданнями достідження є аналіз існуючих підходів до розробки ігор жанру Roguelite, проєктування архітектури програмного забезпечення, реалізація систем керування персонажем, генерації рівнів та прогресії, а також перевірка працездатності створеного програмного продукту.

Об'єктом дослідження є процес розробки комп’ютерної гри жанру Roguelite на платформі Unity.

Предметом дослідження є програмні засоби, алгоритми та механізми реалізації ігрових систем, що забезпечують функціонування комп’ютерної гри «Etemenamki», зокрема систем керування персонажем, взаємодії з противниками, формування рівнів та масштабування складності.

Методи дослідження включають аналіз і узагальнення науково-технічної літератури з питань розробки комп'ютерних ігор жанру Roguelite, методи об'єктно-орієнтованого проєктування для побудови архітектури програмного забезпечення, а також програмну реалізацію і тестування створених ігрових механік. Для реалізації програмного продукту використано засоби ігрового рушія Unity, мову програмування C# та сучасні підходи до розробки інтерактивних програмних систем.

Наукова новизна одержаних результатів полягає у проєктуванні та реалізації програмного продукту, який поєднує модульну архітектуру ігрових систем, процедурне формування рівнів на основі шаблонних кімнат, систему розвитку персонажа із використанням кристалів та механізм динамічного масштабування складності. Запропонований підхід забезпечує можливість подальшого розширення функціональних можливостей гри та може бути використаний як основа для дослідження методів побудови масштабованих ігрових систем.

Практичне значення отриманих результатів полягає у створенні працездатної комп'ютерної гри, яка може бути використана як основа для подальшого розвитку проєкту, розширення функціональних можливостей та дослідження механік жанру Roguelite. Отримані результати також можуть бути використані під час вивчення технологій розробки комп'ютерних ігор із застосуванням ігрового рушія Unity.

Бакалаврська робота містить 72 сторінок, 15 рисунків, 2 таблиці, список використаних джерел із 35 найменувань та додаток.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Постановка задачі

У межах даної роботи необхідно розробити однокористувацьку комп'ютерну гру «Etemenamki» у жанрі Roguelite на рушії Unity. Об'єктом дослідження є процес створення комп'ютерної гри, а предметом - методи та засоби реалізації ключових ігрових механік, таких як генерація рівнів, система прогресії персонажа та масштабування складності. Метою є створити гру з динамічним ігровим процесом, що забезпечує випадковість і повторну цінність проходження.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. провести аналіз жанру Roguelite і порівняти ключові механіки відомих ігор цього жанру;
2. дослідити та обґрунтувати вибір ігрового рушія Unity, мови C# та інструментів розробки;
3. спроєктувати архітектуру програмного забезпечення гри, виділивши основні модулі (гравець, вороги, рівень, інтерфейс тощо);
4. реалізувати систему керування персонажем та бойову систему (рух, стрільба, атаки ворогів);
5. розробити механізм формування рівнів на основі попередньо створених кімнат (префабів) з динамічним поєднанням;
6. реалізувати систему розвитку персонажа (збирання кристалів, бонуси, покращення характеристик);
7. розробити алгоритм динамічного масштабування складності, що залежить від часу або прогресу гравця;
8. провести тестування реалізованого програмного продукту та оцінити його відповідність вимогам.

Під час створення сучасних комп'ютерних ігор особлива увага приділяється забезпеченню балансу між складністю проходження та зацікавленістю

користувача. Надмірна складність може призвести до втрати інтересу гравця, тоді як занадто простий ігровий процес знижує мотивацію до подальшого проходження. Саме тому в іграх жанру Roguelite широко використовуються механізми поступового ускладнення ігрового процесу, системи випадкових покращень та адаптивні алгоритми розвитку персонажа.

Важливою перевагою комп'ютерних ігор із процедурною генерацією є висока реіграбельність. Кожне нове проходження формує унікальний набір подій, ворогів та конфігурацій рівнів, що забезпечує різноманітність ігрового досвіду. У результаті користувач отримує можливість експериментувати з різними стратегіями проходження та способами розвитку персонажа, що позитивно впливає на тривалість взаємодії з програмним продуктом.

Об'єктом розробки є розважальний програмний продукт - комп'ютерна гра, а предметом - сукупність програмних модулів та алгоритмів, які забезпечують функціонування геймплею гри «Etemenamki». У результаті виконання роботи очікується створення цілісного програмного продукту, що демонструє основні засоби об'єктно-орієнтованого проєктування та сучасні методи розробки комп'ютерних ігор, які є важливою складовою сучасних інформаційних технологій [4]

Розробка комп'ютерних ігор є складним процесом, який поєднує програмну інженерію, проєктування програмного забезпечення та створення інтерактивних систем реального часу. Особливістю ігор жанру Roguelite є необхідність забезпечення балансу між випадковістю ігрових подій та збереженням інтересу користувача протягом усього проходження. Для досягнення цієї мети необхідно реалізувати механізми генерації рівнів, масштабування складності та системи розвитку персонажа, які взаємодіють між собою та формують цілісний ігровий процес.

Важливим аспектом розробки є побудова модульної архітектури програмного забезпечення, яка дозволяє розділити функціональність гри на окремі підсистеми. Такий підхід спрощує підтримку програмного коду, полегшує тестування окремих компонентів та забезпечує можливість подальшого

розширення проєкту шляхом додавання нових механік, типів противників та ігрових об’єктів без необхідності суттєвої зміни вже реалізованих модулів.

Сучасні ігрові рушії надають широкий набір інструментів для реалізації складних програмних систем у сфері розробки відеоігор. Зокрема, використання рушія Unity та мови програмування C# дає змогу застосовувати принципи об’єктно-орієнтованого програмування, компонентний підхід до проєктування та сучасні засоби керування ресурсами, що значно спрощує процес розробки та подальшого розширення функціональності гри.

Практична цінність роботи полягає у створенні повнофункціонального програмного продукту, який може бути використаний як основа для подальшого розвитку проєкту шляхом додавання нових типів ворогів, ігрових механік, предметів та систем прогресії. Отримані під час виконання роботи результати також можуть бути використані для дослідження методів процедурної генерації контенту та побудови масштабованих ігрових систем.

1.2. Аналіз жанру Roguelite

Поняття «Roguelike» походить від гри Rogue (1980) і передбачає такі типові елементи, як перманентна смерть персонажа (при смерті гравець починає гру заново з початку), процедурна генерація підземель та покрокова стратегічна система бою. Roguelite - це похідний жанр, який розслабляє деякі з цих обмежень. У Roguelite іграх може бути полегшена система смерті (наприклад, гравець зберігає частину досягнутого прогресу або отримує деякі бонуси при кожному новому запуску), більш динамічний (реального часу) бій та інтуїтивніший інтерфейс [12]. Водночас такі ігри зберігають високий рівень випадковості: змінювані згенеровані рівні, множинні комбінації поліпшень та нелінійний прогрес.

Ключові механіки жанру Roguelite:

1. Смерть персонажа. Зазвичай наявна перманентна або умовна смерть. Наприклад, при смерті гравець втрачає більшу частину здобутків (або починає новий забіг із певними бонусами/поліпшеннями), що стимулює багаторазове

проходження. Деякі Roguelite передбачають "м’яку перманентну смерть", коли гравець зберігає накопичені ресурси чи відкриті навички між іграми.

2. Випадкові покращення та здобич. Розкидані по рівнях ресурси або вибір випадкових бонусів після боїв дають змогу гравцю оновлювати свого персонажа в процесі проходження. Наприклад, найдобуваються випадкові карти навичок, зброї, модифікатори характеристик.

3. Система прогресії між програваннями. Щоб зменшити фрустрацію від перманентної смерті, багато ігор жанру передбачають прокачування між забігами: це можуть бути постійні поліпшення зброї, відкриття нових персонажів, відкриття бонусів тощо. Завдяки цьому проходження стає поступово легшим або різноманітнішим.

4. Повторна цінність (реіграбельність). Основна мета Roguelite - забезпечити цікавість до кількох ігор поспіль через змінність контенту. Це досягається комбінаціями процедурності, великою кількістю варіантів розвитку та різноманітним дизайном ворогів і подій.

Типові особливості сучасних Roguelite-ігор:

1. процедурна або модульна генерація рівнів;
2. система постійних або часткових покращень між забігами;
3. випадкове формування наборів здібностей та предметів;
4. поступове підвищення складності проходження;
5. наявність унікальних противників та босів;
6. стимулювання багаторазового проходження гри.

Поєднання наведених особливостей дозволяє створювати ігрові системи, здатні тривалий час підтримувати інтерес користувача. Саме тому жанр Roguelite залишається одним із найпопулярніших напрямів сучасної індустрії комп’ютерних ігор та активно використовується як незалежними розробниками, так і великими студіями.

Таблиця 1.1

Порівняння ігор жанру Roguelite

Гра	Рік	Платформи	Генерація рівнів	Система прогресії	Основні механіки бою
Hades	2020	ПК, консолі	Збірка зі шаблонів	Постійна (видача скарбів, оновлень між забігами)	Реальний час, ближній/дальній бій
Vampire Survivors	2022	ПК, мобільні, консолі	Зазвичай статична карта (фіксована сцена) з випадковим спавном ворогів	Мішана (тимчасове прокачування в рівні + часткові відкриття назавжди)	Автоатаки (одночасна стрільба кількома напрямами)
Brotato	2022	ПК, консолі	Збірка зі шаблонів (безкінечний рівень з випадковими елементами)	Мішана (розблокування персонажів і поліпшень між раундами)	Двохджойстиковий шутер з далекобійною зброєю
Enter the Gungeon	2016	ПК, консолі	Процедурна (рандомні кімнати з фіксованих модулів)	Постійна (нові предмети та перки відкриваються назавжди)	Двохджойстиковий шутер (стрільба, ухиляння)
The Binding of Isaac	2011	ПК, консолі	Повна процедурна (рандомні кімнати)	Постійна (відкриття предметів, прогрес між іграми)	Двохджойстиковий шутер (головний герой зі сльозами стріляє)

Проведений аналіз, що пказаний в таблиці 1.1 свідчить, що всі перераховані ігри забезпечують високу реіграбельність насамперед за рахунок випадковості. Характерна відмінність полягає в типі генерації рівнів: наприклад, The Binding of Isaac використовує повністю процедурне генерування, тоді як Vampire Survivors обмежується практично фіксованою картою з рандомними подіями. У багатьох сучасних Roguelite, особливо зроблених на Unity, для зручності розробників використовують підхід зі збиранням рівня з готових кімнат-префабів. Це дозволяє ретельно контролювати дизайн кожної кімнати, забезпечує швидке створення рівнів і простоту відлагодження, одночасно зберігаючи елемент випадковості у виборі порядку та параметрів кімнат.

1.3. Аналіз існуючих рішень

Аналіз існуючих ігрових рішень показує, що багатьом проектам жанру Roguelite притаманне використання схожих архітектурних та програмних підходів. Наприклад, для побудови рівнів часто застосовують модульні шаблони: набір заготовлених кімнат-префабів, які можуть бути згенеровані випадковим чином (як у Hades та The Binding of Isaac). У сучасних іграх на Unity бібліотека Tilemap дозволяє автоматизувати процес розстановки тайлів при генерації та легко змінювати дизайн кімнат. Відомі інструменти, такі як ProBuilder або сторонні пакети з Unity Asset Store (наприклад, Dungeon Architect), дають розробникам готові рішення для створення процедурних підземель, однак у нашому випадку була обрана саморобна система через більший контроль над геймплеєм.

Щодо системи ворогів: у багатьох іграх застосовують кінцеві автомати станів (FSM - Finite State Machine) для опису поведінки ворогів (патрулювання, переслідування, атака). Наприклад, кожен ворог має стани “Шукає гравця”, “Преслідує”, “Атакує” та “Помер”. У проекті «Etemenamki» відповідний скрипт EnemyFSM реалізує подібний підхід, що узгоджується з поширеною практикою в ігровій розробці.

Інтерфейс користувача в подібних рішеннях зазвичай реалізується за допомогою Unity UI (Canvas, TextMeshPro для тексту, спрайти для іконок ресурсів). У нашій структурі наявна папка UI, де розміщено елементи інтерфейсу (панель здоров’я, таймер тощо), що відповідає типовому підходу розділення коду на модулі. Більшість ігор використовують патерн MVC/MVP для UI: окремо розробляють логіку відображення та взаємодії користувача.

Окремим напрямом розвитку сучасних ігрових систем є застосування подієво-орієнтованої архітектури. Такий підхід передбачає взаємодію окремих модулів через події та повідомлення без жорсткої залежності між компонентами. Використання подібних механізмів дозволяє спростити масштабування програмного забезпечення та полегшує інтеграцію нових функціональних можливостей у вже існуючий проєкт.

У середовищі Unity подібні принципи реалізуються за допомогою системи компонентів MonoBehaviour, ScriptableObject та шаблонів проєктування Singleton або Observer. Застосування таких рішень сприяє підвищенню гнучкості архітектури та забезпечує зручність супроводу програмного коду на всіх етапах життєвого циклу програмного продукту.

Система збереження прогресу часто базується на ScriptableObject або простих текстових конфігураціях (JSON/XML). У подібних іграх також популярні методи збереження (save/load) між сесіями. У згаданому нами проекті зберігання постійних покращень може реалізовуватися через просту структуру, що накопичує значення характеристик персонажа у зовнішньому файлі чи PlayerPrefs.

Окремої уваги заслуговують системи прогресії персонажа, які є однією з ключових особливостей ігор жанру Roguelite, де ключовими елементами успішного ігрового дизайну є прогресія та виклики для гравця [13]. У більшості сучасних проєктів гравцеві надається можливість змінювати стиль проходження за рахунок вибору покращень, предметів або здібностей. Наприклад, у Hades розвиток персонажа реалізується через благословення богів, тоді як у Vampire Survivors основою прогресії виступає комбінація зброї та пасивних бонусів. У грі «Etemenamki» для цього використовується система кристалів різних типів і розмірів, які впливають на характеристики персонажа та дозволяють формувати індивідуальні стратегії проходження.

В цілому, аналіз існуючих рішень дозволяє зробити висновок: архітектуру проекту «Etemenamki» доцільно будувати за принципом модульності. Уже сформовані папки Scripts/Core, Scripts/Enemy, Scripts/Player, Scripts/Inventory, Scripts/UI відповідають цим підходам і полегшують підтримку та розширення проекту. Подальше використання перевірених патернів (FSM для ворогів, подієво-орієнтовані обробники в UI тощо) забезпечить стійкість і логічність коду.

1.4. Вимоги до програмного продукту

Вимоги до гри «Etemenamki» формуються на основі поставлених завдань та специфіки жанру.

Функціональні вимоги:

- Користувацький інтерфейс повинен відображати поточні характеристики персонажа, такі як здоров’я, заряд, рівень складності та наявні ресурси - кристали.
- Система керування персонажем має забезпечувати плавне пересування двовимірним простором і можливість атакувати та стріляти в напрямку курсору.
- Генератор рівнів повинен випадково формувати послідовність кімнат із набору заготовлених шаблонів із системою переходів/телепортів між ними.
- Бойова система має реалізовувати стрільбу снарядами, зіткнення цих снарядів з ворогами призводить до нанесення пошкоджень згідно з їх характеристиками.
- Вороги повинні переслідувати та атакувати персонажа за заданими правилами (на основі FSM) до завінчення здоров’я персонажа чи їхнього знищення.
- Система прогресії має нараховувати кристали за знищення ворогів і дозволяти витрачати їх на покращення між рівнями.
- Система масштабування складності повинна автоматично збільшувати відношення нанесеного і полученого урона

Нефункціональні вимоги:

- Продукт має бути реалізований на рушії Unity та коректно працювати на цільовій платформі Windows
- Показник продуктивності (FPS) повинен залишатися на рівні ≥30 при типовій кількості об’єктів на екрані.
- Інтерфейс повинен бути інтуїтивно зрозумілим та зручним (наприклад, кольори та розміщення UI-елементів не відволікатимуть від гри).
- Система повинна бути модульною та розширюваною (можливість додавати нові типи ворогів, кімнат чи предметів без суттєвих змін у структурі).

