

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
ДЕРЖАВНИЙ ЗАКЛАД "ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ  
УНІВЕРСИТЕТ  
ІМЕНІ ТАРАСА ШЕВЧЕНКА"

Факультет технологій та інформаційних систем

Ємел'янов Дмитро Сергійович

**РОЗРОБКА ІНТЕРФЕЙСУ ТА ФУНКЦІОНАЛЬНИХ  
КОМПОНЕНТІВ ІНТЕРАКТИВНОГО ПРОГРАМНОГО ПРОДУКТУ**

**кваліфікаційна робота  
здобувача вищої освіти першого (бакалаврського) рівня  
освітньої програми «Комп'ютерні науки та інформаційні  
технології»  
за спеціальністю 122 Комп'ютерні науки**

Особистий підпис \_\_\_\_\_



Науковий керівник \_\_\_\_\_



Владислав КОЗУБ, д-р філософії

Завідувач кафедри \_\_\_\_\_

Микола СЕМЕНОВ, к.пед.н., доцент

Лубни – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
ДЕРЖАВНИЙ ЗАКЛАД "ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ  
УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА"

Факультет технологій та інформаційних систем

**Кафедра** інформаційних технологій та систем

**Освітній рівень** бакалавр

**Напрямок підготовки (спеціальність)** 122 "Комп'ютерні науки"

**Галузь знань** 12 "Інформаційні технології"

ЗАТВЕРДЖУЮ

Завідувач кафедри ІТС

\_\_\_\_\_ (підпис)

Миколай Семенов

"\_\_\_\_\_" \_\_\_\_\_ 2026 р.

**ЗАВДАННЯ**

**НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Ємел'янов Дмитро Сергійович

**1. Тема:** Розробка інтерфейсу та функціональних компонентів інтерактивного програмного продукту

**Керівник кваліфікаційної роботи:** Владислав КОЗУБ, доктор філософії кафедри інформаційних технологій та систем

затверджена наказом по університету від "\_\_\_\_\_" \_\_\_\_\_ 2026 року №

**2. Строк подання здобувачем вищої освіти проєкту:** "13" травня 2026 р.

**3. Вихідні дані до роботи:** Сучасні технології розробки інтерактивних тривимірних програмних продуктів; середовище розробки Unity 3D; мова програмування C#; компонент CharacterController для реалізації системи керування персонажем; система навігації NavMesh та компонент NavMeshAgent для реалізації штучного інтелекту; система Input System для обробки введення користувача; механізм Raycast для реалізації взаємодії з об'єктами; система Canvas для побудови користувацького інтерфейсу; формат JSON для серіалізації та збереження даних;

**4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):** Аналіз предметної області та існуючих рішень у сфері інтерактивних програмних продуктів; огляд та порівняльний аналіз інструментів створення тривимірних застосунків; дослідження принципів побудови користувацького інтерфейсу; визначення недоліків існуючих рішень; формування функціональних та нефункціональних вимог до програмного забезпечення; проєктування загальної архітектури програмного продукту; проєктування користувацького інтерфейсу з урахуванням принципів UI/UX-дизайну; реалізація системи керування персонажем у середовищі Unity 3D; розробка системи штучного інтелекту на основі кінцевого автомата станів; реалізація системи взаємодії з об'єктами сцени; розробка системи збереження та завантаження прогресу; тестування та оцінка продуктивності програмного продукту; порівняння з існуючими аналогами; висновки.

**КАЛЕНДАРНИЙ ПЛАН-ГРАФІК  
ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ**

освітнього ступеня «бакалавр»

зі спеціальності 122 «Комп'ютерні науки»

студента Ємел'янова Дмитра Сергійовича

Тема: Розробка інтерфейсу та функціональних компонентів  
інтерактивного програмного продукту

Науковий керівник: В.Ю. Козуб, канд. техн. наук, доктор філософії

| з/п | Назви робіт                                                                                                                     | Дата       | Примітки |
|-----|---------------------------------------------------------------------------------------------------------------------------------|------------|----------|
| 1   | Аналіз предметної області: огляд сучасних інтерактивних програмних продуктів та підходів до їх розробки                         | 16.02.2026 | Виконано |
|     | Порівняльний аналіз інструментів створення тривимірних застосунків та дослідження принципів побудови користувацького інтерфейсу | 23.02.2026 | Виконано |
|     | Проектний етап: розробка загальної архітектури системи (Use Case, Class, Sequence, State diagrams)                              | 02.03.2026 | Виконано |
|     | Проектування користувацького інтерфейсу та основних функціональних компонентів системи                                          | 16.03.2026 | Виконано |
|     | Технічна реалізація системи керування персонажем та системи штучного інтелекту в середовищі Unity 3D                            | 23.03.2026 | Виконано |
|     | Розробка системи взаємодії з об'єктами, системи збереження даних та модуля керування станами гри                                | 30.03.2026 | Виконано |
|     | Тестування програмного продукту, оцінка продуктивності та порівняння результатів з аналогами                                    | 06.04.2026 | Виконано |
|     | Документація, підготовка висновків, фінальний звіт                                                                              | 13.04.2026 | Виконано |

Науковий керівник:

 В.Ю. Козуб

Здобувач вищої освіти:

 Д.С. Ємел'янов

## ЗМІСТ

|                                                                                                                             |    |
|-----------------------------------------------------------------------------------------------------------------------------|----|
| <b>АНОТАЦІЯ</b> .....                                                                                                       | 5  |
| <b>ABSTRACT</b> .....                                                                                                       | 7  |
| <b>ВСТУП</b> .....                                                                                                          | 1  |
| <b>РОЗДІЛ 1</b> .....                                                                                                       | 5  |
| <b>АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ВИМОГ ДО ІНТЕРФЕЙСУ ТА ФУНКЦІОНАЛЬНИХ КОМПОНЕНТІВ ІНТЕРАКТИВНИХ ПРОГРАМНИХ ПРОДУКТІВ</b> ..... | 5  |
| <b>1.1. Порівняльний аналіз сучасних 3D-рушіїв</b> .....                                                                    | 5  |
| <b>1.2. Можливості Unity 3D</b> .....                                                                                       | 7  |
| <b>1.3. Аналіз підходів до створення інтерфейсу та функціональних компонентів</b> .....                                     | 7  |
| <b>РОЗДІЛ 2</b> .....                                                                                                       | 16 |
| <b>ПРОЄКТУВАННЯ АРХІТЕКТУРИ ІНТЕРФЕЙСУ ТА ФУНКЦІОНАЛЬНИХ КОМПОНЕНТІВ</b> .....                                              | 16 |
| 2.1. Загальна архітектура програмного продукту .....                                                                        | 16 |
| 2.2. Проєктування користувацького інтерфейсу .....                                                                          | 19 |
| 2.3. Проєктування основних функціональних компонентів .....                                                                 | 22 |
| 2.4. UML-моделювання архітектури системи .....                                                                              | 25 |
| <b>РОЗДІЛ 3</b> .....                                                                                                       | 28 |
| <b>РЕАЛІЗАЦІЯ ІНТЕРФЕЙСУ ТА ОСНОВНИХ ФУНКЦІОНАЛЬНИХ КОМПОНЕНТІВ</b> .....                                                   | 28 |
| <b>3.1. Реалізація користувацького інтерфейсу</b> .....                                                                     | 28 |
| 3.2. Реалізація компонента керування персонажем .....                                                                       | 30 |
| 3.3. Реалізація системи штучного інтелекту ворогів .....                                                                    | 35 |
| <b>РОЗДІЛ 4</b> .....                                                                                                       | 41 |
| <b>ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ПРОГРАМНОГО ПРОДУКТУ</b> .....                                                         | 41 |
| 4.1. Організація процесу тестування.....                                                                                    | 41 |
| 4.2. Аналіз результатів тестування та оцінка ефективності системи .....                                                     | 44 |
| 4.3. Перспективи розвитку та вдосконалення програмного продукту.....                                                        | 46 |
| 4.4. Висновки до розділу .....                                                                                              | 50 |
| <b>ВИСНОВКИ</b> .....                                                                                                       | 52 |
| <b>СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ</b> .....                                                                                     | 57 |

## АНОТАЦІЯ

Ємел'янов Дмитро Сергійович. Розробка інтерфейсу та функціональних компонентів інтерактивного програмного продукту (на прикладі тривимірної ігрової сцени в середовищі Unity 3D). Кваліфікаційна робота бакалавра. Спеціальність 122 «Комп'ютерні науки». ДЗ «Луганський національний університет імені Тараса Шевченка», 2026. – 67 с., 2 рис., 1 табл., 2 додатків, 18 джерел.

Об'єкт дослідження – інтерфейс та функціональні компоненти інтерактивного програмного продукту. Предмет дослідження – архітектура користувацького інтерфейсу та основних функціональних компонентів (керування, штучний інтелект, система збереження прогресу) інтерактивного програмного продукту, реалізованого в середовищі Unity 3D.

Мета роботи – розробити сучасний користувацький інтерфейс та функціональні компоненти інтерактивного програмного продукту на прикладі тривимірної ігрової сцени з механіками виживання.

Результати роботи. Розроблено повноцінний інтерактивний програмний продукт у вигляді 3D-ігрової сцени SampleScene (Unity 6.3 LTS). Реалізовано:

сучасний користувацький інтерфейс (Canvas, Slider здоров'я, динамічні підказки, екрани Game Over / Victory);

функціональні компоненти керування персонажем від першої особи (CharacterController + Input System);

штучний інтелект ворогів (NavMeshAgent + система виявлення гравця);

систему взаємодії з об'єктами (Raycast-підбір 5 предметів);

стабільну систему збереження та завантаження прогресу (JSON-серіалізація + Application.persistentDataPath) з коректним відновленням позиції, здоров'я та зібраних предметів.

Проведено тестування продуктивності та зручності інтерфейсу. Розроблений продукт може використовуватися як шаблон для створення інтерактивних програмних рішень.

У результаті виконання роботи створено сучасний, зручний та функціонально повний інтерфейс інтерактивного програмного продукту, що відповідає сучасним вимогам до користувацького досвіду та технічної реалізації.

Ключові слова: ІНТЕРФЕЙС КОРИСТУВАЧА, ФУНКЦІОНАЛЬНІ КОМПОНЕНТИ, UNITY 3D, CHARACTERCONTROLLER, NAVMESHAGENT, JSON-СЕРІАЛІЗАЦІЯ, ІНТЕРАКТИВНИЙ ПРОГРАМНИЙ ПРОДУКТ, ШТУЧНИЙ ІНТЕЛЕКТ.

## ABSTRACT

Yemelyanov Dmytro Oleksandrovych. Development of the interface and functional components of an interactive software product (on the example of a 3D game scene in the Unity 3D environment). Bachelor's qualification work. Specialty 122 "Computer Science". Luhansk National University named after Taras Shevchenko, 2025. – 67 p., 2 fig., 1 tabl., 2 appendices, 18 references.

