Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА ДВОМІРНОЇ ГРИ«ADVENTUROUS WORLD»В ЖАНРІ ADVENTURE-RPG З ВИКОРИСТАННЯМ UNITY

DEVELOPMENT OF THE 2D GAME«ADVENTUROUS WORLD»IN THE ADVENTURE-RPG GENRE USING UNITY

спеціальність 122 Комп'ютерні науки

освітня програма «Комп'ютерні науки»

Виконав: здобувач вищої освіти
групи КН-42
Тлумач Станіслав Сергійович

_____
(підпис)

Керівник: асистент
Сачук Вікторія Олегівна

_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«___» _____ 2026 р.
Гарант освітньої програми:
д.пед.н., професор
Тулашвілі Юрій Йосипович

_____
(підпис)

Луцьк – 2026 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних та інформаційних технологій
Кафедра комп'ютерних наук
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *122 Комп'ютерні науки*
Освітня програма: *«Комп'ютерні науки»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Валерій ЛІЩИНА
«16» січня 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
ПЕРШОГО (БАКАЛАВРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Тлумач Станіслав Сергійович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Розробка 2D гри на Unity та C#»
Керівник асистент Сачук Вікторія Олегівна
затверджені наказом закладу вищої освіти від «*16*» *січня 2026* р. № 32/01-02

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи *«02» чернвня 2026 р.*

3. Вихідні дані до роботи:
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити):
аналіз предметної області, постановка завдання, визначення вимог, реалізація і
логіка, тестування та виправлення
_____
_____

5. Перелік графічного матеріалу:6 рисунків, 11лістингів
_____
_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблеми за темою роботи та постановка завдань дослідження	*Сачук В. О.*		
Теоретичне дослідження	*Сачук В. О.*		
Практична реалізація	*Сачук В. О.*		
Спеціальні розрахунки	*Сачук В. О.*		
Показник запозичень тексту	_____ %		
Інструментальна перевірка	*Кошелюк В. А.*		
Нормоконтроль	*Сачук В. О.*		
Гарант ОПП	*Тулашвілі Ю. Й.*		

7. Дата видачі завдання *«16» січня 2026 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	*Провести огляд літературних джерел по темі кваліфікаційної роботи*	*до 17.02.2026 р*	
2	*Провести аналіз загальної проблеми і вибір напрямків дослідження*	*до 28.02.2025 р.*	
3	*Розробити функціональну схему роботи програмного продукту*	*до 12.03.2026 р*	
4	*Описати засоби розробки об'єкта проектування*	*до 26.03.2026 р.*	
5	*Практична реалізація об'єкта проектування*	*до 30.04.2026 р.*	
6	*Провести спеціальні розрахунки*	*до 18.05.2026 р.*	
7	*Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру*	*до 02.06.2026 р.*	

Здобувач вищої освіти _____Станіслав ТЛУМАЧ

Керівник роботи _____ Вікторія САЧУК

Анотація

Тлумач С. С. Розробка двовимірної гри «ADVENTUROUS WORLD» в жанрі advanture-RPG з використанням Unity. Рукопис. Кваліфікаційна робота бакалавра ОП «Кмпютерні науки» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі здійснено аналіз предметної області, визначено актуальність теми та сформульовано завдання кваліфікаційної роботи. У другому розділі розглянуто вимоги до програмного забезпечення та спроектовано архітектуру майбутньої гри. У третьому розділі описано процес реалізації ігрового застосунку: розробку механік, інтерфейсу, навігації, анімації та тестування. У висновках наведено результати роботи, оцінено ефективність реалізації та перспективи подальшого розвитку проекту.

Ключові слова: Unity, C#, Visual Studio, спрайти, скрипти, 2D-гра, ігровий рушій, програмна архітектура, геймдизайн, анімації.

ABSTRACT

Tlumach S. Development of the 2D Game «ADVENTUROUS WORLD» in the Adventure-RPG Genre Using Unity. Manuscript. Bachelor's qualification thesis of the educational program «Software Engineering» specialty «Software Engineering» Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification thesis consists of an introduction, three chapters, conclusions, a list of references, and appendices.

The first chapter presents an analysis of the subject area, outlines the relevance of the topic, and formulates the objectives of the thesis. The second chapter discusses the software requirements and presents the architecture design of the future game. The third chapter describes the implementation process of the game application, including the development of gameplay mechanics, user interface, navigation, animation, and testing. The conclusions summarize the results of the work, evaluate the implementation effectiveness, and outline the prospects for further development of the project.

Keywords: Unity, C#, Visual Studio, sprites, scripts, 2D game, game engine, software architecture, game design, animations.

ЗМІСТ

ВСТУП

Актуальність теми. Створення ігор у двовимірному форматі не втрачає своєї значущості в сучасній індустрії програмного забезпечення: цей сегмент динамічно зростає як у великих комерційних компаніях, так і серед інді-розробників. Платформа Unity посідає провідне місце серед інструментів для розробки 2D-проєктів завдяки доступності освоєння, багатофункціональному середовищу, кросплатформеності та розгалуженій екосистемі розробників. Проектування гри дозволяє поєднати технічні знання з творчим підходом, охоплюючи програмування, анімацію, дизайн, логіку поведінки об’єктів та взаємодію з гравцем.

Метою цієї кваліфікаційної роботи є розробка повноцінного 2D-рольового ігрового застосунку «ADVENTUROUS WORLD» на базі Unity із застосуванням мови C#. Гра може використовуватися не лише як навчальний приклад, але й як засіб емоційного розвантаження, зокрема для дітей-переселенців, яким подібні інтерактивні проекти можуть допомогти адаптуватися до нових умов через захопливу гру.

Об’єктом кваліфікаційної роботи є процес створення програмного забезпечення для інтерактивної 2D-гри.

Предметом кваліфікаційної роботи є архітектура, логіка та методи реалізації функціоналу гри в середовищі Unity.