Під час формування вимог також враховувалися особливості цільової платформи та обмеження апаратного забезпечення користувачів. Гра повинна забезпечувати стабільну роботу навіть на комп'ютерах середнього рівня продуктивності, що досягається завдяки використанню оптимізованих алгоритмів генерації рівнів, системи приховування неактивних об'єктів та ефективного керування ресурсами рушія Unity. Важливою характеристикою програмного продукту є його розширюваність. Архітектура гри проєктується таким чином, щоб у майбутньому можна було додавати нові типи кімнат, противників, предметів та механіки розвитку персонажа без необхідності суттєвого перепроєктування вже існуючих компонентів системи.

З урахуванням особливостей жанру Roguelite особлива увага приділяється забезпеченню повторної цінності проходження, балансу ігрового процесу та стабільної роботи системи в умовах процедурної генерації контенту. Реалізація наведених вимог дозволить створити програмний продукт, який поєднуватиме динамічний ігровий процес, зручний інтерфейс користувача та можливість подальшого розширення функціональності.

Таким чином, сформульовані вимоги дозволять впевнено перейти до проєктування архітектури та реалізації гри «Etemenamki», забезпечивши необхідний набір функцій та якість користувацького досвіду.

1.5. Висновки по розділу

У першому розділі бакалаврської роботи було проведено аналіз предметної області та сформульовано основні завдання, необхідні для розробки комп'ютерної гри «Etemenamki». Визначено мету, об'єкт і предмет дослідження, а також окреслено ключові етапи створення програмного продукту. Встановлено, що для забезпечення цікавого та динамічного ігрового процесу необхідно реалізувати комплекс взаємопов'язаних механік, серед яких система керування персонажем, бойова система, генерація рівнів, система прогресії та масштабування складності.

Під час аналізу жанру Roguelite було досліджено його походження,

характерні особливості та відмінності від жанру Roguelike. Встановлено, що основними рисами сучасних Roguelite-ігор є процедурне або модульне формування рівнів, наявність елементів випадковості, система розвитку персонажа та висока реіграбельність. Проведене порівняння популярних представників жанру, таких як Hades, Vampire Survivors, Brotato, Enter the Gungeon та The Binding of Isaac, дозволило визначити найбільш ефективні підходи до реалізації ігрових механік та використати їх як основу для проєктування власного програмного продукту.

Аналіз існуючих рішень показав доцільність використання модульної архітектури програмного забезпечення, яка забезпечує зручність супроводу та подальшого розвитку проєкту. Розглянуті підходи до реалізації систем поведінки ворогів, користувацького інтерфейсу, генерації рівнів та збереження прогресу підтвердили ефективність використання компонентного підходу, що підтримується ігровим рушієм Unity. На основі проведеного аналізу було сформовано структуру майбутнього програмного забезпечення та визначено основні підсистеми гри.

У результаті виконання розділу було сформульовано функціональні та нефункціональні вимоги до програмного продукту, які визначають необхідний набір можливостей гри та критерії її якості. Отримані результати стали основою для вибору технологій розробки, проєктування архітектури та подальшої програмної реалізації гри «Etemenamki», що розглядаються в наступних розділах роботи.

РОЗДІЛ 2. МЕТОДИ ТА ЗАСОБИ РОЗРОБКИ

2.1. Вибір ігрового рушія

Для розробки «Etemenamki» розглядалися три популярні рушії: Unity, Unreal Engine і Godot. Внизу наведені порівняння їх особливих якостей:

- Unity - один із найпопулярніших універсальних рушіїв [8]. Має повноцінну підтримку 2D та 3D проєктів і особливо широкі 2D-можливості. Відзначається низьким порогом входу та великою спільнотою, що значно полегшує навчання. Unity підтримує багаточисельні платформи (Windows, macOS, Linux, Web, мобільні, консолі тощо) і має власний Asset Store з тисячами готових ресурсів. Ліцензійна модель передбачає безкоштовну версію Personal (для команд з доходом < $200K), потім - підписку; роялті за продажі відсутні. Unity добре підходить для 2D-ігор завдяки інтегрованим інструментам Tilemap, системі камер і підтримці шаблонів (prefabs). Зокрема, шаблонні кімнати (rooms) можуть зберігатися як префаби, що спрощує процедурну генерацію рівнів (можна вказати у сценарії Unity точку приєднання кімнат і за допомогою prefab динамічно їх інстанціювати). Саме ця можливість легкої роботи з prefab і велика кількість готових 2D-інструментів роблять Unity привабливим вибором для roguelike-проектів типу «Binding of Isaac» чи «Enter the Gungeon».
- Unreal Engine - високопродуктивний рушій, відомий своїми можливостями для AAA-ігор. Має просунуті графічні та системні технології (наприклад, Lumen і Nanite). В Unreal використовується мова C++ і зручна система Blueprint (візуального скриптингу), що дозволяє створювати логіку без кодування. Рушій безкоштовний до $1 млн виручки, після чого застосовується роялті 5% від доходу. Unreal орієнтований на проекти з високими вимогами до графіки; підтримка 2D-ігор існує (Paper2D), але вона не така потужна, як 3D. Наявність готових AAA-функцій і велика співтовариство характерні для Unreal (проте для новачків поріг входу вищий через необхідність вивчення C++).
- Godot - безкоштовний і повністю відкритий рушій з ліцензією MIT.

Працює з 2D та 3D іграми; згодиться для простих і середніх проєктів. Має власні мови - легкий GDScript (схожий на Python) та підтримує C#. Плюсом є відсутність будь-яких обмежень (жодних роялті чи платежів, як зазначено для Godot - «безкоштовний і з відкритим кодом, без умов і роялті»). Однак спільнота Godot менша, а кількість готових ресурсів та документації поступається Unity. Godot зручний для 2D-проєктів (він розроблявся з акцентом на 2D), але на момент розробки «Etemenamki» він не мав настільки ж розвинених інструментів, особливо для шаблонної генерації рівнів чи готових плагінів.

Враховуючи цільову платформу (Windows) та жанр (2D roguelite), найбільш вигідним виявився Unity. Він володіє широким набором засобів для 2D (Canvas, Tilemap, SpriteRenderer), низьким порогом входу, підтримкою C# і вбудованими шаблонами (prefab) для кімнат (що і дозволяє легко організувати процедурну генерацію рівня). Unity надає великий Asset Store з готовими наборами скриптів та графіки, що скорочує час розробки. Наприклад, в дискусіях Unity рекомендують створювати шаблони кімнат у вигляді prefabs і додавати до них скрипти з інформацією про точки входу/виходу, потім динамічно інстанціювати сумісні кімнати поруч. Враховуючи також власний досвід і навички команди, вибір на користь Unity цілком обґрунтований: він легко освоюється, має потужну спільноту розробників (Unity є одним з найпопулярніших рушіїв у світі) та забезпечує гнучкість для 2D-механік.

2.2. Вибір мови програмування

Unity при першому запуску зразу підтримує C# як основну мову скриптів. Решта мов - це застарілий UnityScript (JavaScript-подібний) і Boo (подібний до Python), які вже видалено, а також можливість використовувати C++ через плагіни. C# є типовою для Unity, і саме нею писали більшість ігрової логіки [5]. Таблиця нижче порівнює C# з альтернативами (C++ і GDScript):

- C# - мовна платформа від Microsoft (частина .NET), призначена для об’єктно-орієнтованого програмування [1] Потієнко О. І. Програмування мовою C#.

Має сильну статичну типізацію, збалансовану продуктивність та потужну бібліотеку класів. Вона повністю інтегрована з Unity (API рушія написане на C#, повний доступ) та має численні засоби IDE (intellisense, рефакторинг). Синтаксис C# простіший, ніж у C++, що важливо для швидкого старта. Кодування на C# не вимагає ручного управління пам'яттю (є garbage collector). Працюючи в Unity, розробник використовує C#-скрипти для контролю поведінки ігрових об'єктів наприклад, приклад використання [10]:

```
using UnityEngine;
using UnityEngine.InputSystem;
public class PlayerController : MonoBehaviour {
   public void OnMove(InputValue value) {
      Vector2 v = value.Get<Vector2>();
      transform.Translate(v * Time.deltaTime * speed);
   }
}
```

Unity надає вбудовані компоненти і класи (MonoBehaviour, Transform тощо) для C#.

- C++ - низькорівнева компільована мова з максимальною продуктивністю. Використовується в Unreal Engine (та є можливість додати C++ код через плагін або власну модульну збірку Unity). C++ дає контроль над пам'яттю і вищу швидкість виконання, але складніший у вивченні й відлагодженні. У контексті Unity застосовується рідко, головним чином для створення плагінів чи розширень. Для «Etemenamki» C++ не обрали, оскільки він не сумісний із звичайним workflow Unity (вимагає створення DLL-модулів) і є занадто «низькорівневим» для простого roguelite.
- GDScript - спеціальна скриптова мова рушія Godot, синтаксично схожа на Python. Вона дуже проста і динамічна, добре підходить для швидких прототипів. Однак GDScript повільніша від C# (інтерпретується) і практично не використовується поза Godot. (Godot також підтримує C#, але його екосистема обмежена лише цим рушієм.) Для порівняння, у Godot діє сумісність з C#, але без

Unity API.

З урахуванням вимог проєкту, C# був однозначним вибором. По-перше, він є основною мовою Unity - вся документація та практично всі інструменти Unity орієнтовані саме на C#. По-друге, C# простіше для розробників початкового та середнього рівня, ніж C++. По-третє, C# має великий набір бібліотек (.NET, Unity API, багато open-source-пакетів), а його интеграція з Unity повна: засоби автодоповнення й інспектори прекрасно розпізнають C# код. Таким чином, для підсистем гри (вороги, гравець, інтерфейс тощо) було обрано C# з урахуванням гнучкості й зручності розробки.

2.3. Вибір середовища розробки

Основними кандидатами для IDE були Visual Studio (від Microsoft), Rider (від JetBrains) та Visual Studio Code (VSCode). Наведемо порівняння їхніх характеристик:

- Visual Studio - потужне повнофункціональне середовище (IDE) для C#/.NET. Надає найкращі можливості відлагодження (breakpoints, watch, моментальні вирази, профайлер) і відмінну інтеграцію з Unity (офіційний Unity Tools плагін дозволяє приєднуватися до процесу гри, слідкувати за ходом виконання). Рефакторинг у Visual Studio також дуже розвинений (перейменування, переміщення кодових фрагментів, аналіз коду). Visual Studio Community Edition безкоштовна для індивідуального і некомерційного використання, а платні Professional/Enterprise видання - для великих організацій. Мінус - висока пам’ятевіддача та відносна «вагомість» IDE. Проте саме Visual Studio була обрана як основна: вона забезпечує надійну та офіційно підтримувану платформу, з якою Unity має максимальну сумісність.
- Rider - кросплатформенний C# IDE від JetBrains, побудований на ReSharper. Має не менш потужні інструменти аналізу коду і рефакторингу, часто вважається «більш дотованим» у сфері продуктивності при роботі з великими проектами. Rider добре інтегрується з Unity за допомогою вбудованих плагінів

(налаштування сцени, профіліг ігор, вікна Unity Console тощо). З переваг - швидкий пошук за кодом, відмінна підтримка Unity API і можливість відлагодження, як у Visual Studio. Недоліком є ліцензія: Rider є платним продуктом (проте надається безкоштовно для студентів і open-source розробників). Оскільки в нашому випадку має сенс працювати з безкоштовним інструментом і є досвід використання Visual Studio, було вирішено саме її використовувати, хоча частина команди знайома з Rider (його можна застосовувати в майбутньому для покращення продуктивності розробки).

- Visual Studio Code (VSCode) - легковагий редактор коду з розширеннями. Він безкоштовний і легкий, забезпечує базовий інтелісенс та інтеграцію з Git. Проте VSCode по замовчуванню не має такого ж рівня підтримки Unity (до недавнього часу існував Unity Tools Extension, але Microsoft призупинила розробку плагінів для Unity Debugger). Іншими словами, нативна підтримка Unity у VSCode в останніх версіях обмежена. Користувачі відзначають, що Visual Studio і Rider мають більше функцій, а VSCode є швидшим і легшим, але «заточеним» більше під загальне кодування, ніж під повноцінне відлагодження Unity. Для «Etemenamki» було вирішено застосувати Visual Studio як основне IDE, базуючись на рекомендаціях і офіційній підтримці: саме MS Visual Studio Community дозволяє безкоштовно використовувати всі функції і є стандартом у Unity-розробці. VSCode може використовуватися для легкого редагування (воно швидке), але основна розробка і налагодження реалізовувались у Visual Studio.

Отже, вибір IDE вплинув на підвищення ефективності розробки: за допомогою Visual Studio доступні повнофункціональні засоби налагодження Unity (Attach to Unity, Watch, Breakpoints) та зручна інтеграція з C#-інспекторами. Rider/VSCode розглядалися як альтернативи, але упор зроблено на Visual Studio через офіційну підтримку і відсутність додаткових витрат.

2.4. Використані пакети Unity

У проєкті «Etemenamki» задіяні як стандартні пакети Unity, так і кілька

сторонніх бібліотек. Нижче наведено опис основних з них та їх роль у розробці.

Input System - нова система вводу, що заміняє класичний Input Manager. Вона більш гнучка та дозволяє налаштовувати діалоги (bindings) для будь-яких пристроїв (клавіатура, миша, геймпад, сенсорний екран тощо). У документації Unity описано: «Input System - нова система, більш гнучка й налаштовувана, що може використовуватися замість класичної UnityEngine.Input». У проєкті вона використовується для обробки руху гравця та стрільби. Приклад коду з Input System:

```
using UnityEngine;
using UnityEngine.InputSystem;
public class PlayerControls : MonoBehaviour {
   public float speed = 5f;
   private Vector2 moveDir;
   public void OnMove(InputAction.CallbackContext context) {
      moveDir = context.ReadValue<Vector2>();
   }

   void Update() {
      transform.Translate(new Vector3(moveDir.x, moveDir.y, 0) * speed * Time.deltaTime);
   }
}
```

Тут OnMove спрацьовує, коли гравець натискає клавіші WASD або аналоговий стік, налаштований у InputAction. Система автоматично забезпечує доступ до стандартних пристроїв без додаткового коду. Input System дозволяє використовувати події і action maps, що робить код більш декларативним. Для проекту підключення цього пакету вимагало лише інсталяції через Package Manager та створення Input Actions.

TextMesh Pro - набір інструментів для високоякісного відображення тексту у 2D/3D. Згідно з офіційною документацією, «TextMesh Pro - це набір інструментів

Unity для 2D і 3D тексту, який надає кращий контроль над форматуванням і макетом тексту, ніж звичайні системи UI Text». Фактично TextMesh Pro замінив стандартні Text в Unity, даючи можливість використовувати набори шрифтів (Font Assets), підтримку кернінгу, багату розмітку (кольори, тіні, контури) тощо. У грі за допомогою TextMeshPro відображаються написи в інтерфейсі (кількість кристалів, інформація про гравця тощо). Інтеграція проста: потрібно імпортувати TMP Essentials і додати компонент TextMeshProUGUI до UI-елементів, далі через код можна керувати текстом:

```
using TMPro;
[SerializeField] TextMeshProUGUI crystalCountText;
void UpdateCrystalText(int count) {
   crystalCountText.text = $"Crystals: {count}";
}
```

TextMeshPro також має власні шейдери і оптимізації для чіткого відображення шрифту.