Object of research – the interface and functional components of an interactive software product. Subject of research – the architecture of the user interface and the main functional components (control, artificial intelligence, progress saving system) of an interactive software product implemented in the Unity 3D environment.

Purpose of the study – to develop a modern user interface and functional components of an interactive software product using the example of a 3D game scene with survival mechanics.

A complete interactive software product in the form of the SampleScene 3D game scene (Unity 6.3 LTS) has been developed. The following have been implemented: modern user interface (Canvas, health Slider, dynamic prompts, Game Over / Victory screens); first-person character control components (CharacterController + Input System); enemy artificial intelligence (NavMeshAgent + detection system); object interaction system (Raycast item pickup); stable progress saving and loading system (JSON serialization + Application.persistentDataPath).

Performance and interface usability testing has been carried out. The developed product can be used as a template for creating interactive software solutions.

Conclusions. As a result of the work, a modern, convenient and functionally complete user interface of an interactive software product that meets current requirements for user experience and technical implementation has been created.

Keywords: USER INTERFACE, FUNCTIONAL COMPONENTS, UNITY 3D, CHARACTERCONTROLLER, NAVMESHAGENT, JSON SERIALIZATION, INTERACTIVE SOFTWARE PRODUCT, ARTIFICIAL INTELLIGENC

## ВСТУП

Сучасний етап розвитку інформаційного суспільства характеризується стрімким зростанням ролі програмного забезпечення у всіх сферах діяльності людини. Інтерактивні програмні продукти стали невід'ємною складовою освітніх, розважальних та виробничих. Вони забезпечують ефективну взаємодію між користувачем і комп'ютерною системою, дозволяють візуалізувати складні процеси, автоматизувати діяльність та підвищувати продуктивність праці.

Особливе місце серед інтерактивних систем займають тривимірні додатки, які використовують сучасні графічні технології, моделювання фізичних процесів та елементи штучного інтелекту. Такі системи широко застосовуються у відеоіграх, симуляторах, віртуальній та доповненій реальності, навчальних середовищах, системах підготовки фахівців[13]. Вони дозволяють створювати високий рівень занурення користувача у цифрове середовище та забезпечують інтуїтивну взаємодію з об'єктами. При розробці додатків ключову роль відіграють два основних компоненти: користувацький інтерфейс та функціональна частина. Користувацький інтерфейс визначає спосіб взаємодії користувача з програмою, впливає на зручність використання, швидкість освоєння та загальне враження від продукту. В той час як функціональні компоненти забезпечують логіку роботи системи, її стабільність, продуктивність та можливість масштабування.

Зростання вимог до якості програмного забезпечення призводить до необхідності розробки інтерфейсів, які є не лише візуально привабливими, але й ергономічними й інтуїтивно зрозумілими. Невдалий дизайн інтерфейсу або нестабільна робота функціональних компонентів можуть значно знизити ефективність використання програмного продукту або навіть призвести до відмови користувачів від його використання[17].

Актуальність теми полягає у необхідності дослідження та розробки сучасних підходів до створення інтерфейсу та функціональних компонентів



інтерактивного програмного продукту, що забезпечують високий рівень зручності, ефективності та надійності. Особливої уваги потребує розробка таких систем у середовищах, які дозволяють швидко створювати прототипи та реалізовувати складну логіку взаємодії, зокрема у середовищі Unity 3D.

Unity 3D є одним із популярних інструментів для розробки програмних продуктів. Його популярність зумовлена широкими можливостями, кросплатформеністю, наявністю великої кількості готових компонентів та активною спільнотою розробників. Використання цього середовища розробки дозволяє реалізовувати складні функціональні системи, включаючи керування персонажем, впровадження штучного інтелекту, фізичних взаємодій та системи збереження даних. Процес розробки додатку потребує комплексного підходу, який включає аналіз існуючих рішень, формулювання вимог, проєктування архітектури системи, реалізацію функціональних компонентів та тестування отриманого результату. Особливо важливим є врахування принципів об'єктно-орієнтованого програмування, модульності, повторного використання коду та масштабованості системи[2]. Крім того, Також значну роль відіграє застосування методів системного аналізу та моделювання, що дозволяють формалізувати структуру програмного продукту, визначити взаємозв'язки між його компонентами та оптимізувати процес розробки. Використання UML-діаграм, структурного та поведінкового моделювання сприяє кращому розумінню архітектури системи та полегшує її подальше вдосконалення.

Метою кваліфікаційної роботи є розробка сучасного користувацького інтерфейсу та функціональних компонентів інтерактивного програмного продукту на прикладі тривимірної ігрової сцени в середовищі Unity 3D.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати сучасні інтерактивні програмні продукти та підходи до їх розробки.
- Здійснити порівняльний аналіз інструментів створення тривимірних застосунків.

- Дослідити принципи побудови користувацького інтерфейсу.
- Сформулювати вимоги до інтерфейсу та функціональних компонентів.
- Спроектувати архітектуру програмного продукту із використанням UML-моделювання.
- Реалізувати основні функціональні компоненти, зокрема систему керування персонажем, штучний інтелект, систему взаємодії з об'єктами.
- Розробити систему збереження та завантаження прогресу.
- Провести тестування продуктивності та зручності використання програмного продукту.
- Виконати аналіз отриманих результатів та сформулювати висновки.

**Об'єктом дослідження** є інтерактивні програмні продукти та процес їх створення.

У роботі використовуються такі методи дослідження:

- системний аналіз для дослідження структури програмного продукту.
- методи об'єктно-орієнтованого проектування для побудови архітектури системи.
- методи моделювання для опису взаємодії компонентів.
- Експериментальне програмування для реалізації функціональних модулів.
- Методи тестування для оцінки працездатності та ефективності програмного продукту.