Для реалізації поставленої мети визначено наступні завдання:

– проаналізувати наявні рішення у сфері 2D-геймдеву;

– сформувати технічне завдання та вимоги до функціоналу;

– спроектувати архітектуру гри та її основні модулі;

– реалізувати ключові ігрові механіки (управління персонажем, бойова система);

– провести тестування ігрового застосунку та виправити виявлені помилки;

– підготувати супровідну документацію до розробленого продукту

РОЗДІЛ 1

АНАЛІЗ СЧАСНИХ РІШЕНЬ У СФЕРІ РОЗРОБКИ ІГОР ТА ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Сфера комп'ютерних ігор належить до найбільш стрімко прогресуючих напрямів цифрових технологій. Незважаючи на те, що сучасний ринок значною мірою орієнтований на тривимірні проєкти з фотореалістичною графікою та складними фізичними моделями, двовимірні ігри продовжують утримувати міцні позиції. Їхня стійка присутність в індустрії пояснюється низкою об'єктивних переваг: меншою ресурсоємністю, виразним художнім стилем і можливістю сконцентруватися на глибині ігрового процесу без необхідності у складній тривимірній візуалізації.

Тривимірні проєкти висувають суттєві вимоги до команди розробників та апаратного забезпечення. Побудова об'ємних моделей, налаштування систем освітлення, рендерингу та тривимірної анімації потребує значних витрат часу, кваліфікації та фінансових ресурсів, що робить такі проєкти важкодосяжними для невеликих колективів або одиночних авторів. Натомість двовимірний формат дає змогу зосередити зусилля на проробленості механік, наративу та візуальної концепції.

Успіх таких проєктів, як «Hollow Knight», «Celeste» та «Dead Cells», переконливо демонструє: грамотно збудована 2D-гра здатна конкурувати з тривимірними аналогами за рівнем захопленості гравця та комерційним результатом. Ці приклади стимулюють інтерес до створення авторських двовимірних ігор із власною ідентичністю та оригінальними механіками. Зокрема, «Hollow Knight» відомий глибокою бойовою системою ближнього бою та дослідженням відкритого світу — підходи, що знайшли відображення і в розробленому проєкті «Adventurous World».

Серед доступних інструментів розробки особливе місце займає рушій Unity — одна з найпоширеніших платформ для створення як 2D-, так і 3D-

застосунків. Безкоштовна ліцензія для малих команд і індивідуальних розробників робить його доступним для широкого кола користувачів — від студентів до комерційних інді-студій. Інтеграція з редактором Visual Studio та підтримка мови C# забезпечують гнучкість архітектурних рішень, можливість повторного використання коду та підключення зовнішніх бібліотек.

Вбудований інструментарій Unity охоплює компоненти для роботи з тайловими картами (Tilemap), анімаціями (Animator), системою частинок, навігаційними сітками (NavMesh), а також засоби побудови користувацького інтерфейсу (Canvas/UI). У контексті даного проєкту особливо важливою виявилася підтримка NavMesh для двовимірного простору — завдяки підключенню пакету NavMeshComponents було реалізовано інтелектуальне переміщення ворогів по ігровому рівні. Це дозволяє реалізувати повноцінний ігровий продукт у межах єдиного середовища без залучення сторонніх рушіїв.

Нова система введення Unity Input System, задіяна через клас PlayerInputActions, надає гнучкий і розширюваний спосіб обробки команд гравця. На відміну від застарілого підходу Input.GetAxis, вона дозволяє чітко розмежувати карти дій (переміщення, атака, ривок) і забезпечує коректну обробку подій через систему делегатів C#. Такий підхід широко використовується у сучасних комерційних проєктах і є рекомендованим стандартом для нових розробок на Unity.

Варто наголосити, що 2D-ігри функціонують на широкому спектрі пристроїв, зокрема на слабопотужних комп'ютерах і мобільних платформах. Це суттєво розширює потенційну аудиторію продукту та відкриває можливості для публікації на таких майданчиках, як Steam, Itch.io, Google Play та App Store, де щоденно з'являються нові проєкти від незалежних авторів.

З іншого боку, розробка навіть відносно нескладної двовимірної гри передбачає одночасне застосування знань із різних дисциплін: проєктування ігрових механік, написання програмного коду, роботи з анімацією, побудови інтерфейсу, оптимізації продуктивності та тестування. До типових складнощів, що виникають у процесі розробки, належать: коректна обробка зіткнень між

об'єктами, реалізація відгуку на отримання пошкоджень (hit-feedback), побудова масштабованої архітектури та реалізація зрозумілого інтерфейсу для гравця.

Отже, тематика розробки власного 2D-ігрового застосунку є не лише практично значущою для студентів спеціальності «Інженерія програмного забезпечення», а й перспективною з точки зору опанування сучасного технологічного стеку та формування портфоліо.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

У рамках кваліфікаційної роботи було визначено завдання щодо проєктування та програмної реалізації повноцінної двовимірної комп'ютерної гри в жанрі adventure-RPG під назвою «Adventurous World». Розробка здійснювалася у середовищі Unity із використанням мови програмування C#. Ключовою метою стало створення продукту, що органічно поєднує дослідження ігрового світу, механіки бойових взаємодій ближнього бою, систему пошкоджень і поведінку ворогів на основі штучного інтелекту.

Для досягнення визначеної мети були сформульовані такі завдання:

– провести огляд предметної галузі та дослідити актуальні підходи до розробки двовимірних ігор у жанрі Adventure-RPG; вивчити сучасні ігрові механіки, алгоритми поведінки ворогів і принципи побудови анімаційних систем;

– визначити функціональні та нефункціональні вимоги до продукту, охоплюючи вимоги до швидкодії, структури ігрових рівнів та логіки бойових взаємодій;

– реалізувати систему керування персонажем на основі Rigidbody2D з підтримкою переміщення, орієнтації у просторі та механіки ривка (dash) з відображенням через TrailRenderer;

– розробити бойову систему: атака мечем через PolygonCollider2D, завдання та отримання пошкоджень, тимчасовий імунітет після удару та візуальний ефект спалаху (FlashBlink)

– реалізувати компонент фізичного відкидання (KnockBack) на основі Rigidbody2D.AddForce з керованою тривалістю та автоматичним відновленням після імпульсу;

– запрограмувати логіку поведінки ворога-скелета із скінченним автоматом станів: Idle, Roaming, Chasing, Attacking, Death; навігацію реалізовано через NavMeshAgent із динамічним коригуванням швидкості;

– побудувати анімаційні контролери для персонажа та скелета, а також окремі контролери для анімацій удару мечем у різних напрямках;

– організувати систему сцен із головним меню, переходами між рівнями та екраном завершення гри;

– провести комплексне тестування із застосуванням ручної перевірки, візуального відлагодження в Unity Editor та аналізу граничних ситуацій.

У процесі роботи було опановано ключові технологічні компетентності: принципи роботи рушія Unity, побудову 2D-анімацій, використання колайдерів і тригерів, патерн «Одинак» (Singleton) для глобального доступу до систем, а також організацію коду через систему подій C# (EventHandler).