Tilemap - інструментарій Unity для створення 2D-карт з тайлів. Дозволяє «малювати» рівні, наповнюючи грід (сітку) графічними спрайтами (тайлами). Як зазначено в документації: «Tilemap - це GameObject, що дозволяє швидко створювати 2D рівні за допомогою тайлів і накладання сітки». У проєкті цей пакет використовується для генерації кімнат: кожна кімната формувалася на базі кількох тайлів (стіни, підлога, перешкоди). Наприклад, для кожної сгенерованої кімнати динамічно інстанціюються потрібні Tilemap-компоненти зі скриптом, що задає кількість та розташування плиток. Код роботи з Tilemap може виглядати так (приклад наповнення тайлами):

```
using UnityEngine;
using UnityEngine.Tilemaps;
public class Room : MonoBehaviour {
   public Tilemap floorMap;
   public TileBase floorTile;
```

```
    public void FillFloor(int width, int height) {
        for (int x = 0; x < width; x++) {
            for (int y = 0; y < height; y++) {
                floorMap.SetTile(new Vector3Int(x, y, 0), floorTile);
            }
        }
    }
}
```

Це демонструє, як через Tilemap script можна задавати плани кімнат. Tilemap полегшує ручне та автоматичне формування рівнів з тайлів без написання складної математики рендерингу.

Інші Asset Store пакети. У нашому проєкті сторонніх графічних чи механічних пакетів майже не використовувалося, а замість цього обрано роботу власних скриптів. В архіві проекту присутні власні класи RoomManager, DifficultyTimer, CrystalSystem тощо - це не зовнішні плагіни, а власні модулі. Вони надають функціональність (управління кімнатами, поступове ускладнення гри, система ресурсів “кристали”) і реалізовані за допомогою стандартного скриптингу Unity (скрипти в папках Assets/Scripts). Тобто з боку Unity Package Manager було задіяно вищезгадані пакети (InputSystem, Tilemap тощо), а з боку Asset Store - тільки DOTween (ця бібліотека розповсюджується як Asset). Інших зовнішніх майже не застосовано.

2.5. Архітектура програмного забезпечення

Архітектура проекту збережена в окремому контейнері скриптів, та поділяється на п'ять основних модулів, як показано на рисунку 2.1.

Модульна структура спрощує масштабування та супровід системи [3].

Core/Рівні: Модуль «Core» (Assets/Scripts/Core) включає класи Room, RoomManager, TeleportPortal, EnemyData.

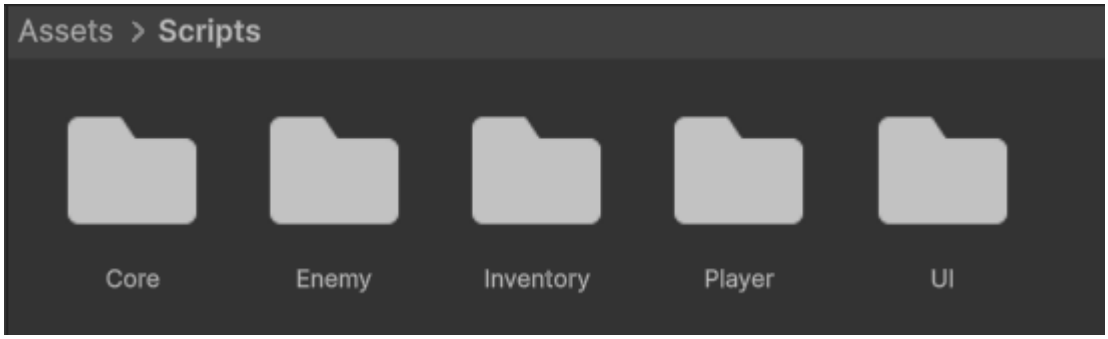

Рисунок 2.1 - Папка скриптів проекту

Вони відповідають за формування рівня: RoomManager керує списком кімнат і їх генерацією (міжкімнатні переходи, ланцюжок кімнат); Room містить дані про розміри та тайли конкретної кімнати та методи її наповнення (наприклад, заповнення Tilemap)【50†L40-L49】; TeleportPortal - об’єкт для переходу між кімнами; EnemyData зберігає базові характеристики ворогів (здоров’я, шкода). Ці класи згруповані в папці Assets/Scripts/Core, як зображено на рисунку 2.1.

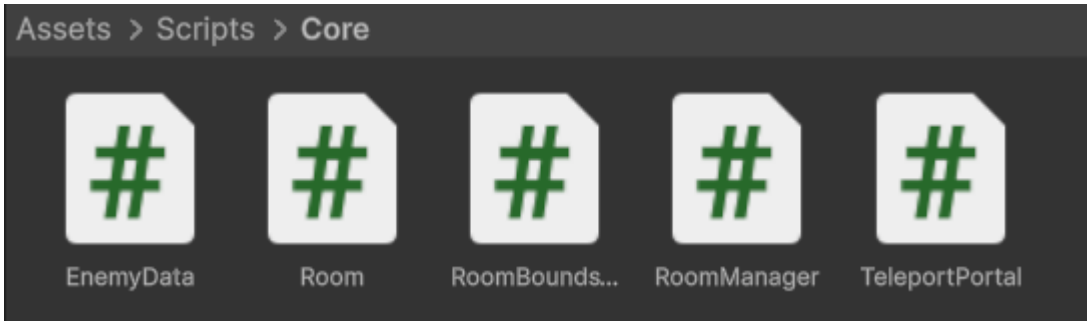

Рисунок 2.2 - Скрипти папки Core

Enemy (Вороги): У папці Assets/Scripts/Enemy зображеній на рисунку 2.3 містяться класи EnemyBrain, EnemyState, EnemyMove, EnemyHealth та їх підкласи. Тут реалізується штучний інтелект ворогів. EnemyBrain координує різні стани ворога (патруль, переслідування, атака), звертаючись до допоміжних класів EnemyMove та EnemyHealth. EnemyState визначає окремі стани ворога як окремі об’єкти, що змінюють поведінку. Наприклад, при попаданні в певну зону активується новий стан. Префаб ворога містить ці компоненти і налаштовані параметри. Також для кожного ворога були зроблені свої скрипти Атаки та руху

що підтягуються і використовується в загальних скриптах

Рисунок 2.3 - Загальні скрипти ворогів та папки для унікальних

Inventory (Інвентар): Папка Assets/Scripts/Inventory, зображена на рисунку 2.5, містить класи CrystalInventory і EquipmentManager. CrystalInventory - система зберігання ресурсів (кристалів); відповідає за рахунок і збільшення кристалів при зборі. EquipmentManager (хоч він і не акцентуючийся на кристалах) міг би відповідати за екіпірування чи покращення, хоча в цій версії гри він може містити методи активації покращень гравця (що відбувається при досягненні певних цілей). Префаби ресурсу «кристал» містять Collider для збирання і викликають метод CrystalInventory.AddCrystals() при підборі.

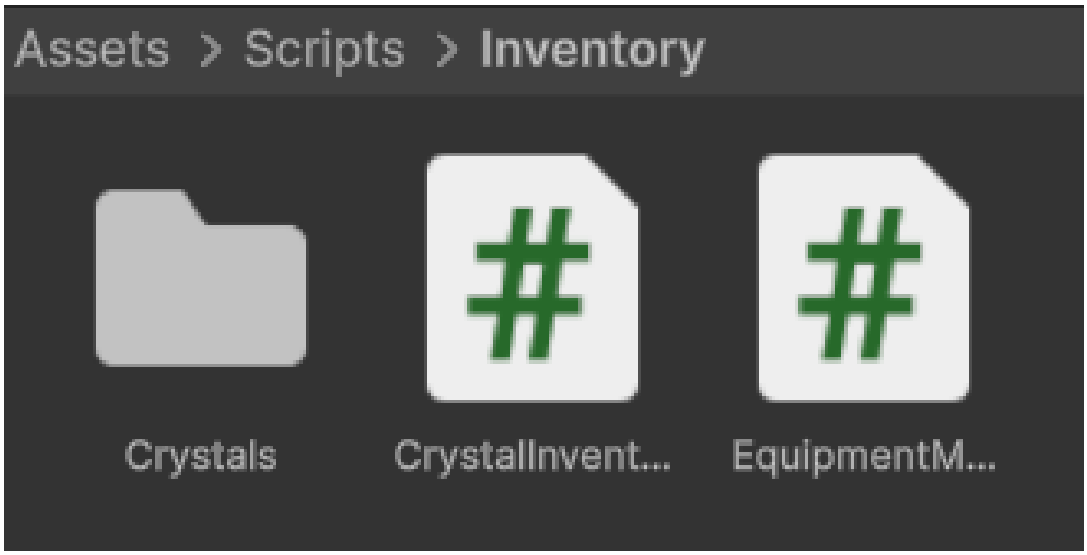

Рисунок 2.4 - Скрипти інвентаря

Player (Гравець): У Assets/Scripts/Player - класи PlayerMovement, Attack, PlayerStatSystem. PlayerMovement обробляє рух (використовуючи Input System), Attack - механіку стрільби по ворогах (створення куль, нанесення шкоди), PlayerStatSystem тримає характеристики гравця (здоров'я, швидкість тощо). Ці компоненти прикріплено до префабу гравця. Всі скрипти зберігаються в окремій папці проекту показаній на рисунку 2.5

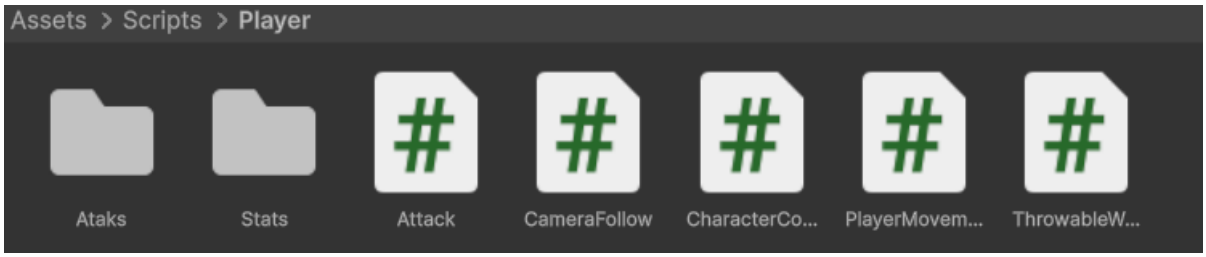

Рисунок 2.5 - Скрипти персонажа

UI: Папка Assets/Scripts/UI, зображена на рисунку 2.6 містить класи InventoryUI, DifficultyTimer, CrystalSlotUI. InventoryUI відображає на екрані інформацію про ресурси та елементи інвентарю, оновлюючи відповідні UI-елементи . CrystalSlotUI - підкомпонент UI, що показує поточну кількість кристалів. DifficultyTimer - логіка поступового збільшення складності (з часом підвищує параметри ворогів. Prefab'и Canvas'у містять ці скрипти: наприклад, GameObject з UI Canvas має компонент InventoryUI, який через посилання оновлює віджети. Подібне структурування ігрових даних підвищує ефективність програмної системи [6]

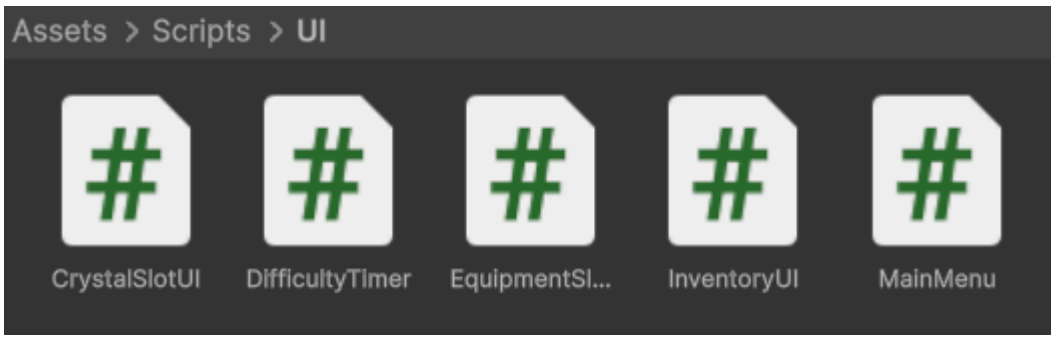

Рисунок 2.6 - Скрипти керування UI

Усі ці скрипти та модулі взаємодіють згідно з принципами ООП [2]: кожен виконує чітко визначені обов’язки. Наприклад, при створенні нової кімнати (RoomManager) відбувається інстанціювання префабу кімнати і виклик методів класу Room для заповнення Tilemap. Відбувається збереження зв’язків між порталами та сусідніми кімнатами. Вороги при появі отримують посилання на гравця і в кожному Update() викликають NavMeshAgent.SetDestination, забезпечуючи переслідування (використовуючи модуль AI Navigation). Після смерті ворога повідомляється CrystalInventory про нагороду, і UI оновлюється через CrystalSlotUI.

2.6. Висновки по розділу

У другому розділі було проведено обґрунтований вибір технологій та програмних засобів, необхідних для розробки комп’ютерної гри «Etemenamki». На основі порівняння сучасних ігрових рушіїв встановлено, що Unity найбільшою мірою відповідає вимогам проєкту завдяки розвиненим засобам створення двовимірних ігор, підтримці процедурної генерації рівнів, наявності великої кількості навчальних матеріалів та широкої спільноти розробників. Додатковими перевагами стали підтримка префабів, інтегровані інструменти роботи з Tilemap та можливість швидкого створення прототипів ігрових механік.

У процесі аналізу мов програмування було визначено, що найбільш доцільним рішенням для реалізації проєкту є використання мови C#. Її повна інтеграція з Unity, підтримка об’єктно-орієнтованого програмування та наявність широкого набору бібліотек забезпечують зручність розробки та подальшого супроводу програмного коду. Для створення та налагодження програмних модулів було обрано середовище розробки Microsoft Visual Studio, яке надає ефективні засоби відлагодження, рефакторингу та аналізу коду, а також підтримує безпосередню взаємодію з проєктами Unity.

Також було розглянуто програмні пакети та інструменти Unity, які використовувалися під час створення гри. Зокрема, Input System забезпечує

сучасну систему обробки користувацького вводу, TextMesh Pro використовується для відображення текстових елементів інтерфейсу, Tilemap дозволяє ефективно створювати та наповнювати ігрові кімнати, а DOTween застосовується для реалізації анімаційних ефектів. Використання готових та перевірених рішень дало змогу зменшити обсяг допоміжного коду та зосередитися на реалізації власних ігрових механік.

У результаті було сформовано архітектуру програмного забезпечення, що базується на принципах модульності та розподілу відповідальності між окремими підсистемами. Виділення модулів Core, Enemy, Inventory, Player та UI дозволило логічно організувати структуру проєкту, спростити підтримку програмного коду та забезпечити можливість подальшого розширення функціональності гри.

Проведений аналіз засобів розробки показав, що поєднання рушія Unity, мови програмування C# та середовища Microsoft Visual Studio утворює ефективну платформу для створення сучасних комп’ютерних ігор. Використання цих технологій дозволяє реалізовувати складні ігрові механіки, забезпечувати зручний процес налагодження програмного коду та підтримувати високу продуктивність програмного продукту на цільовій платформі.

Обрані технологічні рішення створюють передумови для подальшого розвитку проєкту та впровадження нових функціональних можливостей. Завдяки модульній структурі та використанню сучасних інструментів розробки програмне забезпечення може бути адаптоване до інших платформ, розширене новими типами контенту та доповнене механізмами довготривалої прогресії, що є перспективним напрямом розвитку гри «Etemenamki».