**Наукова новизна** одержаних результатів полягає у розробці узагальненого підходу до створення користувацького інтерфейсу та функціональних компонентів інтерактивного програмного продукту, який

забезпечує підвищення зручності використання та ефективності взаємодії користувача з системою.

**Практичне значення** отриманих результатів полягає у створенні інтерактивного додатку, який може бути використаний як основа для подальших розробок у сфері створення тривимірних застосунків, навчальних симуляторів та ігрових проєктів. Розроблений програмний продукт також може застосовуватись як навчальний матеріал при підготовці студентів спеціальності «Комп'ютерні науки». Результати роботи можуть бути використані при розробці додатків різного призначення, а також у навчальному процесі для демонстрації принципів побудови інтерфейсу та функціональних компонентів інтерактивних систем[15].

#### **Структура кваліфікаційної роботи:**

Робота складається зі вступу, чотирьох розділів, висновків та списку використаних джерел. У першому розділі проведено аналіз існуючих рішень та визначено вимоги до інтерфейсу і функціональних компонентів. У другому - розглянуто процес проєктування архітектури програмного продукту. Третій розділ присвячено реалізації інтерфейсу та функціональних компонентів. У четвертому розділі наведено результати тестування та оцінки ефективності розробленої системи.

# **РОЗДІЛ 1**

## **АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ВИМОГ ДО ІНТЕРФЕЙСУ ТА ФУНКЦІОНАЛЬНИХ КОМПОНЕНТІВ ІНТЕРАКТИВНИХ ПРОГРАМНИХ ПРОДУКТІВ**

Сучасний розвиток інтерактивних додатків ставить високі вимоги до якості користувацького інтерфейсу та надійності функціональних компонентів. Користувач очікує інтуїтивного управління, швидкої реакції системи, сучасного візуального оформлення та можливості збереження прогресу в будь-який момент. Тому на етапі аналізу важливо оцінити доступні інструменти розробки, порівняти їх можливості та обрати найбільш підходящий для реалізації поставлених завдань[17].

### **1.1. Порівняльний аналіз сучасних 3D-рушіїв**

На сьогодні для створення тривимірних інтерактивних продуктів найбільш поширеними є три рушії: Unity, Unreal Engine та Godot. Кожен з них має свої сильні сторони, але відрізняється за рівнем зручності, швидкістю розробки та підходом до реалізації інтерфейсу та функціональних компонентів.

Unity 3D - це кросплатформний рушій, який вже багато років займає лідируючі позиції в індустрії завдяки балансу між потужністю та доступністю. Він пропонує вбудовані інструменти для швидкого створення інтерфейсу (Canvas + UGUI), готові компоненти для керування персонажем (CharacterController), навігації (NavMeshAgent) та збереження даних (JSON-серіалізація)[3]. Unity особливо зручний для студентських і середніх проєктів, оскільки має низький поріг входження, велику спільноту та безкоштовну ліцензію для більшості розробників.

Unreal Engine 5 - потужний рушій з винятковою графікою та візуальними можливостями. Він пропонує систему UMG для створення складних інтерфейсів і Behavior Tree для просунутого штучного інтелекту[7]. Однак Unreal має крутішу криву навчання, вищі системні вимоги та меншу швидкість

прототипування порівняно з Unity, що робить його менш оптимальним для бакалаврських робіт обмеженого терміну.

Godot 4.x - повністю відкритий і безкоштовний рушій з гнучкою системою вузлів. Він добре підходить для 2D-проектів і має низькі системні вимоги. Проте для повноцінних 3D-ігор з складним AI та системою збереження прогресу Godot часто потребує додаткових плагінів і більше ручної роботи, ніж Unity.

Для наочності порівняння основних характеристик представлено в таблиці 1.1.

**Таблиця 1.1** Порівняльний аналіз 3D-рушіїв (станом на 2025 рік)

| Параметр                        | Unity 3D                   | Unreal Engine 5           | Godot 4.x                 |
|---------------------------------|----------------------------|---------------------------|---------------------------|
| Зручність створення UI          | Висока<br>(Canvas + UGUI)  | Середня<br>(UMG)          | Висока<br>(Control nodes) |
| Швидкість прототипування        | Найвища                    | Середня                   | Висока                    |
| Вбудований AI                   | Повністю вбудований        | Behavior Tree             | Потребує плагінів         |
| Система збереження прогресу     | JSON + persistentDataPath  | SaveGame                  | JSON (вручну)             |
| Ліцензія                        | Безкоштовно до 200 тис. \$ | Royalty 5% після 1 млн \$ | Повністю безкоштовно      |
| Підтримка студентських проєктів | Найкраща                   | Середня                   | Добра                     |

З наведеного порівняння видно, що Unity 3D є оптимальним вибором для розробки інтерактивного програмного продукту в рамках кваліфікаційної роботи бакалавра.

## 1.2. Можливості Unity 3D

Unity 3D (версія 6.3 LTS) - це потужний, гнучкий і зручний рушій, що ідеально підходить для створення інтерактивних програмних продуктів з сучасним інтерфейсом і складними функціональними компонентами.

Серед ключових можливостей Unity, які були використані в роботі, слід виділити:

Компонентна архітектура - кожен об'єкт у сцені складається з окремих компонентів (Transform, CharacterController, NavMeshAgent, Animator тощо). Це дозволяє легко комбінувати та розширювати функціональність.