Підсумком виконання перелічених завдань є інтерактивний ігровий застосунок із реалізованими ключовими механіками двовимірної рольової гри, що може слугувати основою для подальшого масштабування — додавання нових рівнів, типів ворогів та ігрових систем.

РОЗДІЛ 2

СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

На початковому етапі розробки гри «Adventurous World» було необхідно сформувати чіткі вимоги до продукту — як з боку очікуваного ігрового досвіду, так і з боку технічної реалізації. Оскільки проєкт належить до жанру 2D adventure-RPG, він має задовольняти функціональні потреби гравця та водночас відповідати вимогам стабільної й масштабованої архітектури. Правильно визначені вимоги на ранньому етапі дозволяють уникнути суттєвого рефакторингу коду в майбутньому та забезпечують узгодженість усіх підсистем гри між собою.

На підготовчому етапі було проведено аналіз аналогічних проєктів, реалізованих у середовищі Unity. Зокрема, вивчалась структура таких ігор, як «Dead Cells», «Hollow Knight» та «Celeste». Кожен із цих проєктів поєднує стилізовану візуальну складову з продуманою ігровою механікою та плавною взаємодією з гравцем. Аналіз показав, що ключовими чинниками успіху подібних проєктів є чітка архітектура коду, виразний візуальний зворотний зв'язок на дії гравця та збалансована система складності. За орієнтири при побудові «Adventurous World» було взято модульність архітектури, анімаційну плавність переходів між станами, а також можливість подальшого масштабування без необхідності переробки базового коду.

Вимоги до програмного продукту поділено на функціональні та нефункціональні. До функціональних вимог належать: вільне переміщення персонажа у восьми напрямках із використанням фізичного компонента Rigidbody2D; механіка ривка (dash) з візуальним ефектом залишкового сліду через TrailRenderer та часовим обмеженням повторного використання; бойова взаємодія ближнього бою з мечем через PolygonCollider2D; система отримання пошкоджень із тимчасовим імунітетом після удару та візуальним ефектом спалаху (FlashBlink); фізичне відкидання персонажа через імпульсну силу

(KnockBack); поведінка ворогів на основі скінченного автомата станів із навігацією через NavMeshAgent; відстеження рівня здоров'я гравця та ворогів із обробкою події загибелі; логічна організація сцен із переходами між ними.

До нефункціональних вимог віднесено: стабільну частоту кадрів на середньопотужних пристроях, логічну структуру коду що допускає розширення без рефакторингу основних класів, чітке розмежування відповідальності між компонентами та відсутність прямих залежностей між непов'язаними системами. Окремою вимогою є зрозумілість інтерфейсу для гравця: усі ігрові події мають супроводжуватись відповідним візуальним або анімаційним відгуком.

Загальну архітектуру гри побудовано за модульним принципом: кожен ігровий об'єкт керується набором незалежних скриптів із чітко визначеною відповідальністю. Взаємодія між компонентами організована через події C# (EventHandler), а для глобального доступу до ключових систем застосовано патерн «Одинак» (Singleton). Структурну організацію основних класів та зв'язки між ними наведено на рисунку 2.1.

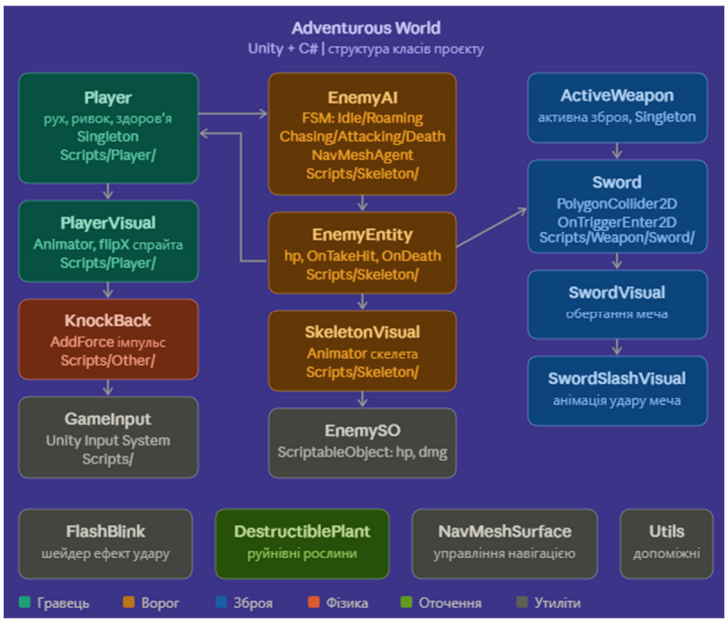

Рисунок 2.1 – Компонентна структура гри «Adventurous World»

Ініціалізація головного персонажа при запуску сцени наведена у (лістингу 2.1). У методі Awake зберігаються посилання на компоненти Rigidbody2D та KnockBack, а також встановлюється глобальний синглтон-екземпляр для доступу з інших систем.

Лістинг 2.1 – Ініціалізація героя

```
private void Awake() {
    Instance = this;
    _rb = GetComponent<Rigidbody2D>();
    _knockBack = GetComponent<KnockBack>();
    _initialMovingSpeed = movingSpeed;
}
```

кінець лістингу 2.1

Переміщення персонажа (лістинг 2.2) реалізовано через Rigidbody2D.MovePosition у методі FixedUpdate, що забезпечує фізично коректне та плавне пересування без ривків. Одночасно відстежується стан руху для перемикання анімацій. Переміщення блокується на час дії ефекту відкидання.

Лістинг 2.2 – Рух героя

```
private void HandleMovement() {
    _rb.MovePosition(_rb.position + _inputVector *
(movingSpeed * Time.fixedDeltaTime));
    _isRunning = Mathf.Abs(_inputVector.x) > _minMovingSpeed
              || Mathf.Abs(_inputVector.y) > _minMovingSpeed;
}
```

кінець лістингу 2.2

Система отримання пошкоджень (лістинг 2.3) передбачає зменшення поточного рівня здоров'я, активацію ефекту відкидання та запуск корутини тимчасового імунітету. Це унеможливлює повторне пошкодження одразу після удару та дає гравцеві час для реакції.