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ГРИ

3.1. Система керування персонажем

Однією з ключових складових будь-якої комп’ютерної гри жанру roguelite є система керування персонажем, оскільки саме вона забезпечує безпосередню взаємодію гравця з ігровим світом. Від якості реалізації механіки пересування залежить комфорт керування, швидкість реакції користувача та загальне сприйняття ігрового процесу. Тому під час розробки гри особлива увага приділялася створенню інтуїтивно зрозумілої та плавної системи руху персонажа.

Основним завданням підсистеми керування є обробка введення користувача, перетворення отриманих команд у відповідні дії персонажа та синхронізація цих дій із фізичним рушієм Unity. Для реалізації зазначеної функціональності було створено окремий програмний компонент PlayerMovement, який відповідає за пересування персонажа, обробку стрибків та взаємодію з фізичним середовищем.

При розробці було використано компонентний підхід, що є базовим принципом рушія Unity. Згідно з цим підходом кожна підсистема реалізується окремим компонентом, який може бути доданий до ігрового об'єкта. Такий підхід забезпечує високу модульність проєкту, спрощує супровід програмного коду та дозволяє незалежно розширювати окремі елементи функціональності.

Для реалізації фізики руху персонажа використовується компонент Rigidbody2D, який входить до складу рушія Unity [9]. Даний компонент забезпечує коректну обробку прискорення, гравітації, зіткнень з іншими об'єктами сцени та взаємодію з фізичними матеріалами. Використання вбудованого фізичного рушія дозволило відмовитися від ручного розрахунку переміщень та забезпечити природну поведінку персонажа в межах ігрового середовища.

Обробка переміщення здійснюється в реальному часі шляхом аналізу стану клавіш керування. Після отримання даних про натискання відповідних клавіш формується горизонтальний вектор руху, який передається до фізичного

компонента персонажа. На основі цього вектора відбувається зміна швидкості об'єкта вздовж горизонтальної осі. Такий підхід забезпечує плавне пересування та миттєву реакцію персонажа на дії користувача.

Важливою складовою системи є механізм зміни напрямку погляду персонажа. У двовимірних іграх платформного типу зміна напрямку руху зазвичай супроводжується дзеркальним відображенням спрайта. Для реалізації даної функції використовується масштабування об'єкта вздовж осі X. При зміні напрямку руху значення масштабу автоматично інвертується, що створює візуальний ефект повороту персонажа без необхідності використання окремих графічних ресурсів.

Окрему роль у системі керування відіграє механіка стрибка. Для виконання стрибка необхідно визначити, чи знаходиться персонаж на поверхні. З цією метою реалізовано перевірку контакту із землею за допомогою спеціальної контрольної точки та механізму фізичного сканування простору. Після підтвердження контакту із землею система дозволяє виконання стрибка шляхом прикладання вертикального імпульсу до компонента Rigidbody2D.

Використання перевірки наявності опори дозволяє уникнути багаторазового виконання стрибка у повітрі та забезпечує коректну поведінку персонажа під час проходження рівнів. Такий механізм широко застосовується в сучасних платформерах та забезпечує передбачувану реакцію системи на дії гравця.

Для підвищення якості ігрового процесу система руху також інтегрована з анімаційною підсистемою. Поточний стан персонажа передається до компонента Animator, який відповідає за відображення відповідних анімацій. Залежно від швидкості переміщення та факту знаходження в повітрі система автоматично переключає анімаційні стани між режимами очікування, руху та стрибка. Завдяки цьому досягається узгодженість між логікою гри та її візуальним представленням.

Крім базових функцій переміщення, система керування взаємодіє з іншими підсистемами проєкту. Зокрема, модуль руху пов'язаний із системою характеристик персонажа, яка може змінювати швидкість пересування залежно від отриманих покращень або рівня складності гри. Така інтеграція забезпечує можливість масштабування функціональності без необхідності суттєвої

модифікації основного програмного коду.

З точки зору користувача керування реалізовано за допомогою стандартного набору клавіш, характерного для більшості двовимірних екшен-ігор. Переміщення персонажа виконується клавішами A та D, стрибок здійснюється натисканням клавіші Space, відкриття інвентарю виконується клавішею Tab. Така схема була обрана через її поширеність серед гравців та відсутність необхідності додаткового навчання користувача.

Таким чином, реалізована система керування персонажем забезпечує плавне та передбачуване пересування, коректну взаємодію з фізичним середовищем, підтримку анімацій та можливість інтеграції з іншими ігровими механіками. Модульна структура рішення дозволяє легко розширювати функціональність у майбутньому та підтримує подальший розвиток проєкту без значних змін існуючої архітектури.

3.2. Бойова система

Бойова система є однією з центральних складових гри «Etemenamki», оскільки саме вона формує основну взаємодію між гравцем та ігровим світом. У жанрі Roguelite бойові механіки безпосередньо впливають на динаміку проходження, складність і рівень залучення користувача до ігрового процесу. Під час розробки гри основною метою було створення простої для освоєння, але достатньо гнучкої системи бою, яка дозволяє поєднувати дистанційні та ближні атаки залежно від ситуації на рівні.

Особливістю реалізованої бойової системи є використання двох незалежних типів атак. Основним засобом боротьби з противниками виступає дистанційна атака, яка активується натисканням лівої кнопки миші. Додатково реалізовано ближню атаку мечем, що виконується правою кнопкою миші та використовується під час безпосереднього контакту з ворогами. Такий підхід дозволяє урізноманітнити ігровий процес та надає користувачеві можливість адаптувати стиль гри відповідно до поточної ситуації.

Для реалізації дистанційної атаки було створено окрему систему снарядів. Після отримання команди від користувача відбувається створення нового ігрового об'єкта, який представляє собою проєктиль. Положення створення снаряда визначається поточним розташуванням персонажа, а напрямок руху обчислюється відповідно до позиції курсора миші на ігровому полі. Завдяки цьому забезпечується інтуїтивно зрозуміле керування стрільбою та висока точність наведення.

Після створення снаряд отримує вектор швидкості та починає рухатися у визначеному напрямку. Для обробки фізичних взаємодій використовуються стандартні засоби фізичного рушія Unity. При зіткненні з противником відбувається передача значення пошкодження до системи здоров'я ворога, після чого проєктиль автоматично видаляється зі сцени. Додатково було реалізовано перевірки, що запобігають небажаній взаємодії снарядів із деякими допоміжними об'єктами рівня, зокрема предметами збору ресурсів. Розмір пошкодження, що наноситься противнику, не є сталою величиною. Його значення формується на основі характеристик персонажа та модифікується системою масштабування складності. Схему взаємодії снаряду з ворогами зображено на рисунку 3.1. Такий підхід забезпечує узгоджену роботу всіх підсистем гри та дозволяє підтримувати баланс між силою персонажа і складністю проходження.

Окрім дистанційного бою в грі реалізовано механіку ближньої атаки. Її основним призначенням є надання гравцеві альтернативного способу боротьби з ворогами на короткій дистанції. На відміну від проєктилів, мечова атака використовує перевірку області ураження навколо персонажа. Під час виконання атаки система визначає всі об'єкти противників, які потрапили до зони дії удару, після чого кожному з них передається відповідне значення пошкодження.

Рисунок 3.1 - Схема роботи дистанційної атаки

Застосування двох різних механік атаки дозволяє створити більш варіативний ігровий процес. Дистанційний бій забезпечує безпечне знищення цілей на відстані, тоді як ближній бій дозволяє ефективно реагувати на ситуації, коли вороги підходять занадто близько до персонажа. Подібне поєднання механік часто використовується в сучасних представниках жанру Roguelite та сприяє підвищенню динамічності гри.

Для реалізації поведінки противників бойова система тісно інтегрована з компонентами штучного інтелекту. Кожен ворог використовує набір характеристик, що зберігаються у структурі EnemyData, включаючи показники здоров'я, сили атаки та інші параметри. Завдяки цьому різні типи противників можуть мати унікальні бойові характеристики та вимагати від користувача різних підходів під час проходження гри.

Окрему роль у підтриманні інтересу до ігрового процесу відіграє система візуального та аудіосупроводу бойових дій. Виконання атак супроводжується відповідними анімаціями персонажа та противників, а також відтворенням звукових ефектів. Візуальний зворотний зв'язок дозволяє гравцю швидко оцінювати результати власних дій і підвищує загальне сприйняття бойового процесу. Таким чином, реалізована бойова система забезпечує взаємодію між персонажем і противниками за допомогою поєднання дистанційних та ближніх атак, підтримує механізми отримання і нанесення пошкоджень, інтегрується із системою характеристик та штучним інтелектом ворогів. Використання модульної архітектури дозволило створити гнучку основу для подальшого розвитку проєкту та додавання нових типів зброї, атак і бойових механік у майбутніх версіях гри.

3.3. Штучний інтелект ворогів

Однією з основних складових ігрового процесу є система поведінки противників, яка визначає спосіб їхньої взаємодії з персонажем та навколишнім середовищем. Під час розробки гри «Etemenamki» було реалізовано модульну систему штучного інтелекту, що дозволяє використовувати спільну логіку

керування для декількох типів ворогів із можливістю подальшого розширення функціональності. Основою реалізації виступає компонент EnemyBrain, який відповідає за аналіз поточної ситуації та прийняття рішень щодо подальших дій противника. Робота системи побудована за принципом машини станів (Finite State Machine, FSM), що є одним із найбільш поширених підходів до реалізації ігрового штучного інтелекту. Кожен ворог може перебувати у певному стані, наприклад очікування, пересування, переслідування цілі або атаки. Залежно від положення персонажа, відстані до нього та інших параметрів система автоматично виконує перехід між станами. Такий підхід забезпечує передбачувану та стабільну поведінку противників, а також дозволяє уникнути надмірної складності реалізації, що особливо важливо для невеликих ігрових проєктів.

Важливою особливістю реалізованої архітектури є розділення загальної логіки руху та індивідуальної поведінки окремих типів ворогів. Для цього було створено базовий компонент EnemyMovemen, що містить спільний функціонал пересування та взаємодії з фізичною системою Unity, який наявний в префабі на рисунку 3.2. На основі цього компонента реалізуються окремі механіки руху для різних типів противників, зокрема кажанів, слизів, големів та літаючих черепів. Завдяки використанню успадкування та модульної структури коду вдалося уникнути дублювання функціональності та забезпечити можливість створення нових типів ворогів без суттєвої зміни вже існуючої системи. Окрім руху, кожен противник використовує компоненти EnemyHealth та EnemyAttack, які відповідають відповідно за обробку отриманого пошкодження та виконання атакуючих дій. Параметри здоров'я, сили атаки та інших характеристик зберігаються у структурі EnemyData, що дозволяє централізовано налаштовувати баланс гри та швидко змінювати характеристики окремих ворогів без модифікації програмного коду.

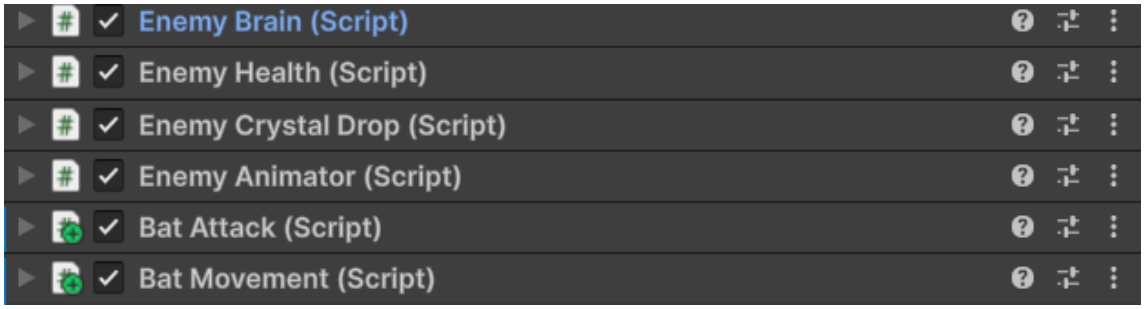

Рисунок 3.2 - Компоненти ворога в інспекторі

Для підвищення візуальної якості ігрового процесу система штучного інтелекту інтегрована з підсистемою анімації, приклад аніматора зображений на рисунку 3.4. Після зміни стану противника відповідна інформація передається до компонента EnemyAnimation, який керує переходами між анімаціями очікування, руху, отримання пошкодження та атаки. Додатково система штучного інтелекту взаємодіє з бойовою системою та механізмом масштабування складності, внаслідок чого характеристики ворогів можуть автоматично змінюватися в процесі проходження гри. Реалізований підхід дозволив створити гнучку та масштабовану систему поведінки противників, яка забезпечує достатню різноманітність ігрового процесу та може бути використана як основа для подальшого розширення функціональності проєкту.

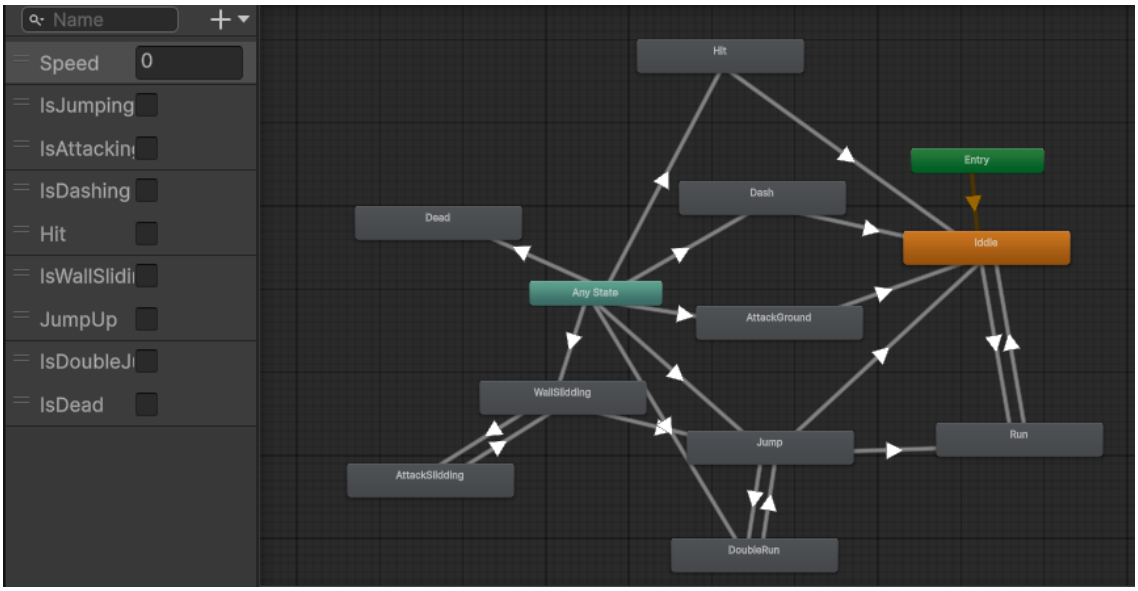

Рисунок 3.4 - Машина станів анімації противника

3.4. Процедурна генерація рівнів

Однією з ключових особливостей жанру Roguelite є висока реіграбельність, яка досягається шляхом зміни умов проходження між окремими ігровими сесіями [14], Для реалізації цього принципу в грі «Etemenamki» було розроблено систему процедурного формування рівнів на основі заготовлених шаблонів кімнат. На відміну від повністю випадкової генерації геометрії рівня, використаний підхід передбачає формування карти шляхом вибору та комбінування готових кімнат із попередньо створеного набору. Управління процесом генерації виконує компонент RoomManager, який відповідає за вибір кімнат, їх розміщення та побудову структури рівня. Кожна кімната містить інформацію про власні межі, внутрішнє наповнення та можливі точки переходу до інших кімнат.