Система користувацького інтерфейсу (Canvas + UGUI) - швидке створення HUD, динамічних підказок, слайдерів і екранів завершення гри. Підтримка адаптивного масштабування та анімацій.

Input System - сучасна система введення, яка дозволяє гнучко налаштовувати керування (WASD + миша + клавіші взаємодії).

NavMeshAgent - вбудований інструмент для створення штучного інтелекту з автоматичним обходом перешкод.

CharacterController - готовий компонент для реалістичного керування персонажем від першої особи з підтримкою гравітації, стрибків і колізій.

JSON-серіалізація та Application.persistentDataPath - надійний і простий механізм збереження прогресу (позиція, здоров'я, кількість предметів).

Universal Render Pipeline (URP) - сучасний рендерер, який забезпечує високу продуктивність і якісне освітлення сцени[14].

Саме поєднання цих інструментів дозволило створити додаток, який має сучасний інтерфейс, стабільні функціональні компоненти та можливість швидкого розширення.

## 1.3. Аналіз підходів до створення інтерфейсу та функціональних компонентів

Створення сучасного інтерактивного програмного продукту вимагає не лише вибору рушія, а й правильного підходу до проєктування інтерфейсу та функціональних компонентів. Ці два елементи тісно пов'язані між собою:

інтерфейс визначає, наскільки зручно користувач взаємодіє з продуктом, а функціональні компоненти забезпечують логіку, стабільність і розширюваність системи.

Підходи до створення користувацького інтерфейсу

У сучасній розробці 3D-продуктів існують три основні підходи до реалізації інтерфейсу:

**Screen Space UI** - усі елементи інтерфейсу (здоров'я, підказки, меню) накладаються поверх 3D-сцени як 2D-шар. Це найпростіший і найшвидший у реалізації спосіб. Переваги: висока продуктивність, легкість адаптації під різні роздільні здатності, простота анімацій. Недоліки: менш «занурювальний» вигляд порівняно з іншими підходами.

**World Space UI** - елементи інтерфейсу існують безпосередньо в тривимірному світі як 3D-об'єкти. Такий підхід часто використовується для інтерактивних панелей, голограм або інформаційних табло. Переваги: висока реалістичність і занурення. Недоліки: складніше масштабування, вищі вимоги до продуктивності, проблеми з видимістю на великих відстанях.

**Гібридний підхід (Screen Space + World Space)** - найбільш сучасний і гнучкий. Основний HUD (індикатор здоров'я, лічильник предметів) знаходиться в екранному просторі, а динамічні підказки взаємодії (наприклад, «Press E to pick up») з'являються в світовому просторі біля об'єктів. Саме цей підхід обрано в роботі, оскільки він поєднує зручність управління з високим рівнем занурення.

У даному проєкті реалізовано гібридний підхід на базі **Canvas (Screen Space - Camera)**. Це дозволило досягти балансу між швидкістю роботи та візуальною привабливістю. Динамічні підказки реалізовано через **World Space Text**, що з'являється безпосередньо біля об'єкта взаємодії, завдяки чому користувач відчуває природність процесу. Функціональні компоненти (керування, штучний інтелект, взаємодія, збереження прогресу) є основою інтерактивності продукту. У сучасній практиці використовуються такі основні підходи:

**Компонентна архітектура** - кожен функціональний елемент є окремим компонентом, який можна додавати, видаляти або замінювати без переписування всього коду[7]. Цей підхід забезпечує високу модульність і був використаний у роботі (CharacterController, NavMeshAgent, PickupSystem тощо).

**Finite State Machine (FSM)** - скінченний автомат станів. Найпоширеніший спосіб реалізації штучного інтелекту (стан «Пошук», «Переслідування», «Атака»)[8]. У роботі саме на основі FSM побудовано поведінку ворогів.

**Event-driven architecture** - система подій і делегатів. Використовується для зв'язку між компонентами (наприклад, при підборі предмета викликається подія, яка оновлює лічильник і UI).

**Data-driven design** - дані (здоров'я, кількість предметів, позиція) відокремлені від логіки. Це дозволяє легко зберігати та завантажувати стан через JSON.

У роботі поєднано компонентну архітектуру з FSM і data-driven підходом. Така комбінація забезпечує чистоту коду, високу швидкодію та можливість швидкого розширення продукту (додавання нових предметів, ворогів або механік).

Переваги обраних підходів:

- Обрані підходи (гібридний UI + компонентна архітектура + FSM + JSON) мають ряд важливих переваг:
- Висока продуктивність навіть на середніх комп'ютерах.
- Легкість підтримки та розширення.
- Інтуїтивність для користувача.
- Можливість швидкого тестування окремих компонентів.

#### **1.4. Аналіз функціональних компонентів інтерактивних програмних продуктів**

У сучасних інтерактивних системах, зокрема в тривимірних ігрових середовищах, функціональні компоненти мають складну структуру та



взаємодіють між собою в режимі реального часу. Це пояснюється необхідністю забезпечення високої швидкості обробки подій, синхронізації між різними підсистемами та підтримки динамічної поведінки об'єктів. Принципово важливим є те, що всі компоненти мають працювати узгоджено - затримки або збої в одному з них здатні негативно вплинути на роботу системи в цілому.

Одним із ключових функціональних компонентів інтерактивного програмного продукту є система керування персонажем. Вона забезпечує взаємодію користувача з віртуальним середовищем шляхом обробки сигналів введення та перетворення їх у відповідні дії об'єкта. У середовищі Unity така система, як правило, реалізується із використанням компонента `CharacterController` або фізичного рушія, що дозволяє враховувати гравітацію, зіткнення та інші фізичні властивості[5]. Ключовим аспектом тут є забезпечення плавності руху, точності керування та передбачуваної реакції системи, оскільки саме це безпосередньо визначає якість користувацького досвіду.