Лістинг 2.3 – Отримання шкоди

```
public void TakeDamage(Transform damageSource, int damage) {
    if (_canTakeDamage && _isAlive) {
        _canTakeDamage = false
        _currentHealth = Mathf.Max(0, _currentHealth -=
damage);
        _knockBack.GetKnockedBack(damageSource);
        OnFlashBlink?.Invoke(this, EventArgs.Empty);
        StartCoroutine(DamageRecoveryRoutine());
    }
    DetectDeath();
}
```

кінець лістингу 2.3

Компонент фізичного відкидання KnockBack (лістинг 2.4) є універсальним і використовується як для гравця, так і може бути застосований до інших об'єктів. Сила відкидання обчислюється як нормалізований вектор від джерела пошкодження до цілі, помножений на задану величину імпульсу.

Лістинг 2.4 – Відштовхування головного героя

```
public void GetKnockedBack(Transform damageSource) {
    IsGettingKnockedBack = true;
    _knockBackMovingTimer = knockBackMovingTimerMax;
    Vector2 difference = (transform.position -
damageSource.position).normalized
                         * knockBackForce;
    _rb.AddForce(difference, ForceMode2D.Impulse);
}
```

кінець лістингу 2.4

Орієнтація спрайта персонажа (лістинг 2.5) визначається шляхом порівняння позиції курсора миші з позицією гравця на екрані. Це забезпечує інтуїтивне управління — герой завжди повернутий у бік дії гравця, що також важливо для коректного відображення атак.

Лістинг 2.5 – Визначення напрямку погляду гравця

```
private void AdjustPlayerFacingDirection() {
    Vector3 mousePos = GameInput.Instance.GetMousePosition();
    Vector3 playerPos =
Player.Instance.GetPlayerScreenPosition();
    spriteRenderer.flipX = mousePos.x < playerPos.x;
}
```

кінець лістингу 2.5

Атака ворога (лістинг 2.6) побудована на основі системи подій C#. Після завершення таймера перезарядки викликається подія OnEnemyAttack, на яку підписані відповідні обробники в інших компонентах, що усуває пряму залежність між системами.

Лістинг 2.6 – Атака ворога

```
private void AttackingTarget() {
    if (Time.time > _nextAttackTime) {
        OnEnemyAttack?.Invoke(this, EventArgs.Empty);
        _nextAttackTime = Time.time + attackRate;
    }
}
```

кінець лістингу 2.6

Зміна візуальної орієнтації ворога (лістинг 2.7) здійснюється через обертання transform.rotation. Порівнюються координати джерела та цілі переміщення, після чого спрайт розгортається у відповідний бік. Цей метод застосовується як під час руху, так і під час фази атаки.

Лістинг 2.7 – Візуальна анімація смерті

```
private void ChangeFacingDirection(Vector3 sourcePosition,
Vector3 targetPosition) {
    transform.rotation = sourcePosition.x > targetPosition.x
        ? Quaternion.Euler(0, -180, 0)
        : Quaternion.Euler(0, 0, 0);
}
```

кінець лістингу 2.7

Таким чином, проєктування гри «Adventurous World» ґрунтувалося на чіткому визначенні вимог та побудові модульної архітектури, що забезпечує незалежність компонентів, зрозумілу структуру коду та можливість подальшого розширення проєкту.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

У процесі реалізації проєкту було визначено інструментарій, методи та алгоритмічні підходи, що забезпечують ефективну розробку з урахуванням вимог до продуктивності, модульності та розширюваності.

Основним середовищем розробки обрано ігровий рушій Unity завдяки розвиненому набору вбудованих підсистем: фізичному рушію на основі Box2D, системі анімацій Animator, інструментам побудови рівнів Tilemap, підтримці навігаційних сіток NavMesh та засобам побудови інтерфейсу через Canvas. Мовою програмування є C# — сучасна об'єктно-орієнтована мова з підтримкою делегатів, подій, корутин та узагальнень. Редактором коду слугує Visual Studio з повноцінною підтримкою IntelliSense, вбудованим відлагоджувачем та інтеграцією з Unity Editor.

У проєкті застосовано такі методи та підходи:

Компонентно-орієнтована архітектура — кожен ігровий об'єкт є набором незалежних компонентів із чітко визначеною відповідальністю. Наприклад, логіка пересування, обробка пошкоджень і візуальні ефекти реалізовані в окремих класах (Player, KnockBack, FlashBlink), що спрощує тестування та повторне використання.

Патерн «Одинак» (Singleton) — для глобального доступу до ключових систем без жорстких залежностей між класами використовується статичний екземпляр (Player.Instance, GameInput.Instance, ActiveWeapon.Instance).

Подієво-орієнтована взаємодія — зв'язок між компонентами організовано через стандартні події C# (EventHandler). Зокрема, OnPlayerDeath,

OnEnemyAttack та OnFlashBlink дозволяють підписувати обробники без прямого зв'язування класів між собою.

Скінченний автомат станів (FSM) — поведінка ворога-скелета описана через перелічення State з чотирма активними станами: Roaming, Chasing, Attacking та Death. Переходи між станами визначаються відстанню до гравця та його статусом. Такий підхід забезпечує передбачувану та розширювану поведінку ворогів.

Алгоритм навігації на основі NavMesh — для переміщення ворога по рівню використовується NavMeshAgent з підключеним пакетом NavMeshComponents для двовимірного простору. Агент автоматично прокладає шлях з обходом перешкод, а швидкість динамічно змінюється залежно від поточного стану: у режимі патрулювання використовується базова швидкість, у режимі переслідування вона множиться на коефіцієнт chasingSpeedMultiplier.

Система виявлення зіткнень — бойова взаємодія реалізована через PolygonCollider2D на об'єкті меча та OnTriggerEnter2D на компоненті Sword. Колайдер атаки вмикається лише на момент удару та одразу вимикається, що унеможливлює повторне нанесення пошкодження одним замахом. Ворог наносить пошкодження гравцеві через OnTriggerStay2D у компоненті EnemyEntity, що моделює контактну шкоду за перебування в зоні ураження.

Система анімацій — переходи між анімаційними станами керуються через Animator та булеві параметри (isRunning, isDie). Для гравця та ворога-скелета реалізовані окремі Animator Controller з відповідними кліпами. Анімації меча (SwordSlashDown, SwordSlashUp, SwordSwingDown, SwordSwingUp) виділені в окремі контролери для зручності керування.

Візуальний зворотний зв'язок — при отриманні пошкодження активується матеріал FlashBlink, що короткочасно змінює колір спрайта. Це реалізовано через корутину в компоненті FlashBlink і підписку на подію OnFlashBlink від класу Player.