Для забезпечення різноманітності проходжень кожна кімната містить власний набір ігрових об'єктів, включаючи противників, декоративні елементи та точки взаємодії. Під час генерації рівня система не лише визначає порядок розташування кімнат, а й формує загальну структуру проходження, що впливає на складність та темп ігрового процесу. Такий підхід дозволяє поєднати переваги випадкової генерації з контрольованим дизайном рівнів, зберігаючи баланс між непередбачуваністю та зручністю проходження для гравця.

Використання префабів кімнат значно спрощує процес розробки та подальшого розширення гри. Розробнику достатньо створити новий шаблон кімнати та додати його до набору доступних модулів, після чого він автоматично може брати участь у генерації нових рівнів. Приклад набору префабів, що використовуються системою генерації, наведено на рисунку 3.5. Така організація ресурсів підвищує масштабованість проєкту та спрощує внесення змін до структури ігрового світу без модифікації основного програмного коду.

Запропонований механізм процедурного формування рівнів добре відповідає особливостям жанру Roguelite, оскільки забезпечує високу реіграбельність та створює унікальні умови для кожного нового проходження. У подальшому система може бути розширена шляхом додавання спеціальних кімнат,

подієвих зон або унікальних випробувань, що дозволить ще більше урізноманітнити ігровий процес та підвищити зацікавленість користувачів.

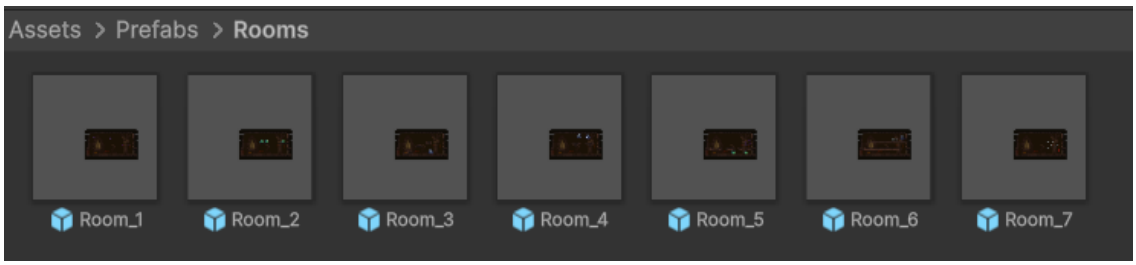

Рисунок 3.5 - Префаби створених кімнат

Після завершення формування карти між окремими кімнатами створюються спеціальні телепорти, які забезпечують переміщення персонажа між частинами рівня. Такий підхід дозволив спростити навігацію та уникнути необхідності реалізації складної системи коридорів або переходів між локаціями. Використання шаблонних кімнат також позитивно вплинуло на швидкість розробки, оскільки дало можливість контролювати якість рівнів та уникнути появи некоректних конфігурацій. Незважаючи на використання готових модулів, випадковий порядок їхнього розташування забезпечує достатню різноманітність проходжень і підтримує характерну для жанру непередбачуваність.

3.5.Система масштабування складності

Для підтримання інтересу гравця протягом усієї ігрової сесії в проєкті реалізовано систему поступового масштабування складності. Її основою є компонент DifficultyTimer, який безперервно відстежує тривалість поточного проходження. Візуально робота системи відображається за допомогою спеціального індикатора у користувацькому інтерфейсі, що дозволяє гравцю контролювати наближення наступного етапу підвищення складності. Після завершення чергового циклу таймера значення рівня складності автоматично збільшується.

Підвищення складності безпосередньо впливає на характеристики

учасників бою. Зі збільшенням рівня складності зростають показники здоров'я та сили атаки противників, що робить виживання більш складним і вимагає від гравця ефективнішого використання доступних ресурсів та покращень. Одночасно система також коригує параметри персонажа, зокрема величину нанесеного пошкодження, що дозволяє зберігати баланс між розвитком героя та силою ворогів. Такий підхід забезпечує поступове ускладнення проходження без різких стрибків складності та підтримує динаміку ігрового процесу протягом тривалих сесій.

Реалізована система масштабування складності базується на поступовому збільшенні рівня складності через визначені проміжки часу. Такий підхід відповідає концепції жанру Roguelite, де гравець повинен постійно адаптуватися до нових умов проходження та ефективно використовувати доступні ресурси. Зі збільшенням рівня складності зростають характеристики противників, що робить кожне наступне зіткнення більш небезпечним та стимулює користувача вдосконалювати власну стратегію гри. Водночас система враховує розвиток персонажа, забезпечуючи баланс між зростанням сили ворогів і можливостями гравця.

Для реалізації механізму масштабування використовується спеціальний множник, значення якого залежить від поточного рівня складності. Зі збільшенням цього параметра змінюються коефіцієнти нанесеного та отриманого пошкодження, що безпосередньо впливає на динаміку бойової системи.

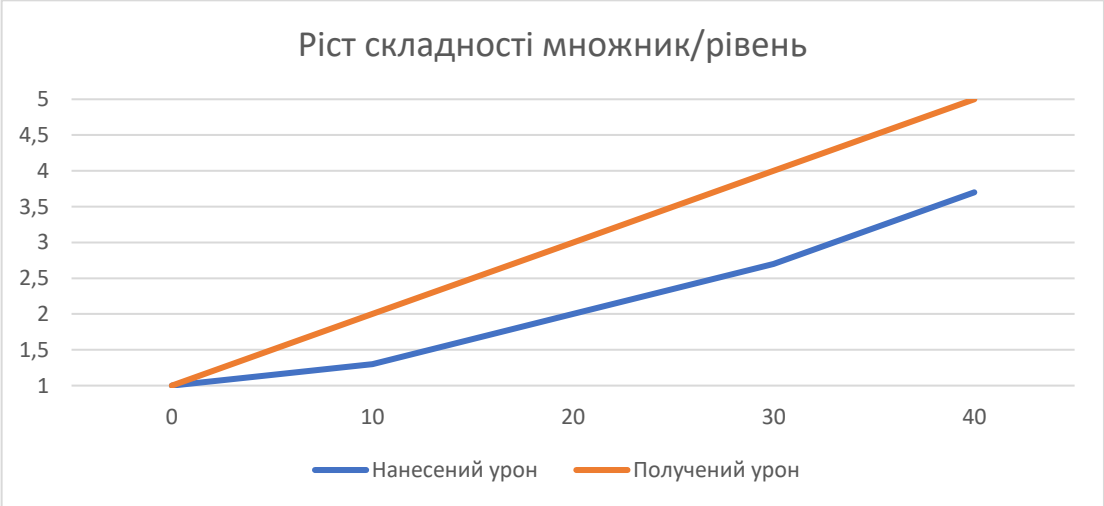

Рисунок 3.6 - Діаграма росту складності

Графічне представлення зміни відповідних параметрів наведено на рисунку 3.6. Як видно з діаграми, зі зростанням рівня складності швидкість підвищення отриманого пошкодження є вищою, ніж приріст ефективності персонажа, що підтримує необхідний рівень виклику та запобігає надмірному спрощенню ігрового процесу.

Важливою перевагою запропонованої системи є її гнучкість та можливість подальшого налаштування. Значення коефіцієнтів масштабування можуть змінюватися залежно від особливостей балансу гри, кількості доступних покращень або типів противників. Завдяки такому підходу розробник отримує можливість адаптувати складність до потреб різних категорій гравців, а також розширювати функціональність системи без внесення суттєвих змін до загальної архітектури програмного забезпечення.

3.6. Система прогресії персонажа

Система розвитку персонажа в грі побудована навколо механіки збору кристалів, які виступають основним ресурсом для покращення характеристик героя. Після знищення противників на рівні можуть випадати спеціальні предмети-пікапи різних типів. Під час контакту персонажа з таким об'єктом відповідний кристал автоматично додається до інвентаря. Збір ресурсів є одним із ключових елементів ігрового циклу, оскільки саме від кількості та якості отриманих кристалів залежить подальший розвиток персонажа та його ефективність у бою.

Для спрощення налаштування та балансування всі параметри кристалів винесені до окремих об'єктів даних. Кожен кристал містить інформацію про власний тип, набір бонусів та величину впливу на характеристики героя. Кристали поділяються за кольорами, причому кожен колір відповідає окремому напрямку розвитку персонажа. Усередині кожної кольорової групи також існує поділ за розміром. Більші кристали використовують ту саму гілку покращень, однак забезпечують значно потужніший ефект порівняно з меншими аналогами.

Для організації отриманих ресурсів у грі реалізовано систему слотів

інвентаря. Інтерфейс дозволяє відображати наявні кристали та розподіляти їх між відповідними комірками. Взаємодія між інвентарем, слотами та системою екіпірування реалізована через компоненти CrystalInventory, EquipmentManager, InventoryUI та CrystalSlotUI, які забезпечують збереження даних та їхнє коректне відображення в інтерфейсі.

Рисунок 3.7 - Слоти для кристалів

Установлені кристали безпосередньо впливають на характеристики персонажа через систему PlayerStatSystem. Після розміщення кристала у відповідному слоті, що показані на рисунку 3.7, його бонуси автоматично враховуються під час розрахунку підсумкових параметрів героя. Завдяки такому підходу гравець отримує можливість самостійно формувати стиль гри та адаптувати персонажа до поточних умов проходження, що є важливою складовою жанру Roguelite.

3.7. Висновки по розділу

У третьому розділі було розглянуто процес програмної реалізації основних підсистем комп'ютерної гри «Etemenamki». У ході розробки реалізовано систему керування персонажем, яка забезпечує обробку користувацького вводу, пересування, виконання стрибків та взаємодію з фізичним середовищем. Завдяки

використанню компонентного підходу Unity та інтеграції з анімаційною системою вдалося створити плавне й інтуїтивно зрозуміле керування, що відповідає вимогам сучасних двовимірних ігор жанру Roguelite.

Важливим результатом роботи стала реалізація бойової системи, яка поєднує механіки дистанційних і ближніх атак. Використання проєктилів, системи виявлення області ураження та механізмів нанесення і отримання пошкоджень дозволило створити гнучку основу для взаємодії між персонажем і противниками.

Окрему увагу приділено створенню механізмів, які формують характерні особливості жанру Roguelite. Для цього реалізовано систему процедурного формування рівнів на основі шаблонних кімнат, що дозволяє створювати різноманітні конфігурації ігрового світу при кожному новому проходженні. Додатково впроваджено систему масштабування складності, яка поступово змінює характеристики персонажа та противників залежно від тривалості ігрової сесії.

Проведена програмна реалізація підтвердила ефективність обраних технологій та архітектурних рішень, а також можливість застосування ігрового рушія Unity для створення повноцінних програмних продуктів жанру Roguelite. Реалізована архітектура дозволяє без суттєвих змін доповнювати проєкт новими типами ворогів, механіками розвитку персонажа, рівнями та іншими елементами ігрового процесу, що створює передумови для подальшого розвитку та вдосконалення гри.

Завершальним елементом програмної реалізації стала система прогресії персонажа, побудована навколо механіки збору та використання кристалів. Реалізовано структуру зберігання даних про кристали, систему інвентаря, слоти екіпірування та механізм автоматичного перерахунку характеристик героя залежно від отриманих покращень. У результаті було створено цілісний комплекс взаємопов’язаних підсистем, які забезпечують стабільне функціонування гри та формують основу для її подальшого розвитку. Реалізована архітектура дозволяє без суттєвих змін доповнювати проєкт новими типами ворогів, механіками розвитку персонажа, рівнями та іншими елементами ігрового процесу.

РОЗДІЛ 4. ПРОЕКТУВАННЯ ТА АРХІТЕКТУРА ГРИ

4.1. Архітектура програмного забезпечення гри

Під час розробки комп'ютерної гри «Etemenamki» було використано компонентно-орієнтований підхід до проєктування програмного забезпечення, який є одним із ключових принципів роботи ігрового рушія Unity. Даний підхід передбачає поділ функціональності програми на окремі незалежні компоненти, кожен з яких відповідає за виконання конкретного набору завдань. Це дозволяє забезпечити високу гнучкість системи, спростити підтримку програмного коду та полегшити процес подальшого розширення функціональності гри. Використання компонентної архітектури особливо актуальне для ігрових застосунків, оскільки вони складаються з великої кількості взаємопов'язаних механік, які повинні працювати узгоджено та стабільно.

Архітектура програмного забезпечення гри побудована на взаємодії ігрових об'єктів та компонентів типу MonoBehaviour. Кожний об'єкт сцени містить набір скриптів, що реалізують окремі аспекти його поведінки та забезпечують взаємодію з іншими елементами ігрового середовища. Наприклад, об'єкт головного героя поєднує компоненти керування рухом, анімацією, бойовою системою та характеристиками персонажа. Аналогічний підхід використовується і для ворогів, які містять компоненти штучного інтелекту, пересування, обробки пошкоджень та анімацій. Така організація дозволяє повторно використовувати програмний код та спрощує створення нових ігрових об'єктів на основі вже реалізованих модулів. Структура проєкту організована за функціональним принципом і розділена на окремі директорії, серед яких можна виділити Scripts/Core, Scripts/Enemy, Scripts/Player, Scripts/Inventory та Scripts/UI. Папка Core містить базові класи, які відповідають за генерацію рівнів та взаємодію між кімнатами. У ній розташовані класи Room, RoomManager та TeleportPortal, що забезпечують формування структури рівня та переміщення гравця між окремими його частинами. Подібний поділ дозволяє локалізувати функціональність окремих підсистем та полегшує

навігацію програмним кодом під час розробки.

Важливою перевагою використаної архітектури є можливість незалежної розробки окремих підсистем гри. Наприклад, зміни в логіці роботи противників не потребують модифікації системи керування персонажем або генерації рівнів. Це значно спрощує процес тестування програмного забезпечення та зменшує ймовірність виникнення помилок під час внесення змін. Крім того, компонентний підхід забезпечує повторне використання вже реалізованих програмних модулів, що скорочує час розробки нових ігрових механік та підвищує якість програмного коду.

Основні переваги обраної архітектури:

1. модульність програмного забезпечення;
2. можливість повторного використання компонентів;
3. спрощення тестування окремих підсистем;
4. висока масштабованість проєкту;
5. зручність підтримки та оновлення гри.

Застосування таких принципів проєктування є особливо важливим для ігрових застосунків жанру Roguelite, оскільки вони постійно розширюються новими механіками, противниками та елементами прогресії. Гнучка архітектура дозволяє адаптувати програмний продукт до майбутніх змін без необхідності повного перепроєктування системи.

Окрема увага приділена реалізації користувацького інтерфейсу. Для цього використовуються компоненти InventoryUI, CrystalSlotUI та DifficultyTimer. Інтерфейс забезпечує відображення стану персонажа, інвентаря та поточного рівня складності. Відкриття інвентаря здійснюється за допомогою клавіші Tab, а всі зміни характеристик миттєво відображаються у відповідних елементах інтерфейсу. Для реалізації UI використовувалися стандартні засоби Unity UI та бібліотека TextMeshPro, що забезпечило якісне відображення текстової інформації та масштабованість елементів на різних роздільних здатностях екрана.. Для координації роботи окремих підсистем застосовуються спеціалізовані менеджери та механізми обміну даними між компонентами. Подібна архітектура забезпечує

слабке зв’язування між модулями та зменшує залежність окремих частин програми одна від одної. Завдяки цьому стає можливим подальше розширення функціональності гри шляхом додавання нових типів ворогів, кімнат, предметів або механік без суттєвого перепроєктування існуючої системи.