Не менш важливою складовою є система штучного інтелекту, яка визначає поведінку неігрових персонажів та об'єктів середовища. У інтерактивних системах штучний інтелект зазвичай реалізується на основі скінченних автоматів, поведінкових дерев або навігаційних алгоритмів[16]. У середовищі Unity для цього широко застосовується компонент `NavMeshAgent`, що дозволяє автоматично прокладати маршрут об'єкта з урахуванням перешкод[4]. Такий підхід суттєво спрощує реалізацію складної поведінки, забезпечуючи при цьому стабільність і високу ефективність роботи системи.

Важливим елементом функціональної архітектури є система взаємодії з об'єктами, яка надає користувачеві можливість впливати на елементи середовища. Її реалізація передбачає визначення об'єктів, доступних для взаємодії, та обробку відповідних подій. У тривимірних системах для цього часто використовується метод `Raycast`, який дозволяє

визначити об'єкт, на який спрямований погляд або курсор користувача. Це забезпечує інтуїтивний та природний спосіб взаємодії, що є особливо важливим для ігрових застосунків.

Окрему роль відіграє система збереження та завантаження даних, яка забезпечує можливість відновлення стану системи після її завершення або перезапуску. У сучасних програмних продуктах для цього використовуються формати серіалізації, зокрема JSON, що дозволяють зберігати структуровані дані у зручному для подальшої обробки вигляді[10]. У середовищі Unity доступ до файлової системи здійснюється через спеціальні шляхи, наприклад `Application.persistentDataPath`, що гарантує кросплатформеність та безпечність збереження даних.

Завершує перелік ключових компонентів система обробки подій, яка забезпечує зв'язок між різними підсистемами програмного продукту. Вона дозволяє реагувати на дії користувача, зміни стану об'єктів та інші події у системі. Ефективна реалізація цієї системи сприяє зменшенню залежності між компонентами, підвищенню гнучкості архітектури та спрощенню процесу розробки загалом.

### **1.5. Аналіз архітектур інтерактивних програмних систем**

Архітектура програмної системи є одним із найважливіших чинників, що визначає якість, надійність та перспективність програмного продукту. Саме архітектурні рішення формують загальну структуру застосунку, встановлюють спосіб взаємодії між окремими модулями, визначають принципи розподілу функціональності та впливають на подальший розвиток програмного забезпечення.

Для інтерактивних програмних продуктів роль архітектури є особливо значущою, оскільки такі системи функціонують у режимі реального часу, безперервно обробляють дії користувача, змінюють внутрішній стан та оновлюють візуальне середовище. На відміну від звичайних інформаційних систем, вони мають підвищені вимоги до швидкодії, стабільності та гнучкості. Будь-яка затримка в обробці подій або

помилка у взаємодії компонентів безпосередньо позначається на користувацькому досвіді. Саме тому при проєктуванні таких систем необхідно приділяти особливу увагу вибору архітектурного підходу, який забезпечить ефективне функціонування продукту як на етапі первинної реалізації, так і в процесі подальшого супроводу та модернізації.

Архітектура інтерактивного програмного продукту має відповідати декільком ключовим вимогам. По-перше, система повинна бути модульною - складатися з окремих логічно завершених частин, які можна змінювати незалежно одна від одної. По-друге, важливою є масштабованість, тобто можливість розширення функціоналу без необхідності повного перепроєктування. По-третє, необхідною умовою є підтримуваність - здатність швидко виявляти помилки, оновлювати код та адаптувати продукт до нових вимог. Суттєве значення має й повторне використання компонентів, що дозволяє скоротити час розробки та знизити кількість помилок.

У сучасній практиці розробки інтерактивних програмних систем застосовується декілька архітектурних підходів, кожен із яких має власні переваги та обмеження. Вибір конкретної моделі залежить від масштабів проєкту, технічних вимог, складності функціоналу та обраного середовища розробки.

Одним із найпростіших є **монолітна архітектура**, у межах якої всі функціональні частини продукту реалізуються як єдиний застосунок із тісно пов'язаними компонентами. На ранніх етапах розробки такий підхід зручний, оскільки дозволяє швидко створити базову версію без складної організації взаємодії між модулями. Проте зі збільшенням обсягу функціоналу система поступово ускладнюється: зміни в одному компоненті можуть зачіпати інші, а підтримка великої кодової бази стає менш ефективною. Для невеликих навчальних або тестових проєктів монолітна архітектура може бути прийнятною, однак для складних інтерактивних систем вона має суттєві обмеження.

Більш сучасним рішенням є **модульна архітектура**, яка передбачає розділення продукту на незалежні підсистеми. Кожен модуль виконує визначену функцію: керування користувачем, роботу інтерфейсу, обробку фізики, збереження даних або реалізацію штучного інтелекту. Такий підхід забезпечує кращу організацію коду, дозволяє паралельно працювати над різними частинами проєкту та суттєво спрощує супровід системи. При необхідності внесення змін розробник може працювати з конкретним модулем, не втручаючись в інші підсистеми. Саме модульна архітектура вважається базовою для більшості сучасних програмних продуктів.