Сукупність обраних засобів і підходів забезпечила реалізацію повноцінного ігрового застосунку з чіткою архітектурою, стабільною

поведінкою об'єктів та достатнім рівнем гнучкості для подальшого розвитку проєкту.

РОЗДІЛ 3

РОЗРОБКА ДВОВИМІРНОЇ ГРИ З ВИКОРИСТАННЯМ UNITY

3.1 Практична реалізація ігрових механік та інтерфейсу

У ході розробки застосунку «Adventurous World» було побудовано повноцінну структуру двовимірної рольової гри, що охоплює систему керування персонажем, бойові взаємодії, поведінку ворогів на основі штучного інтелекту та елементи користувацького інтерфейсу. Ключовою метою реалізації стало створення середовища, у якому гравець може вільно досліджувати ігровий світ, вступати в бій із ворогами та отримувати чіткий візуальний зворотний зв'язок на кожну свою дію.

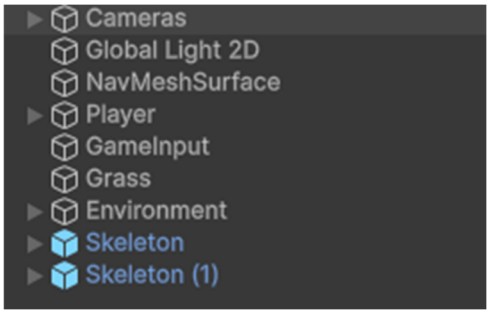

Рисунок 3.1 – Кінцевий вигляд ієрархії гри

Переміщення персонажа реалізовано через нову систему введення Unity Input System у поєднанні з фізичним компонентом Rigidbody2D. Гравець має змогу рухатись у восьми напрямках, а система анімацій динамічно реагує на зміну вектора введення: при русі активується анімація ходьби, при зупинці — повернення до стану спокою. Блокування переміщення під час ефекту відкидання реалізовано через перевірку прапора IsGettingKnockedBack у класі KnockBack, що унеможливлює некоректне поєднання фізичного імпульсу з ручним керуванням.

Система отримання пошкоджень побудована на взаємодії кількох компонентів. При контакті ворога з гравцем викликається метод TakeDamage,

який зменшує поточне здоров'я, запускає ефект відкидання через KnockBack.GetKnockedBack, активує короткочасний імунітет через корутину DamageRecoveryRoutine та ініціює візуальний спалах через подію OnFlashBlink

Такий ланцюжок подій забезпечує природну реакцію персонажа на удар і дає гравцеві достатньо часу для відповіді.

Системі анімацій приділено особливу увагу, оскільки вона відіграє роль не лише декоративного елементу, а й важливого інструменту зворотного зв'язку. Для персонажа гравця реалізовано три анімаційні стани через Animator Controller: спокій (Idle), біг (Running) та загибель (Death). Перемикання між ними відбувається автоматично на основі булевих параметрів isRunning та isDie, що змінюються в класі PlayerVisual залежно від поточного стану гравця.

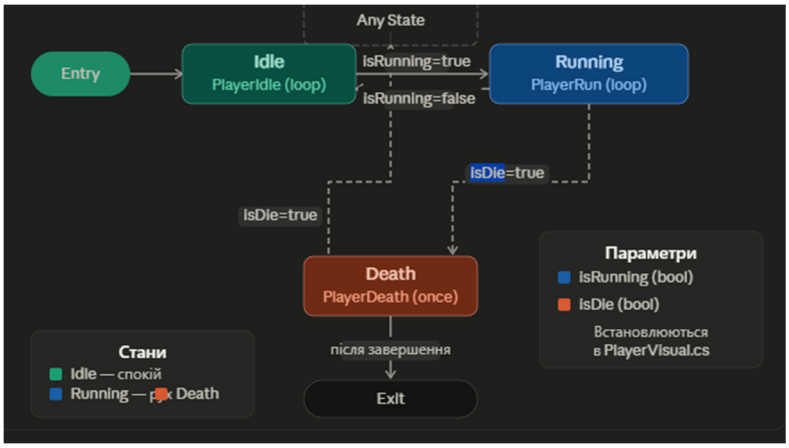

Рисунок 3.2 – Вигляд аніматора PlayerVisual

Синхронізацію анімацій із бойовою логікою забезпечено через систему подій C#. Зокрема, подія OnPlayerDeath сповіщає підписані компоненти про загибель персонажа: PlayerVisual активує анімацію смерті, а система керування блокує подальше введення. Це відповідає принципу слабкої зв'язаності — жоден клас не звертається до іншого напряму, а лише реагує на публічні події.

Орієнтація персонажа реалізована через порівняння екранної позиції курсора з позицією гравця у методі AdjustPlayerFacingDirection класу PlayerVisual. При зміщенні курсора вліво від персонажа спрайт відзеркалюється через властивість SpriteRenderer.flip□. Це забезпечує інтуїтивне відображення

напрямку атаки та логічно пов'язує бойову механіку з орієнтацією героя в просторі.

Для ворога-скелета реалізовано окремий Animator Controller з анімаційними станами: Idle, Roaming, Attacking, TakeHit та Death. Перемикання між станами відбувається автоматично в класі SkeletonVisual на основі підписки на події від EnemyAI та EnemyEntity. Такий підхід дозволяє чітко розмежувати логіку поведінки та візуальне представлення ворога.

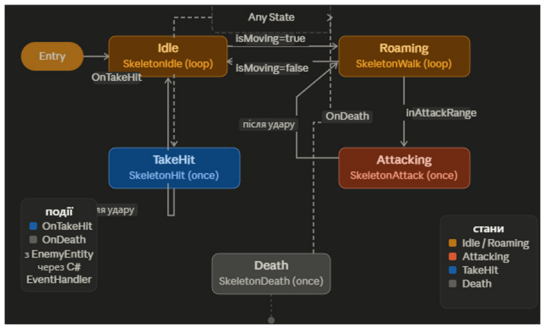

Рисунок 3.3 – Вигляд аніматора SkeletonVisual

Поведінка ворога-скелета побудована на скінченному автоматі станів у класі EnemyAI. Перелічення State містить значення Idle, Roaming, Chasing, Attacking та Death. У стані Roaming ворог обирає випадкову точку в межах заданого радіусу та рухається до неї через NavMeshAgent. При виявленні гравця в зоні chaseRadius відбувається перехід у стан Chasing із збільшенням швидкості агента через множник chasingSpeedMultiplier. При наближенні до дистанції attackRadius ворог зупиняється та переходить до фази атаки, яка ініціюється через подію OnEnemyAttack.