4.2. Взаємодія компонентів

Ефективність функціонування комп’ютерної гри значною мірою залежить від правильно організованої взаємодії між окремими програмними підсистемами. Під час розробки гри «Etemenamki» було реалізовано модульний підхід, за якого кожна підсистема відповідає за виконання власного набору функцій та взаємодіє з іншими компонентами через спільні дані та виклики методів. Такий підхід дозволяє зменшити залежність між окремими частинами програми та спростити подальший розвиток проєкту. При побудові архітектури використовувалися принципи проєктування ігрових систем [11].

Центральним елементом ігрового процесу є персонаж користувача. Його робота забезпечується сукупністю компонентів, які відповідають за обробку введення, переміщення, виконання атак та керування характеристиками. Введення з клавіатури та миші обробляється системою керування персонажем, яка забезпечує рух за допомогою клавіш A та D, стрибок клавішею Space, а також використання дистанційної та ближньої атак. Усі зміни стану персонажа передаються до інших підсистем, зокрема системи анімації та бойової системи.

Бойова система тісно пов’язана із системою характеристик персонажа та механізмом масштабування складності. Під час виконання атак обчислюється величина пошкодження, яка залежить від поточних параметрів героя та активних покращень. Після нанесення пошкодження інформація передається системі здоров’я противника, яка визначає подальший стан ворога. У випадку знищення противника активується механізм випадіння кристалів, що створює взаємозв’язок між бойовою системою та системою прогресії персонажа. Важливу роль у роботі гри відіграє система штучного інтелекту противників. Кожний ворог використовує

власний набір компонентів для аналізу положення гравця та вибору подальших дій. Загальна логіка поведінки реалізована через компоненти EnemyBrain та EnemyState, тоді як безпосередній рух виконується за допомогою EnemyMove та його похідних реалізацій. Завдяки такому підходу різні типи противників можуть використовувати власні алгоритми переміщення та атаки, зберігаючи спільну базову архітектуру.

начна частина взаємодії між підсистемами відбувається в режимі реального часу. Наприклад, зміна характеристик персонажа внаслідок встановлення кристалів миттєво впливає на бойову систему, величину пошкодження та відображення відповідних значень у користувацькому інтерфейсі. Аналогічно підвищення рівня складності автоматично змінює параметри противників та коригує баланс гри без необхідності ручного втручання користувача. Така організація взаємодії забезпечує цілісність ігрового процесу та узгодженість роботи всіх компонентів системи.

У процесі виконання гри між підсистемами передаються такі типи даних:

1. характеристики персонажа;
2. стан здоров'я противників;
3. інформація про рівень складності;
4. дані інвентаря та кристалів;
5. координати ігрових об'єктів;
6. стан кімнат та телепортів.

Передача даних між компонентами реалізована через публічні методи, змінні та механізми взаємодії компонентів Unity. Такий підхід дозволяє забезпечити стабільну роботу програми навіть за великої кількості одночасно активних об’єктів на сцені та сприяє підтриманню високої продуктивності гри.

Система генерації рівнів взаємодіє з більшістю інших компонентів гри. Під час створення нового проходження компонент RoomManager формує структуру рівня, вибираючи кімнати з попередньо створеного набору префабів. Після завершення генерації в кімнатах розміщуються вороги та інші інтерактивні об’єкти. Подібний механізм дозволяє забезпечити різноманітність проходжень без необхідності ручного створення кожного рівня.

Для переміщення між окремими кімнатами використовуються телепорти, які активуються після виконання визначених умов. Такий спосіб організації переходів дозволяє спростити навігацію ігровим світом та зменшити складність реалізації процедурно сформованих рівнів. Крім того, використання телепортів дає змогу легко розширювати структуру карти шляхом додавання нових типів кімнат та спеціальних зон. Окреме місце в архітектурі займає система прогресії персонажа. Після перемоги над противниками гравець отримує кристали різних типів та розмірів, які можуть бути встановлені у відповідні слоти інвентаря. Кожен кристал містить набір характеристик, що впливають на параметри персонажа. Після зміни спорядження система автоматично перераховує характеристики героя та застосовує відповідні бонуси до його параметрів.

Для забезпечення зручності взаємодії з грою реалізовано окрему підсистему користувацького інтерфейсу. Вона відповідає за відображення поточного стану персонажа, рівня складності, інвентаря та інших ігрових параметрів. Відкриття інвентаря здійснюється за допомогою клавіші Tab, а всі зміни характеристик миттєво відображаються на екрані. Використання компонентів Unity UI та TextMeshPro забезпечує якісне відображення графічних елементів та текстової інформації.

4.3. Організація даних та ігрових систем

Під час розробки гри «Etemenamki» особлива увага приділялася організації даних та структурі ігрових ресурсів. Правильне зберігання та обробка інформації є важливою складовою будь-якого програмного продукту, оскільки безпосередньо впливає на продуктивність системи, зручність розробки та можливість подальшого розширення функціональності. У середовищі Unity для цього використовуються сцени, префаби, спрайти, анімації та програмні компоненти, які утворюють єдину структуру проєкту.

Основною сценою гри є ігровий рівень, у межах якого відбувається генерація кімнат, розміщення ворогів та взаємодія гравця з навколишнім середовищем. Крім ігрової сцени, у проєкті також присутні сцени головного меню

та завершення гри. Поділ функціональності між окремими сценами дозволяє зменшити навантаження на систему та спростити керування різними етапами ігрового процесу. Кожна сцена містить власний набір об’єктів та компонентів, необхідних для виконання конкретних завдань. Для повторного використання ігрових об’єктів у проєкті активно застосовуються префаби Unity. За допомогою префабів реалізовано кімнати рівнів, противників, кристали, телепорти та окремі елементи інтерфейсу. Використання такого підходу дозволяє створювати нові екземпляри об’єктів під час виконання програми без необхідності повторного налаштування їх параметрів. Це особливо важливо для процедурної генерації рівнів, де велика кількість об’єктів створюється динамічно під час кожного нового проходження.

Система кристалів базується на використанні структур даних, які зберігають інформацію про тип кристала, його розмір та набір характеристик. Кожен кристал впливає на певні параметри персонажа та може бути встановлений у відповідний слот інвентаря. Подібна організація даних спрощує балансування гри та дозволяє легко додавати нові види покращень без суттєвих змін існуючої логіки програмного забезпечення.

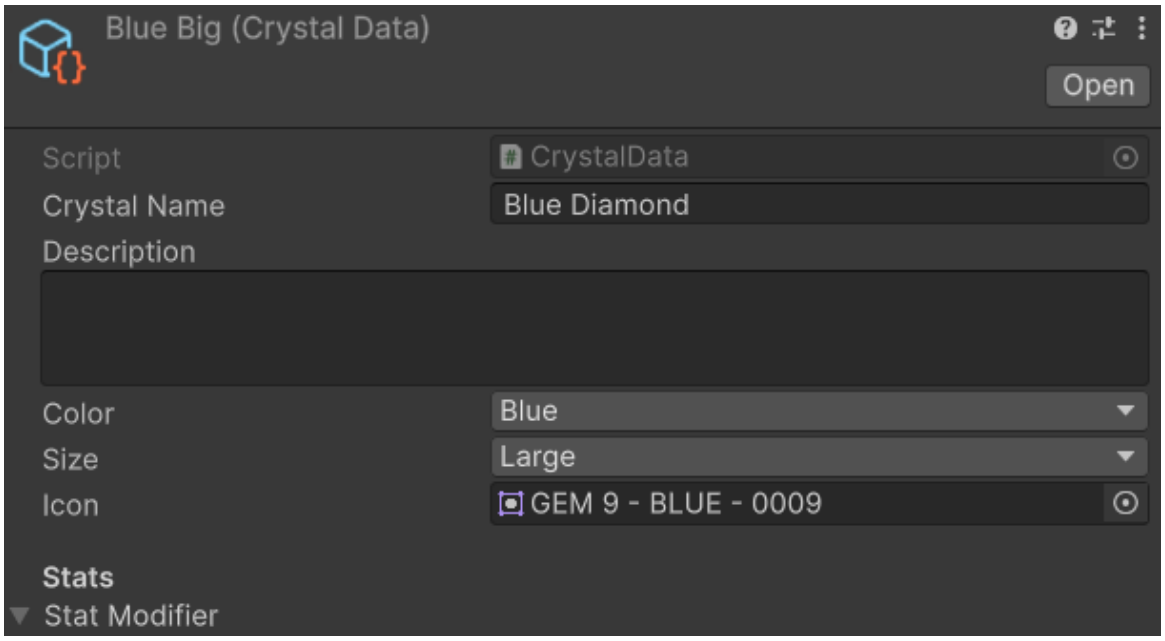

Рисунок 4.1 - Дата кристалу

Основні категорії ігрових ресурсів проєкту:

- спрайти персонажа та ворогів;
- анімації та Animator Controller;
- тайлмапи кімнат;
- префаби рівнів;
- звукові ефекти;
- UI-елементи інтерфейсу;
- ScriptableObject для кристалів;
- програмні скрипти C#.

Для зручності балансування гри більшість параметрів об'єктів винесено в окремі налаштування, які можуть змінюватися без модифікації основної логіки програми, що зберігаються в окремих датах для кожного виду кристалів, рисунок 4.1. Такий підхід спрощує налаштування характеристик противників, кристалів та персонажа, а також дозволяє швидко адаптувати баланс гри під час тестування. Зміна окремих параметрів не потребує переписування програмного коду, що значно пришвидшує процес розробки.

Окрему роль відіграє система зберігання характеристик персонажа. До основних параметрів належать запас здоров’я, величина пошкодження, швидкість пересування та інші характеристики, що змінюються в процесі гри. Значення параметрів динамічно перераховуються після отримання нових покращень або зміни рівня складності. Завдяки цьому забезпечується коректна взаємодія між системою прогресії, бойовою системою та механізмом масштабування складності.

Графічні ресурси гри представлені двовимірними спрайтами персонажа, противників, елементів оточення та користувацького інтерфейсу. Для відтворення анімацій використовується система Animator, яка дозволяє плавно змінювати стани об’єктів залежно від їхньої поведінки. Використання окремих анімаційних контролерів для персонажа та противників забезпечує гнучке керування анімаціями та спрощує подальше розширення візуальної складової гри.

Підсумовуючи, організація даних та ігрових ресурсів у проєкті «Etemenamki» побудована відповідно до принципів модульності та повторного

використання компонентів. Застосування сцен, префабів та структурованих даних дозволило створити гнучку архітектуру, яка забезпечує стабільну роботу гри та створює основу для її подальшого розвитку.

Централізоване зберігання ресурсів та чітка структура директорій проєкту полегшують навігацію між файлами та сприяють підтримці великого обсягу контенту. Це особливо важливо для проєктів жанру Roguelite, які передбачають регулярне додавання нових механік, предметів та ігрових елементів.

4.4. Оптимізація та продуктивність гри

Під час розробки гри «Etemenamki» значна увага приділялася питанням оптимізації та забезпечення стабільної роботи програмного забезпечення. На відміну від лінійних ігор із заздалегідь визначеним вмістом, проєкти жанру Roguelite постійно створюють нові комбінації кімнат, противників та ігрових подій. Це призводить до динамічної зміни навантаження на систему та вимагає використання ефективних механізмів керування ресурсами. Саме тому ще на етапі проєктування архітектури враховувалися питання продуктивності, масштабованості та можливості подальшого розширення функціоналу гри.

Одним із найважливіших аспектів оптимізації стало використання процедурної генерації рівнів. У грі не створюється повна карта заздалегідь — нові кімнати формуються в процесі проходження на основі підготовленого набору префабів. Завдяки цьому зменшується обсяг даних, які одночасно знаходяться в оперативній пам’яті, а кожне проходження залишається унікальним для користувача.

Для підтримання стабільної продуктивності в проєкті застосовувалися такі підходи:

- використання префабів для повторного використання ігрових об’єктів;
- автоматичне видалення або деактивація неактивних кімнат;
- розподіл логіки між окремими програмними компонентами;
- обмеження кількості активних об’єктів на сцені;

• повторне використання вже створених ресурсів Unity.

Використання перелічених рішень дозволило знизити навантаження на процесор та оперативну пам’ять, а також спростило подальшу підтримку програмного коду.

Важливу роль у забезпеченні продуктивності відіграє компонентна архітектура Unity. Кожний ігровий об’єкт складається з набору незалежних компонентів, які відповідають лише за власний функціонал. Наприклад, логіка пересування персонажа, система анімації та механізм нанесення пошкоджень реалізовані окремими скриптами. Такий підхід не лише покращує читабельність коду, але й полегшує його тестування та повторне використання в інших частинах проєкту.

Окремої уваги потребує оптимізація процедурної генерації рівнів. Оскільки структура ігрового світу формується динамічно, важливо забезпечити коректне створення та видалення кімнат без виникнення помилок або надмірного споживання пам’яті. У реалізованій системі кімнати завантажуються лише за необхідності, а попередні частини рівня можуть бути видалені після втрати актуальності. Це дозволяє підтримувати стабільну продуктивність навіть під час тривалих ігрових сесій.

Під час створення штучного інтелекту ворогів також враховувалися питання ефективності обчислень. Обробка поведінки противників виконується лише тоді, коли вони знаходяться в активній частині рівня та можуть взаємодіяти з гравцем. Такий підхід зменшує кількість непотрібних розрахунків і позитивно впливає на швидкодію програми.

Результати проведеного тестування показали, що розроблений програмний продукт забезпечує стабільну роботу на цільовій платформі Windows та підтримує комфортний рівень продуктивності під час виконання основних ігрових сценаріїв. Запропоновані архітектурні рішення створюють основу для подальшого розвитку гри, включаючи додавання нових типів ворогів, кімнат та ігрових механік без суттєвого зниження швидкодії системи.

4.5. Висновки по розділу

У даному розділі було розглянуто особливості проєктування та архітектури програмного забезпечення гри «Etemenamki». Проведений аналіз показав, що використання компонентно-орієнтованого підходу та засобів рушія Unity дозволяє створювати гнучкі та масштабовані ігрові системи. Побудована архітектура забезпечує чіткий розподіл функціональності між окремими модулями та спрощує процес підтримки програмного продукту.

У ході дослідження було описано структуру програмного забезпечення гри та визначено основні підсистеми, які забезпечують її функціонування. До таких підсистем належать система керування персонажем, штучний інтелект ворогів, генерація рівнів, система прогресії персонажа та користувацький інтерфейс. Організація взаємодії між цими компонентами дозволяє забезпечити стабільне виконання ігрової логіки та коректний обмін даними між окремими модулями. Увагу було приділено організації даних та ігрових ресурсів. Використання сцен, префабів та структурованих програмних компонентів дозволило

Розроблена архітектура забезпечує можливість подальшого розширення функціональних можливостей гри без необхідності суттєвого перепроєктування вже реалізованих компонентів. Завдяки модульній структурі до проєкту можуть бути додані нові типи ворогів, кімнат, предметів та ігрових механік із мінімальним впливом на існуючу систему. Це підвищує масштабованість програмного забезпечення та спрощує його супровід у майбутньому.

Результати проєктування підтверджують доцільність використання обраних архітектурних рішень та програмних засобів для розробки гри «Etemenamki». Реалізована структура програмного забезпечення відповідає сучасним принципам розробки ігрових застосунків, забезпечує стабільність роботи системи та створює основу для подальшого розвитку й удосконалення програмного продукту.

РОЗДІЛ 5. ОПИС ТА ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

5.1. Інтерфейс користувача

Інтерфейс користувача є важливою складовою будь-якого програмного продукту, оскільки саме через нього відбувається взаємодія користувача з системою. Під час розробки гри «Etemenamki» основною метою було створення простого та зрозумілого інтерфейсу, який надає гравцеві всю необхідну інформацію без перевантаження ігрового екрана другорядними елементами.