Для інтерактивних систем, що створюються у середовищі Unity, найбільш характерною є **компонентно-орієнтована архітектура**. Її сутність полягає в тому, що кожен об'єкт сцени формується як контейнер, до якого додаються окремі компоненти - позиціонування, візуалізація, фізика зіткнення та скрипт поведінки[7]. Поведінка об'єкта формується не через складну ієрархію класів, а шляхом комбінування незалежних функціональних блоків. Перевагою такого підходу є висока гнучкість: розробник може легко змінювати функціональність об'єкта, додаючи або видаляючи компоненти без необхідності переписування логіки. Наприклад, для перетворення статичного об'єкта на інтерактивний достатньо додати скрипт взаємодії та фізичний колайдер - це суттєво прискорює розробку та спрощує експериментування з різними механіками.

Компонентно-орієнтована архітектура також добре узгоджується з принципами повторного використання коду. Один і той самий компонент може застосовуватись до великої кількості об'єктів, що зменшує дублювання та підвищує якість системи. Водночас цей підхід потребує чіткої організації взаємодії між елементами - за відсутності продуманих правил система може стати надмірно фрагментованою, що ускладнить пошук помилок і розуміння загальної логіки. Саме тому важливо дотримуватись єдиних принципів найменування, структурування та взаємодії компонентів.

Окремої уваги заслуговує архітектурний шаблон **MVC (Model-ViewController)**, широко поширений в інформаційних системах і вебзастосунках[11]. У його основі лежить відокремлення даних системи від інтерфейсу користувача та логіки керування. У чистому вигляді MVC рідше застосовується у 3D-продуктах, однак окремі його принципи активно використовуються при побудові інтерфейсів, меню, систем налаштувань та збереження прогресу.

Для великих інтерактивних проєктів також можуть застосовуватись **подієво-орієнтовані архітектури**, у яких взаємодія між компонентами відбувається через події[8]. Натискання кнопки запускає подію відкриття меню, завершення місії активує подію перемоги, а зниження рівня здоров'я до нуля - подію завершення гри. Такий підхід дозволяє зменшити жорстку залежність між модулями та підвищити гнучкість системи.

У контексті даної роботи найбільш доцільним є поєднання компонентно-орієнтованої архітектури з елементами модульного та подієвого підходів. Основні об'єкти сцени реалізуються через систему компонентів Unity, логічні підсистеми відокремлюються у вигляді самостійних модулів, а взаємодія між ними організовується через події. Такий підхід забезпечує високу гнучкість, зручність супроводу та можливість подальшого розширення продукту. З практичної точки зору, для розроблюваної системи доцільно виділити такі архітектурні модулі: модуль керування персонажем, модуль користувацького інтерфейсу, модуль штучного інтелекту, модуль взаємодії з об'єктами, модуль збереження даних та модуль керування станами гри. Кожен із них несе власну відповідальність, що відповідає сучасним принципам програмної інженерії.

Проведений аналіз сучасних архітектурних підходів свідчить про те, що для інтерактивних програмних продуктів найбільш ефективними є модульні та компонентно-орієнтовані рішення. Вони дозволяють забезпечити високу продуктивність, зручність розробки, підтримуваність і

можливість масштабування системи. Саме ці принципи покладено в основу архітектури програмного продукту, що розробляється у межах даної кваліфікаційної роботи.

## РОЗДІЛ 2

# ПРОЄКТУВАННЯ АРХІТЕКТУРИ ІНТЕРФЕЙСУ ТА ФУНКЦІОНАЛЬНИХ КОМПОНЕНТІВ

### 2.1. Загальна архітектура програмного продукту

Етап проєктування є одним із найважливіших у процесі створення програмного забезпечення, оскільки саме на цьому етапі формується логічна структура системи, визначаються взаємозв'язки між її складовими, способи обробки даних, механізми взаємодії користувача з програмним продуктом та принципи реалізації функціональних можливостей. Для інтерактивних програмних продуктів значення якісного проєктування є особливо високим, оскільки такі системи повинні працювати у режимі реального часу, швидко реагувати на події, забезпечувати стабільність роботи та зручність використання.

У межах даної кваліфікаційної роботи проєктувався інтерактивний програмний продукт у вигляді тривимірної ігрової сцени, реалізованої в середовищі Unity 3D. Основною метою проєктування було створення логічно цілісної, масштабованої та зручної в супроводі системи, яка включає сучасний користувацький інтерфейс і набір функціональних компонентів, необхідних для повноцінної взаємодії користувача з віртуальним середовищем.

Загальна архітектура програмного продукту побудована за модульним принципом із використанням компонентно-орієнтованого підходу, характерного для середовища Unity. Система складається з окремих незалежних підсистем, кожна з яких виконує власні функції, але водночас взаємодіє з іншими модулями через визначені механізми обміну даними та подіями. До складу загальної архітектури входять такі основні підсистеми:

- модуль керування персонажем.
- модуль користувацького інтерфейсу.
- модуль взаємодії з об'єктами сцени.
- модуль штучного інтелекту супротивників.

- модуль збереження та завантаження прогресу.
- модуль керування станами гри.
- модуль аудіовізуального супроводу.

Кожен із зазначених модулів реалізує окрему функціональну область, що дозволяє спростити розробку, тестування та подальше вдосконалення системи.

**Модуль керування персонажем** відповідає за обробку введення користувача, переміщення ігрового персонажа, обертання камери, взаємодію з фізичним середовищем та виконання дій, пов'язаних із переміщенням у просторі. Саме цей модуль забезпечує безпосередній зв'язок користувача з віртуальним світом, тому до нього висуваються підвищені вимоги щодо точності, швидкодії та плавності роботи.

**Модуль користувацького інтерфейсу** реалізує відображення допоміжної інформації на екрані. До нього входять індикатори здоров'я персонажа, повідомлення, підказки, меню паузи, екрани завершення гри та інші елементи взаємодії. Правильно спроектований інтерфейс повинен бути інформативним, не перевантаженим зайвими деталями та інтуїтивно зрозумілим для користувача[17].