Навігацію ворогів по ігровій карті забезпечено засобами пакету NavMeshComponents для двовимірного простору. На підготовчому етапі поверхні сцени було розмічено: прохідні зони позначено як Walkable, а перешкоди — стіни, водні об'єкти та декоративні елементи — як Not Walkable. Кожен ворог має компонент NavMeshAgent, який автоматично прокладає маршрут до цільової позиції та оновлює його при зміні положення гравця. Завдяки цьому вороги коректно обходять перешкоди та не проходять крізь непрохідні об'єкти.

Для побудови ігрових локацій застосовано систему Tilemap із кількома тематичними палітрами тайлів. Кожна палітра відповідає окремому типу поверхні: трав'яний покрив, земля, водні об'єкти та декоративні елементи. Колізії непрохідних об'єктів оброблено через TilemapCollider2D у поєднанні з CompositeCollider2D для оптимізації фізичних обчислень. Це дозволяє гравцеві інтуїтивно розуміти межі прохідних зон без додаткових підказок

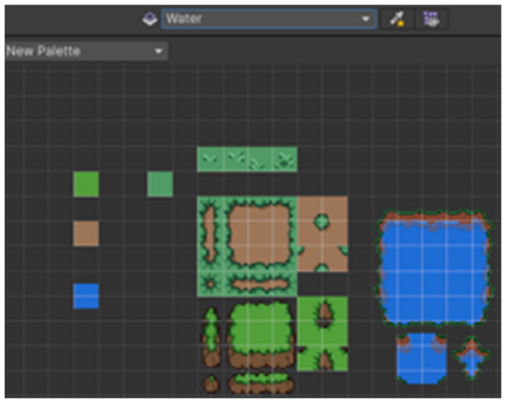

Рисунок 3.4 – Tile Palette

Система анімацій меча виділена в окремі контролери: SwordSlashDown, SwordSlashUp, SwordSwingDown та SwordSwingUp. Активна зброя обробляється через клас ActiveWeapon (Singleton), який зберігає посилання на поточний об'єкт зброї та надає доступ до нього з інших систем. Атака реєструється в методі OnTriggerEnter2D класу Sword: при вході ворога в зону PolygonCollider2D меча викликається метод TakeDamage компонента EnemyEntity. Колайдер атаки вмикається лише на момент удару та одразу деактивується, що унеможливлює повторне пошкодження одним замахом.

Для стеження камери за персонажем використано пакет Cinemachine. Віртуальна камера налаштована на автоматичне відстеження об'єкта гравця зі згладженням переміщення, що забезпечує плавний та природний рух без ривків. Параметри мертвої зони та затримки відстеження підібрано таким чином, щоб камера не реагувала на дрібні коливання позиції персонажа.

Систему переходів між сценами реалізовано через тригерні зони: при входженні гравця в зону переходу запускається ефект затемнення екрана (fade),

після чого завантажується нова сцена, а персонаж розміщується у відповідній точці входу. Це забезпечує плавну зміну локацій без візуальних стрибків.

3.2 Структура ігрових ассетів та програмного коду

Графічну основу застосунку складають двовимірні спрайти у форматі PNG із підтримкою прозорості. Усі графічні ресурси імпортовані та налаштовані вручну: для кожного спрайта визначено тип фільтрації, режим компресії та параметри нарізки для спрайтових аркушів. Використання піксельної графіки з єдиною колірною палітрою забезпечує стилістичну цілісність усіх ігрових елементів.

Анімовані об'єкти — персонаж гравця та ворог-скелет — керуються через Animator та Animation Clip. Кожен анімаційний кліп є окремою послідовністю спрайтів, що відтворюється циклічно або одноразово залежно від налаштувань. Для ефекту візуального спалаху при отриманні пошкодження застосовано окремий матеріал шейдера, що короткочасно змінює колір спрайта без використання додаткових анімаційних кліпів.

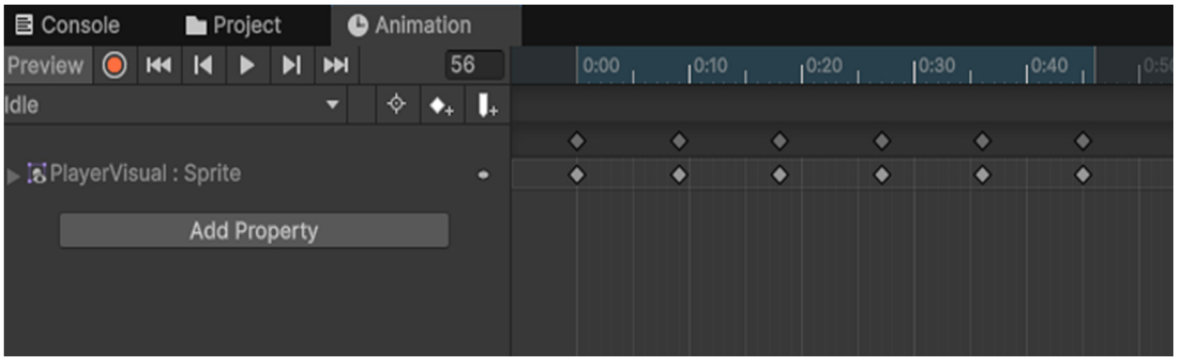

Рисунок 3.5 – Налаштування анімації для PlayerVisual

Ресурси проєкту організовано в тематичні директорії з чіткими іменами, що відповідають їхньому вмісту. Папка Animations містить усі анімаційні кліпи та контролери. У директорії Prefabs зберігаються заздалегідь підготовлені шаблони об'єктів: персонаж гравця, ворог-скелет, зброя та точки переходу між сценами. Папка Scripts розбита на підкаталоги відповідно до функціональної належності: Player, Skeleton, Weapon, Other. Графічні ресурси розміщено у Sprites, тайли та палітри — у Tilemap, матеріали шейдерів — у Materials.