Головне меню гри забезпечує доступ до основних функцій програми, зокрема запуску гри та завершення роботи застосунку. Після початку проходження користувач потрапляє на ігровий рівень, де основні елементи інтерфейсу розташовані таким чином, щоб не заважати огляду ігрового світу. У верхній частині екрана відображаються характеристики персонажа та індикатор рівня складності. Окремий елемент інтерфейсу використовується для візуалізації роботи таймера складності, який демонструє наближення чергового етапу збільшення складності гри.

Головне меню гри виконує не лише навігаційну функцію, але й формує перше враження користувача про програмний продукт. Під час його розробки особлива увага приділялася відповідності візуального оформлення загальній стилістиці гри та забезпеченню інтуїтивної взаємодії з основними елементами керування. Як показано на рисунку 5.1, інтерфейс меню містить лише необхідні елементи та не перевантажує користувача зайвою інформацією, що сприяє швидкому переходу до ігрового процесу.

Для керування системою прогресії реалізовано окреме вікно інвентаря, яке відкривається за натисканням клавіші Tab. У вікні відображаються всі доступні кристали та слоти для їх встановлення. Інтерфейс дозволяє швидко оцінити поточний стан розвитку персонажа та виконати необхідні зміни без переривання ігрового процесу на тривалий час. Для реалізації елементів інтерфейсу використовувалися стандартні засоби Unity UI та TextMeshPro, що забезпечило коректне масштабування елементів і якісне відображення текстової інформації.

Рисунок 5.1 - Меню гри

Важливою характеристикою інтерфейсу є його адаптивність до різних ігрових ситуацій. Під час активних бойових сцен користувач повинен швидко отримувати інформацію про поточний стан персонажа, не відволікаючись від керування. Саме тому найбільш важливі показники, такі як запас здоров’я, рівень складності та доступні ресурси, відображаються постійно. Менш критичні елементи, зокрема інвентар кристалів, відкриваються лише за потреби, що дозволяє зменшити навантаження на екран та покращити сприйняття інформації.

Під час проєктування інтерфейсу враховувалися основні принципи дизайну користувацьких систем: читабельність, візуальна ієрархія та узгодженість елементів. Для текстових компонентів використовувалися шрифти TextMeshPro, які забезпечують високу якість відображення незалежно від роздільної здатності екрана. Додатково було реалізовано систему оновлення інтерфейсу в реальному часі, завдяки якій зміни характеристик персонажа миттєво відображаються для користувача.

Основні елементи інтерфейсу гри:

- головне меню;
- індикатор здоров’я персонажа;
- лічильник кристалів;

- індикатор рівня складності;
- таймер складності;
- вікно інвентаря;
- слоти для встановлення кристалів.

Завдяки поєднанню функціональності та мінімалістичного дизайну інтерфейс не перевантажує користувача та забезпечує комфортну взаємодію з програмним продуктом протягом тривалих ігрових сесій.

Важливим аспектом розробки інтерфейсу є забезпечення його зрозумілості та зручності використання під час динамічного ігрового процесу. Саме тому інформаційні елементи були розміщені у верхній частині екрана та згруповані за функціональним призначенням. Такий підхід дозволяє гравцеві швидко оцінювати поточний стан персонажа, контролювати рівень складності та взаємодіяти з інвентарем без необхідності відволікатися від основних ігрових подій.

У процесі розробки особлива увага приділялася візуальній узгодженості елементів інтерфейсу з загальним художнім стилем гри. Завдяки цьому інтерфейс сприймається як невід'ємна частина ігрового середовища та не створює дискомфорту під час проходження.

5.2. Ігровий процес

Ігровий процес «Etemenamki» побудований навколо циклу дослідження рівнів, боротьби з противниками та розвитку персонажа. Після початку гри користувач отримує контроль над головним героєм і може переміщуватися між кімнатами рівня, використовуючи систему телепортів. Кожне нове проходження формує іншу конфігурацію рівня, що забезпечує різноманітність ігрових ситуацій та підвищує реіграбельність проєкту [15].

Під час дослідження рівнів гравець зустрічає різні типи ворогів, які використовують власні моделі поведінки та характеристики. Для боротьби з ними доступні дистанційні атаки за допомогою снарядів та ближній бій із використанням меча. Успішне знищення противників дозволяє отримувати кристали різних типів

і розмірів, які використовуються для покращення характеристик персонажа.

У міру збільшення тривалості ігрової сесії активується система масштабування складності. Рівень складності поступово зростає, що призводить до збільшення характеристик ворогів та зміни балансу бойової системи. Це стимулює користувача ефективно використовувати отримані покращення та адаптувати власну тактику до нових умов проходження.

Важливою особливістю ігрового процесу є поєднання дослідження світу та постійної бойової взаємодії. Гравець повинен не лише знищувати противників, а й ефективно керувати наявними ресурсами та обирати оптимальну стратегію розвитку персонажа. Завдяки випадковому розташуванню кімнат і різним комбінаціям ворогів кожне нове проходження створює унікальні ігрові ситуації та вимагає адаптації до нових умов.

Характерною особливістю гри є поєднання елементів дослідження та виживання. Користувач постійно змушений оцінювати поточний стан персонажа, обирати оптимальний маршрут проходження та адаптувати власну стратегію до нових умов. Через процедурне формування рівнів гравець не може повністю передбачити майбутню конфігурацію кімнат або розташування противників, що робить кожне проходження унікальним.

Важливу роль у формуванні ігрового досвіду відіграє система поступового збільшення складності. Зі зростанням рівня складності підвищуються характеристики противників, що стимулює користувача ефективніше використовувати отримані покращення та комбінувати різні типи кристалів.

Основні етапи ігрового циклу можна подати у такому вигляді:

1. Дослідження нової кімнати.
2. Виявлення та знищення противників.
3. Отримання кристалів та ресурсів.
4. Покращення характеристик персонажа.
5. Перехід до наступної кімнати.
6. Подальше збільшення складності гри.

Окрему увагу приділено створенню відчуття прогресу під час проходження.

Навіть після поразки користувач отримує досвід взаємодії з механіками гри та може застосувати набуті знання під час наступних спроб. Такий підхід відповідає основним принципам жанру Roguelite та підтримує зацікавленість гравця протягом тривалого часу.

Під час проходження рівнів користувач взаємодіє з різними елементами ігрового середовища, зокрема телепортами між кімнатами та предметами для покращення характеристик персонажа. Приклад ігрової кімнати з розташованими противниками наведено на рисунку 5.2. Використання попередньо створених шаблонів кімнат дозволяє поєднати контрольоване проєктування рівнів із механіками процедурної генерації, характерними для жанру Roguelite.

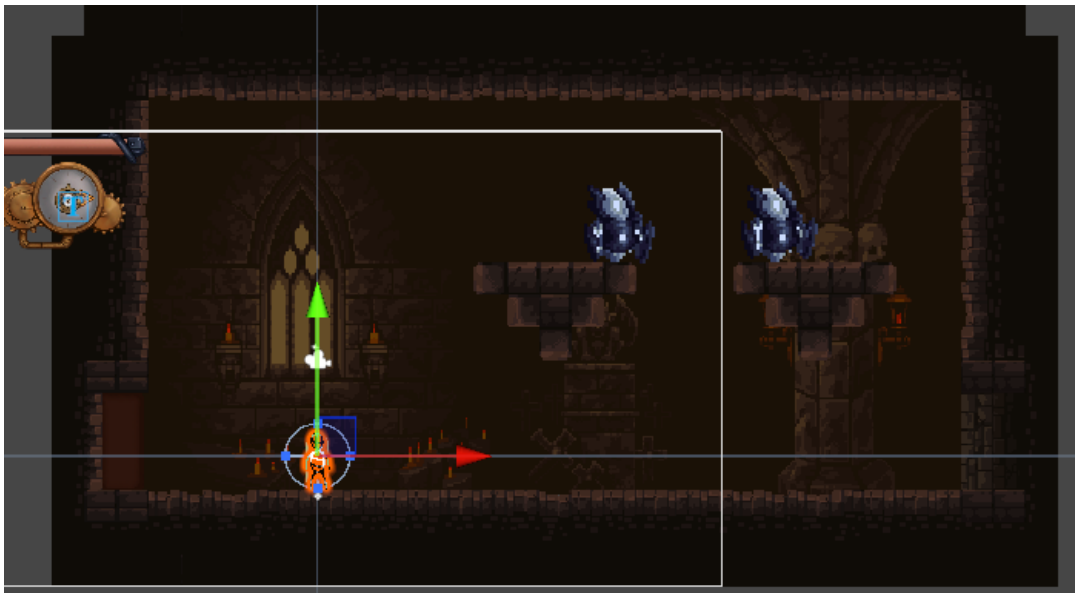

Рисунок 5.2 - Вигляд префабу кімнати з ворогами

Окрему роль у формуванні ігрового процесу відіграє система прогресії персонажа, яка базується на використанні кристалів різних типів. Отримані під час проходження покращення безпосередньо впливають на характеристики героя та відкривають можливість формування різних стилів гри. Такий підхід підвищує реіграбельність проєкту та мотивує користувача експериментувати з різними варіантами розвитку персонажа.

Таким чином, основний цикл гри складається з дослідження рівня, боротьби

з противниками, збору ресурсів, покращення персонажа та подальшого виживання в умовах постійно зростаючої складності.

5.3. Тестування функціональності

Після завершення реалізації основних підсистем було проведено тестування програмного продукту. Метою тестування стала перевірка коректності роботи реалізованих механік, стабільності функціонування гри та відповідності отриманих результатів поставленим вимогам.

Під час тестування перевірялася робота системи керування персонажем, бойової системи, механізму генерації рівнів, системи прогресії та масштабування складності. Особлива увага приділялася взаємодії між окремими підсистемами, оскільки помилки найчастіше виникають саме під час обміну даними між компонентами програмного забезпечення.

Тестування виконувалося шляхом багаторазового проходження гри з різними сценаріями використання. Отримані результати таблиця 5.1 підтвердили коректність роботи основних механік та відсутність критичних помилок, що перешкоджають проходженню гри.

Таблиця 5.1

Результати тестування функціональності

Функція	**Очікуваний результат**	**Фактичний результат**
Рух персонажа	Коректне переміщення та стрибок	Виконується коректно
Стрільба	Створення та рух проєктиля	Виконується коректно
Атака мечем	Нанесення пошкодження в зоні удару	Виконується коректно
Генерація рівня	Створення нового набору кімнат	Виконується коректно
Телепортація	Переміщення між кімнатами	Виконується коректно
Збір кристалів	Додавання ресурсу до інвентаря	Виконується коректно
Робота інвентаря	Відображення та встановлення кристалів	Виконується коректно
Масштабування складності	Збільшення рівня складності з часом	Виконується коректно

Під час проведення тестування особлива увага приділялася перевірці стабільності роботи програмного продукту в умовах тривалих ігрових сесій. Оскільки гра належить до жанру Roguelite, важливим аспектом стала перевірка коректності роботи процедурної генерації рівнів та взаємодії окремих підсистем між собою. Багаторазове створення нових конфігурацій кімнат дозволило переконатися у відсутності критичних помилок генерації та забезпеченні доступності всіх частин рівня для гравця.

Під час тестування також аналізувалася поведінка гри в нестандартних ситуаціях. Перевірялися сценарії швидкого переміщення між кімнатами, одночасної присутності великої кількості ворогів та активного використання бойових механік. Окрема увага приділялася перевірці коректності роботи колізій, взаємодії проєктилів із противниками та функціонуванню системи збору кристалів.

Для оцінювання стабільності програмного продукту було проведено серію тривалих ігрових сесій. Це дозволило перевірити коректність роботи системи процедурної генерації рівнів та переконатися у відсутності критичних помилок накопичення об’єктів у пам’яті. Результати випробувань показали стабільну роботу гри та відсутність помітного зниження продуктивності під час тривалого проходження.

Під час тестування оцінювалися такі критерії:

- коректність виконання ігрових механік;
- стабільність роботи системи;
- продуктивність програмного продукту;
- зручність взаємодії користувача з інтерфейсом;
- відсутність критичних помилок;
- відповідність поставленим вимогам.

Отримані результати підтвердили працездатність реалізованих підсистем та показали готовність програмного продукту до подальшого розвитку і вдосконалення.

Окремо перевірялася працездатність системи прогресії персонажа та механізму масштабування складності. Тестування включало збір кристалів різних

типів, встановлення їх у відповідні слоти та аналіз впливу на характеристики персонажа. Додатково оцінювалася коректність зміни параметрів ворогів і героя залежно від поточного рівня складності, що дозволило перевірити баланс ігрового процесу та відповідність реалізованих механізмів початковим вимогам.

Проведене тестування, записане в таблиці 5.3, підтвердило ефективність обраних архітектурних рішень та використаних методів реалізації програмного забезпечення. Модульна структура проєкту забезпечила коректну взаємодію між підсистемами керування персонажем, штучного інтелекту ворогів, генерації рівнів та інтерфейсу користувача. Отримані результати свідчать про готовність програмного продукту до подальшого розвитку та розширення його функціональних можливостей.

5.4. Оцінка результатів

У результаті виконання бакалаврської роботи було розроблено комп'ютерну гру «Etemenamki» жанру Roguelite на платформі Unity. Створений програмний продукт реалізує всі основні механіки, визначені на етапі постановки задачі, включаючи систему керування персонажем, бойову систему, поведінку противників, генерацію рівнів, систему розвитку персонажа та механізм масштабування складності.

Під час розробки було використано сучасні підходи до створення ігрового програмного забезпечення, зокрема компонентну архітектуру Unity, об'єктно-орієнтоване програмування та модульний принцип побудови систем. Це дозволило створити структуру проєкту, яка може бути легко розширена в майбутньому без необхідності суттєвого перепроєктування вже реалізованих компонентів.

Важливим результатом роботи стало створення програмного продукту, який поєднує характерні риси жанру Roguelite із сучасними підходами до розробки програмного забезпечення. Реалізовані механіки забезпечують високу реіграбельність гри, оскільки кожне нове проходження супроводжується зміною структури рівнів, складу противників та можливостей розвитку персонажа. Це

дозволяє підтримувати інтерес користувача протягом тривалого часу та створює передумови для подальшого розширення функціональності проєкту.

Проведене тестування підтвердило працездатність основних механік гри та їхню коректну взаємодію між собою. Реалізована система генерації рівнів забезпечує різноманітність проходжень, а механізм масштабування складності підтримує інтерес користувача протягом тривалих ігрових сесій. Система кристалів надає можливість формування різних варіантів розвитку персонажа та створює додаткову стратегічну складову ігрового процесу.

Перспективи подальшого розвитку гри включають розширення набору доступних кімнат, додавання нових типів ворогів і бойових механік, а також реалізацію системи довготривалої прогресії між окремими проходженнями. Крім того, можливим напрямом розвитку є адаптація програмного продукту для інших платформ та вдосконалення системи балансування складності. Це дозволить підвищити якість ігрового процесу та розширити потенційну аудиторію користувачів. Аналіз отриманих результатів показав, що використання рушія Unity та мови програмування C# є ефективним рішенням для створення двовимірних ігор жанру Roguelite. Компонентна архітектура рушія дозволила реалізувати незалежні підсистеми, які взаємодіють між собою та можуть бути легко модифіковані в майбутньому. Це особливо важливо для ігрових проєктів, які передбачають регулярне розширення функціональності та додавання нового контенту.

Практична цінність розробленого програмного продукту полягає не лише у створенні гри, а й у можливості використання реалізованих механік як основи для подальших досліджень у сфері процедурної генерації контенту та побудови адаптивних ігрових систем. Отримані результати можуть бути використані під час розробки інших проєктів, пов’язаних зі створенням інтерактивних програмних систем.

Розроблена гра демонструє можливості сучасних технологій розробки програмного забезпечення та підтверджує ефективність застосування принципів об’єктно-орієнтованого програмування у сфері створення комп’ютерних ігор.