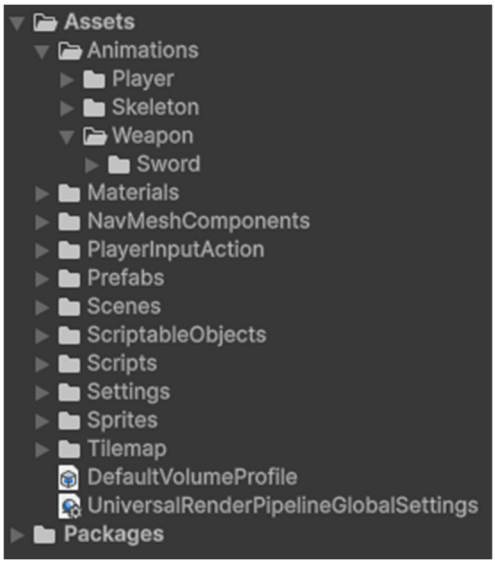

Рисунок 3.6 – Повна структура папок Assets

Кодова база побудована відповідно до принципів об'єктно-орієнтованого програмування із чітким розмежуванням відповідальності між класами. Глобальні системи (Player, GameInput, ActiveWeapon) реалізовано за патерном Singleton, що забезпечує уніфікований доступ до них з будь-якого компонента без необхідності передавати посилання вручну. Взаємодія між непов'язаними системами організована виключно через події C# (EventHandler), що відповідає принципу слабкої зв'язаності та спрощує подальше масштабування.

Компонент KnockBack є прикладом повторно використовуваної логіки: він реалізований як окремий скрипт, що підключається до будь-якого об'єкта, який має отримувати фізичний імпульс. Це унеможливлює дублювання однакової логіки в різних класах і дозволяє легко застосувати механіку відкидання до нових типів ворогів без змін у базовому коді.

Весь програмний код написано мовою C# у середовищі Visual Studio з підтримкою IntelliSense, що прискорює навігацію між класами та зменшує кількість синтаксичних помилок. Інтеграція Visual Studio з Unity Editor дозволяє встановлювати точки зупинки та відстежувати значення змінних у реальному часі під час відтворення гри.

3.3 Тестування та виправлення помилок

Завершальним етапом розробки «Adventurous World» стало комплексне тестування усіх реалізованих механік. Перевірка проводилась як безпосередньо в Unity Editor у режимі відтворення, так і у зібраній версії застосунку під платформу Windows. Це дозволило виявити помилки обох типів — логічні, що проявляються в редакторі, та технічні, що виникають лише у скомпільованій збірці. Оскільки проєкт розроблявся одноосібно, основним методом перевірки було ручне тестування кожного модуля гри в дії.

Першим кроком перевірялась система керування персонажем. Тестувались переміщення у всіх восьми напрямках, коректність блокування руху під час відкидання та правильність перемикання анімацій. Було виявлено, що при активному ефекті відкидання гравець міг частково зберігати керування, що порушувало фізичний імпульс. Помилку усунено шляхом додавання перевірки IsGettingKnockedBack на початку FixedUpdate — при активному відкиданні метод переміщення не викликається (лістинг 3.1).

Лістинг 3.1 – Блокування руху під час відкидання

```
private void FixedUpdate() {
    if (_knockBack.IsGettingKnockedBack)
        return;
    HandleMovement();
}
```

кінець лістингу 3.1

Тестування механіки ривка виявило, що при швидкому повторному натисканні клавіші ривка ефект міг активуватись до завершення кулдауну. Для усунення цього в методі Dash додано перевірку прапора _isDashing — новий ривок запускається лише якщо попередній завершився (лістинг 3.2).

Лістинг 3.2 – Захист від повторної активації ривка

```
private void Dash() {
    if (!_isDashing)
        StartCoroutine(DashRoutine());
}
private IEnumerator DashRoutine() {
```

```
    _isDashing = true;
    movingSpeed *= dashSpeed;
    trailRenderer.emitting = true;
    yield return new WaitForSeconds(dashTime);

    trailRenderer.emitting = false;
    movingSpeed = _initialMovingSpeed;

    yield return new WaitForSeconds(dashCoolDownTime);
    _isDashing = false;
}
```

кінець лістингу 3.2

Бойова система потребувала окремої перевірки коректності виявлення зіткнень. При певному розташуванні ворога відносно персонажа атака мечем могла не реєструватись. Проблему вирішено через коректне керування станом PolygonCollider2D — колайдер вимикається та одразу вмикається при кожній атаці, що гарантує реєстрацію нового зіткнення (лістинг 3.3).

Лістинг 3.3 – Активація колайдера атаки меча

```
private void Dash() {
    if (!_isDashing)
        StartCoroutine(DashRoutine());
}
private IEnumerator DashRoutine() {
    _isDashing = true;
    movingSpeed *= dashSpeed;
    trailRenderer.emitting = true;
    yield return new WaitForSeconds(dashTime);
    trailRenderer.emitting = false;
    movingSpeed = _initialMovingSpeed;
    yield return new WaitForSeconds(dashCoolDownTime);
    _isDashing = false;
}
```

кінець лістингу 3.3

Під час тестування системи пошкоджень перевірявся весь ланцюжок: отримання удару → зменшення здоров'я → відкидання → тимчасовий імунітет

→ виявлення загибелі. Було виявлено, що без перевірки прапора _canTakeDamage ворог міг завдати кількох пошкоджень майже одночасно через OnTriggerStay2D. Захист реалізовано через встановлення _canTakeDamage =false одразу при вході в TakeDamage та відновлення через корутину (лістинг 3.4).

Лістинг 3.4 – Активація колайдера атаки меча

```
private void Dash() {
    if (!_isDashing)
        StartCoroutine(DashRoutine());
}
private IEnumerator DashRoutine() {
    _isDashing = true;
    movingSpeed *= dashSpeed;
    trailRenderer.emitting = true;
    yield return new WaitForSeconds(dashTime);
    trailRenderer.emitting = false;
    movingSpeed = _initialMovingSpeed;
    yield return new WaitForSeconds(dashCoolDownTime);
    _isDashing = false;
}
```

кінець лістингу 3.4

У процесі налагодження застосовувались такі методи: виведення діагностичних повідомлень через Debug.Log для відстеження значень здоров'я та поточного стану у реальному часі; візуалізація зон виявлення через OnDrawGizmosSelected для перевірки радіусів chasingDistance та attackingDistance; покрокове виконання коду через точки зупину у Visual Studio для аналізу переходів між станами FSM; виведення поточного стану ворога на екран у текстовому вигляді для перевірки коректності ланцюжка Idle → Roaming → Chasing → Attacking → Death.

За результатами тестування всі виявлені помилки було усунено. Застосунок демонструє стабільну частоту кадрів, коректну бойову логіку, передбачувану поведінку ворогів та правильне відображення інтерфейсу. Модульна архітектура на основі подій C# та патерну Singleton забезпечує можливість подальшого розширення проєкту без суттєвих змін у наявному коді.

ВИСНОВКИ

У межах виконання кваліфікаційної роботи бакалавра було успішно спроєктовано та реалізовано повноцінний двовимірний ігровий застосунок «Adventurous World» у жанрі Adventure-RPG. Розробка здійснювалась із застосуванням рушія Unity та об'єктно-орієнтованої мови програмування C#. Створений продукт поєднує механіки бойової взаємодії ближнього бою, переміщення з підтримкою ривка, поведінку ворогів на основі штучного інтелекту, систему переходів між сценами та модульну архітектуру, що забезпечує зручне масштабування проєкту.

Під час аналізу предметної області обґрунтовано актуальність обраної тематики: незалежна розробка 2D-ігор залишається затребуваним напрямом як у комерційному, так і в освітньому середовищі. Проведено огляд аналогічних проєктів — «Hollow Knight», «Celeste», «Dead Cells» — та виокремлено ключові архітектурні й геймдизайнерські підходи, що знайшли відображення в розробленому застосунку. Зазначено, що проєкт може слугувати інструментом психологічного розвантаження, зокрема для дітей-переселенців, яким інтерактивне середовище допомагає адаптуватися до нових умов.

Проведення технічного аналізу дало змогу сформулювати функціональні та нефункціональні вимоги до застосунку, побудувати модульну архітектуру із чітким розмежуванням відповідальності між класами та описати логіку взаємодії між компонентами. Розроблено структурну діаграму основних класів проєкту, що відображає зв'язки між системами гравця, ворога та зброї.

Обґрунтовано вибір технологічного стеку: рушій Unity обрано за підтримку двовимірної фізики на основі Box2D, системи анімацій Animator, тайлових карт Tilemap та навігаційних сіток NavMeshComponents. Мова C# забезпечила стабільність, читабельність коду та зручну реалізацію подієво-орієнтованої взаємодії через EventHandler. Розробка велась у середовищі Visual Studio з підтримкою IntelliSense та вбудованим відлагоджувачем.

Реалізовано всі основні компоненти застосунку: систему керування персонажем із підтримкою восьминапрямленого переміщення та механікою ривка через TrailRenderer; бойову систему на основі PolygonCollider2D з коректним увімкненням зони ураження при кожній атаці; компонент фізичного відкидання KnockBack із імпульсною силою Rigidbody2D; логіку ворога-скелета зі скінченним автоматом станів (Idle, Roaming, Chasing, Attacking, Death) та навігацією через NavMeshAgent; систему анімацій для гравця та ворога з окремими контролерами; візуальний ефект спалаху FlashBlink через матеріал шейдера; централізовану обробку введення через Unity Input System у класі GameInput.

Для забезпечення глобального доступу до ключових систем застосовано патерн «Одинак» (Singleton) у класах Player, GameInput та ActiveWeapon. Взаємодію між непов'язаними компонентами організовано через події C# — OnPlayerDeath, OnFlashBlink, OnEnemyAttack, OnTakeHit, OnDeath — що відповідає принципу слабкої зв'язаності та спрощує подальше розширення проєкту.

Процес тестування охопив ручну функціональну перевірку всіх механік, аналіз переходів між станами FSM, коректність бойових взаємодій та стабільність роботи NavMeshAgent. Виявлені помилки — некоректне збереження керування під час відкидання, повторна активація ривка до завершення кулдауну, хаотична зміна орієнтації ворога при мінімальній різниці координат, продовження переслідування після загибелі гравця — були усунені шляхом внесення цільових змін до відповідних методів.

Візуальну складову застосунку побудовано на основі піксельних спрайтів у форматі PNG, організованих в анімаційні послідовності через Animation Clip та Animator Controller. Ігрові локації сформовано засобами Tilemap із кількома тематичними палітрами. Інтерфейс адаптовано до різних роздільностей екрана через Canvas Scaler із режимом Scale With Screen Size.

Результати роботи повністю відповідають завданням, сформульованим у першому розділі. Реалізовано структуру сцен, логіку персонажа, поведінку

ворогів, систему зброї, анімації та переходи між локаціями. Увесь програмний код написано самостійно без використання сторонніх шаблонів.

Розроблений застосунок має потенціал для подальшого розширення: додавання нових типів ворогів із відмінною поведінкою, системи інвентаря та предметів, механіки прокачування персонажа, а також підтримки мобільної платформи через перенесення введення на сенсорний екран. Проєкт може слугувати демонстраційним матеріалом на курсах із Unity або програмування ігор, а також стартовою основою для подальших студентських розробок.

Таким чином, усі поставлені завдання кваліфікаційної роботи виконано в повному обсязі. «Adventurous World» є завершеним та працездатним ігровим застосунком, що демонструє практичне застосування сучасних підходів до розробки програмного забезпечення й має перспективу подальшого розвитку як навчального та розважального продукту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- Unity Technologies. Unity Scripting API Reference. URL: https://docs.unity3d.com/ScriptReference/ (дата звернення: 04.02.2026).
- Microsoft. C# Language Documentation. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 04.02.2025).
- Microsoft. Visual Studio IDE. URL: https://visualstudio.microsoft.com (дата звернення: 10.02.2026).
- Unity Technologies. Unity Hub & Engine Download. URL: https://unity.com/download (дата звернення: 10.02.2026).
- GameDev.tv. Unity 2D Game Development Course. URL: https://www.gamedev.tv/p/unity-2d-game-development (дата звернення: 14.02.2026).
- Unity Technologies. Unity Asset Store. URL: https://assetstore.unity.com/ (дата звернення: 28.02.2026).
- Kenney. Free Game Assets & Art. URL: https://kenney.nl/assets (дата звернення: 05.03.2026).
- Itch.io. Pixel Art Game Assets. URL: https://itch.io/game-assets/tag-pixel-art (дата звернення: 10.02.2026).
- Catlike Coding. Unity Tutorials. URL: https://catlikecoding.com/unity/tutorials/ (дата звернення: 10.05.2026).
- Ray Wenderlich. Unity 2D Tutorials & Resources. URL: https://www.raywenderlich.com/unity (дата звернення: 04.03.2023).
- GitHub. Unity 2D Game Projects. URL: https://github.com/search?q=unity+2d+game&type (дата звернення: 04.05.2025).
- Game Developer (Gamasutra). URL: https://www.gamedeveloper.com (дата звернення: 21.04.2023).