Отримані результати свідчать про успішне досягнення поставленої мети

роботи. Розроблений програмний продукт може бути використаний як основа для подальшого розвитку проєкту, зокрема шляхом додавання нових типів противників, додаткових кімнат, системи довготривалої прогресії між проходженнями та підтримки інших програмних платформ.

5.5. Висновки по розділу

У четвертому розділі було розглянуто результати реалізації програмного продукту та проведено оцінку його працездатності. Виконано опис користувацького інтерфейсу гри «Etemenamki», який забезпечує зручну взаємодію гравця з основними ігровими механіками. Реалізовані елементи інтерфейсу надають користувачеві доступ до інформації про стан персонажа, рівень складності, інвентар та систему розвитку, не перевантажуючи ігровий екран зайвими деталями. Використання засобів Unity UI та TextMeshPro дозволило забезпечити коректне відображення елементів інтерфейсу та їх узгодженість із загальним візуальним стилем гри.

У ході аналізу ігрового процесу було встановлено, що реалізовані механіки формують цілісний цикл проходження, який включає дослідження рівнів, боротьбу з противниками, збір кристалів та розвиток персонажа. Поєднання бойової системи, механік прогресії та елементів випадковості відповідає особливостям жанру Roguelite та сприяє підвищенню реіграбельності програмного продукту.

Для перевірки працездатності розробленої гри було проведено тестування основних функціональних компонентів. У процесі тестування перевірялася коректність роботи системи керування персонажем, бойової системи, генерації рівнів, інвентаря, механізму збору кристалів та системи масштабування складності. Отримані результати підтвердили відповідність фактичної роботи програмного продукту поставленим вимогам та відсутність критичних помилок, які могли б унеможливити використання гри або суттєво вплинути на ігровий процес.

За результатами проведеної оцінки встановлено, що поставлена мета бакалаврської роботи була досягнута. Розроблений програмний продукт реалізує

всі передбачені функціональні можливості та демонструє стабільну роботу основних підсистем. Запропонована архітектура дозволяє продовжувати розвиток проєкту шляхом додавання нових типів ворогів, ігрових механік, рівнів та елементів довготривалої прогресії, що створює основу для подальшого вдосконалення гри та розширення її функціональності.

ВИСНОВКИ

У бакалаврській роботі вирішено задачу розробки комп’ютерної гри «Etemenamki» жанру Roguelite із використанням ігрового рушія Unity та мови програмування C#. У процесі виконання роботи було досліджено особливості жанру Roguelite, сучасні підходи до розробки ігрового програмного забезпечення та засоби створення інтерактивних двовимірних застосунків. Проведений аналіз предметної області дозволив визначити основні функціональні вимоги до програмного продукту та сформувати концепцію гри, орієнтованої на багаторазове проходження та поступове зростання складності.

На основі проведеного аналізу було обґрунтовано вибір ігрового рушія Unity, мови програмування C# та середовища розробки Microsoft Visual Studio. Використання зазначених технологій дозволило реалізувати модульну архітектуру програмного забезпечення, яка забезпечує зручність розширення функціональності, підтримку окремих підсистем та подальший розвиток проєкту. Також було досліджено можливості вбудованих інструментів Unity та пакетів, що використовувалися під час створення гри.

У роботі реалізовано основні ігрові механіки, характерні для жанру Roguelite. Зокрема, розроблено систему керування персонажем, бойову систему з підтримкою дистанційних та ближніх атак, систему поведінки противників на основі машини станів, а також механізм процедурного формування рівнів із використанням набору шаблонних кімнат. Додатково реалізовано систему масштабування складності, яка забезпечує поступове ускладнення проходження шляхом зміни характеристик персонажа та противників залежно від тривалості ігрової сесії.

Особливу увагу приділено створенню системи прогресії персонажа, що базується на механіці збору кристалів різних типів та їх подальшому використанні для покращення характеристик героя. Реалізована система інвентаря та слотів дозволяє формувати різні варіанти розвитку персонажа та підвищує варіативність ігрового процесу. Використання окремих структур даних для зберігання

параметрів кристалів забезпечило гнучкість налаштування балансу гри та спростило подальше розширення функціональності.

Проведене тестування підтвердило коректність роботи основних підсистем програмного продукту та їхньої взаємодії між собою. У процесі перевірки було встановлено, що реалізовані механіки керування, бойової взаємодії, генерації рівнів, розвитку персонажа та масштабування складності функціонують відповідно до поставлених вимог. Отримані результати свідчать про досягнення мети бакалаврської роботи та успішне виконання поставлених завдань.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. **Потієнко О. І.** Програмування мовою C# : навчальний посібник. - Київ : КПІ ім. Ігоря Сікорського, 2020. - 312 с.
2. **Морзе Н. В., Кузьмінська О. Г.** Об’єктно-орієнтоване програмування : навчальний посібник. - Київ : Університет «Україна», 2019. - 280 с.
3. **Завадський І. О.** Основи програмної інженерії : навчальний посібник. - Київ : Видавничий дім «Слово», 2018. - 352 с.
4. **Биков В. Ю., Спірін О. М.** Інформаційні технології та програмні системи : навчальний посібник. - Київ : Атіка, 2017. - 416 с.
5. **Глинський Я. М.** Практикум з інформатики та програмування. - Львів : Деол, 2021. - 304 с.
6. **Пасічник В. В., Шаховська Н. Б.** Організація баз даних та знань. - Львів : Магнолія 2006, 2020. - 444 с.
7. **Трофименко О. Г., Прокоп Ю. В.** Сучасні технології розробки програмного забезпечення : навчальний посібник. - Одеса : ОНПУ, 2021. - 287 с.
8. **Unity Technologies. Unity Manual:** https://docs.unity3d.com/Manual/
9. **Unity Technologies. Unity Scripting API**: https://docs.unity3d.com/ScriptReference/
10. **Microsoft Corporation. C# Documentation:** https://learn.microsoft.com/en-us/dotnet/csharp/
11. **Nystrom R.** Game Programming Patterns. - USA : Genever Benning, 2014. - 354 p.
12. **Schell J.** The Art of Game Design: A Book of Lenses. - 4th ed. - Boca Raton : CRC Press, 2019. - 640 p.
13. **Rogers S.** Level Up! The Guide to Great Video Game Design. - 3rd ed. - Hoboken : Wiley, 2014. - 550 p.
14. **Shaker N., Togelius J., Nelson M.** Procedural Content Generation in Games. - Springer, 2016. - 237 p.

15. **Fullerton T.** Game Design Workshop: A Playcentric Approach to Creating Innovative Games. - 5th ed. - CRC Press, 2024. - 566 p.

16. **Gregory J.** Game Engine Architecture. - 3rd ed. - Boca Raton : CRC Press, 2018. - 1136 p.

17. **Millington I.** Artificial Intelligence for Games. - 3rd ed. - Boca Raton : CRC Press, 2019. - 900 p.

18. **Buckland M.** Programming Game AI by Example. - Plano : Jones & Bartlett Learning, 2005. - 494 p.

19. **Yannakakis G. N., Togelius J.** Artificial Intelligence and Games. - Cham : Springer, 2018. - 716 p.

20. **Novak J.** Game Development Essentials: An Introduction. - 3rd ed. - Clifton Park : Delmar Cengage Learning, 2011. - 640 p.

21. **Salen K., Zimmerman E.** Rules of Play: Game Design Fundamentals. - Cambridge : MIT Press, 2004. - 688 p.

22. **Adams E.** Fundamentals of Game Design. - 4th ed. - Berkeley : New Riders, 2014. - 592 p.

23. **Rabin S.** Introduction to Game Development. - 2nd ed. - Boston : Charles River Media, 2010. - 980 p.

24. **Dormans J., Bakkes S.** Generating Missions and Spaces for Adaptable Play Experiences // IEEE Transactions on Computational Intelligence and AI in Games. - 2011. - Vol. 3, No. 3. - P. 216-228.

25. **Togelius J., Kastbjerg E., Schedl D., Yannakakis G.** What Is Procedural Content Generation? // Proceedings of the FDG Workshop on Procedural Content Generation in Games. - 2011. - P. 1-6.

26. **Lewis M., Jacobson J.** Game Engines in Scientific Research // Communications of the ACM. - 2002. - Vol. 45, No. 1. - P. 27-31.

27. **Rollings A., Adams E.** Fundamentals of Game Design. - Upper Saddle River : Prentice Hall, 2006. - 480 p.

28. **Bethke E.** Game Development and Production. - 3rd ed. - Plano : Wordware Publishing, 2003. - 928 p.

29. **Eberly D.** 3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics. - 2nd ed. - Boca Raton : CRC Press, 2006. - 1040 p.

30. **McShaffry M., Graham D.** Game Coding Complete. - 4th ed. - Boston : Cengage Learning, 2012. - 1024 p.

31. **Funge J.** Artificial Intelligence for Computer Games: An Introduction. - Boca Raton : CRC Press, 2004. - 256 p.

32. **Novak J.** Developing Virtual Worlds with Unity 2022 and C#. - Berkeley : Apress, 2023. - 520 p.

33. **Goldstone W.** Unity Game Development Essentials. - Birmingham : Packt Publishing, 2009. - 488 p.

34. **Hocking J.** Unity in Action: Multiplatform Game Development in C#. - 3rd ed. - Shelter Island : Manning Publications, 2022. - 400 p.

35. **Murray J.** Hamlet on the Holodeck: The Future of Narrative in Cyberspace. - Cambridge : MIT Press, 2017. - 448 p.

ДОДАТКИ

RoomManager

```
using System.Collections.Generic;
using UnityEngine;
public class RoomManager : MonoBehaviour
{
    public static RoomManager Instance;
    [Header("Lobby")]
    public TeleportPortal lobbyPortal;
    public Transform firstRoomSpawnPoint;

    [Header("Rooms")]
    public Room[] roomPrefabs;

    [Header("Generation")]
    public float roomSpacing = 35f;
    private readonly List<Room> spawnedRooms = new();
    private void Awake()
    {
        Instance = this;
    }
    private void Start()
    {
        GenerateFirstRoom();
    }
    private void GenerateFirstRoom()
    {
        Room room = Instantiate(
            roomPrefabs[Random.Range(0, roomPrefabs.Length)],
            firstRoomSpawnPoint.position,
            Quaternion.identity);
        room.roomIndex = 1;
        lobbyPortal.linkedPortal = room.entryPortal;
        spawnedRooms.Add(room);
    }
```

```
    public void PlayerEnteredRoom(Room room)
    {
        if (room.nextRoomGenerated)
            return;
        GenerateNextRoom(room);
        room.nextRoomGenerated = true;
    }
    private void GenerateNextRoom(Room currentRoom)
    {
        int nextIndex = currentRoom.roomIndex + 1;

        Vector3 spawnPosition =
            firstRoomSpawnPoint.position +
            Vector3.right * roomSpacing * nextIndex;

        Room nextRoom = Instantiate(
            roomPrefabs[Random.Range(0, roomPrefabs.Length)],
            spawnPosition,
            Quaternion.identity);

        nextRoom.roomIndex = nextIndex;
        currentRoom.exitPortal.linkedPortal = nextRoom.entryPortal;
        spawnedRooms.Add(nextRoom);
        if (spawnedRooms.Count > 3)
        {
            Destroy(spawnedRooms[0].gameObject);
            spawnedRooms.RemoveAt(0);
        }
    }
}
```

DifficultyTimer

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;
```

```
public class DifficultyTimer : MonoBehaviour
{
    [Header("Timer")]
    public RectTransform arrow;
    public float rotationTime = 60f;

    [Header("Difficulty")]
    public int difficultyLevel = 0;

    [Header("UI")]
    public TMP_Text difficultyText;

    private float currentTime;

    private void Start()
    {
        UpdateDifficultyText();
    }

    private void Update()
    {
        currentTime += Time.deltaTime;

        float progress = currentTime / rotationTime;

        // Оберт за годинниковою стрілкою
        arrow.localRotation = Quaternion.Euler(0, 0, -progress * 360f);

        if (currentTime >= rotationTime)
        {
            currentTime -= rotationTime;

            difficultyLevel++;

            PlayerStatSystem stats =
```

```
            FindFirstObjectByType<PlayerStatSystem>();

        if (stats != null)
        {
            stats.RecalculateStats();
        }

        UpdateDifficultyText();
    }
  }

  private void UpdateDifficultyText()
  {
    if (difficultyText != null)
    {
        difficultyText.text = difficultyLevel.ToString();
    }
  }
}
```

PlayerStats

```
using UnityEngine;

[System.Serializable]
public class PlayerStats
{
  [Header("Health")]
  public float maxHealth = 10;
  public float healthRegen = 0;
  public float damageTakenMultiplier = 1;

  [Header("Mana")]
  public float maxMana = 100;
  public float manaRegen = 0;
  public float manaCostMultiplier = 1;
```

```
    [Header("Projectile")]
    public float projectileDamage = 1;
    public float projectileSpeed = 12;
    public float projectileSize = 1;
    public float projectileCooldown = 0.2f;

    public float projectileSpread = 0;
    public int projectileCount = 1;

    [Header("Slash")]
    public float slashDamage = 2;
    public float slashSize = 1;
    public float slashCooldown = 0.22f;
    public float slashComboCooldown = 0.12f;

    public int maxComboAttacks = 4;

    public bool comboEnabled = true;

    public float firstSlashSizeMultiplier = 1.15f;

    [Header("Mana Gain")]
    public float manaRestoreOnSlashHit = 8;
}
```

EnemyBrain

```
using UnityEngine;

public class EnemyBrain : MonoBehaviour
{
    [Header("Target")]
    private Transform player;
    public Transform Player => player;
```

```
public EnemyData data;

[Header("Ranges")]
private float detectionRange =>
    data.detectionRange;
private float attackRange =>
    data.attackRange;

[Header("Attack")]
[SerializeField]
private bool ignoreAttackRange;

[Header("State")]
public EnemyState currentState;
private EnemyMovement movement;
private EnemyAttack attack;
private EnemyHealth health;
private bool wasProvoked = false;
private void Awake()
{
    movement =
        GetComponent<EnemyMovement>();
    attack =
        GetComponent<EnemyAttack>();
    health =
        GetComponent<EnemyHealth>();
}

private void Start()
{
    player =
        GameObject.FindGameObjectWithTag(
            "Player"
        ).transform;
}
private void Update()
```

```
{
    if (health.isDead)
    {
        currentState =
            EnemyState.Dead;

        return;
    }
    float distance =
        Vector2.Distance(
            transform.position,
            player.position
        );
    switch (currentState)
    {
        case EnemyState.Idle:
            if (distance <= detectionRange)
            {
                currentState =
                    EnemyState.Chase;
            }
            if (wasProvoked)
            {
                currentState =
                    EnemyState.Chase;
            }
            break;
        case EnemyState.Chase:
            movement.MoveTo(
                player.position
            );
            bool canAttack =
                ignoreAttackRange ||
                distance <= attackRange;

            if (canAttack)
```

```
            {
                currentState =
                    EnemyState.Attack;
            }

            break;

        case EnemyState.Attack:
            attack.DoAttack();
            if (!ignoreAttackRange)
            {
                if (distance > attackRange)
                {
                    currentState =
                        EnemyState.Chase;
                }
            }

            break;
    }
  }
  public void Provoke()
  {
    wasProvoked = true;
    if (currentState ==
        EnemyState.Idle)
    {
        currentState =
            EnemyState.Chase;
    }
  }
}
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: “ Розробка комп’ютерної гри Etemenamki в жанрі Roguelite, використовуючи серидовище Unity”

”

Обсяг пояснювальної записки: 72 аркуші

Дата закінчення роботи 10 червня 2026р.

Підпис _____