**Модуль взаємодії з об'єктами сцени** забезпечує можливість використання предметів, збирання ресурсів, активації механізмів або інших дій у середовищі. Реалізація цього модуля передбачає виявлення доступних для взаємодії об'єктів та обробку команд користувача.

**Модуль штучного інтелекту** відповідає за поведінку супротивників та автономних об'єктів середовища. До його функцій належать виявлення гравця, пересування по сцені, переслідування цілі, зміна станів поведінки та реагування на дії користувача.

**Модуль збереження прогресу** забезпечує запис поточного стану гри до зовнішнього сховища даних і подальше відновлення цієї інформації. У межах проєкту передбачено збереження позиції персонажа, стану здоров'я, зібраних предметів та інших параметрів.



**Модуль керування станами гри** відповідає за перемикання між різними режимами роботи системи - активна гра, пауза, перемога, поразка, завантаження рівня тощо.

З технічної точки зору архітектура побудована таким чином, щоб кожен модуль мав мінімальну залежність від інших частин системи. Це дозволяє змінювати або вдосконалювати окремі компоненти без необхідності повного перепроєктування продукту. У середовищі Unity основною структурною одиницею є `GameObject`, до якого можуть приєднуватися різні компоненти. Архітектура програмного продукту реалізується через набір ігрових об'єктів, які містять відповідні скрипти, візуальні елементи, фізичні компоненти та інші модулі. Наприклад, персонаж користувача містить компонент переміщення, камеру, колайдер, систему здоров'я та модуль взаємодії з предметами.

Для забезпечення зручності підтримки проєкту всі програмні класи структуровано за окремими скриптами: `Scripts/FirstPersonController`, `Scripts/EnemyAI`, `Scripts/SaveData`, `Scripts/GameManager` тощо. Такий підхід покращує навігацію в проєкті та відповідає сучасним принципам організації програмного коду[1].

При виборі архітектурних рішень також враховувались нефункціональні вимоги - продуктивність, стабільність роботи та можливість подальшого масштабування. Перевага надавалась легким та ефективним механізмам, які не створюють зайвого навантаження на процесор або відеосистему. Важливим аспектом є можливість розширення системи в майбутньому: до існуючої архітектури можна додати нові типи ворогів, інвентар, багаторівневу систему місій, нові елементи інтерфейсу або мережеву взаємодію без кардинальної зміни вже реалізованих компонентів.

Загальна архітектура програмного продукту являє собою сукупність взаємопов'язаних модулів, побудованих на основі компонентного підходу середовища Unity. Така структура забезпечує зручність розробки, ефективність функціонування, простоту тестування та перспективність подальшого розвитку програмного продукту.

## **2.2. Проектування користувацького інтерфейсу**

Користувацький інтерфейс є одним із найважливіших складників будь-якого інтерактивного програмного продукту, оскільки саме через нього здійснюється безпосередня взаємодія користувача із системою. Незалежно від технічної складності програмного забезпечення, рівня реалізації функціональних можливостей або якості внутрішньої архітектури, кінцеве сприйняття продукту значною мірою залежить від зручності, зрозумілості та ефективності інтерфейсу. У зв'язку з цим процес його проектування є одним із ключових етапів створення інтерактивної системи.

У межах даної кваліфікаційної роботи користувацький інтерфейс проектувався для тривимірного інтерактивного програмного продукту, реалізованого у середовищі Unity 3D. Основною метою було створення сучасного, інформативного та інтуїтивно зрозумілого інтерфейсу, який забезпечує зручну взаємодію з віртуальним середовищем і не перевантажує екран зайвими елементами.

На етапі проектування враховувалось, що система функціонує в режимі реального часу, тому користувач повинен швидко отримувати необхідну інформацію щодо поточного стану гри, власного персонажа, доступних дій та результатів взаємодії із середовищем. Водночас інтерфейс не повинен відволікати увагу від основного процесу або перекривати важливі елементи тривимірної сцени. Проектування здійснювалося з урахуванням сучасних принципів UI/UX-дизайну: логічність структури, мінімалізм, єдність стилю, швидкість сприйняття інформації, контрастність елементів, адаптивність до різних роздільних здатностей екрана та зручність керування[17].

У середовищі Unity для реалізації інтерфейсу використовується система Canvas, яка дозволяє створювати двовимірні елементи поверх тривимірної сцени. Такий підхід є оптимальним для програмних продуктів ігрового типу, оскільки забезпечує незалежність інтерфейсу від просторового розташування об'єктів та стабільне відображення службової інформації.

У структурі користувацького інтерфейсу передбачено декілька логічних груп елементів. До **першої групи** належать постійно активні інформаційні елементи, що відображаються під час основного процесу взаємодії: індикатор здоров'я персонажа, текстові підказки, інформація про зібрані предмети та прицільний маркер у центрі екрана. Ці елементи повинні бути компактними, але водночас добре помітними.

Індикатор здоров'я доцільно реалізувати у вигляді горизонтальної шкали типу Slider, яка змінює своє значення залежно від поточного стану персонажа. Такий спосіб відображення є найбільш зрозумілим, оскільки дозволяє миттєво оцінити рівень залишкового ресурсу. Для покращення сприйняття доцільно застосовувати зміну кольору шкали: зелений при високому рівні здоров'я, жовтий - при середньому, червоний - у критичному стані.

Важливим елементом є система текстових підказок, яка повідомляє користувача про можливість взаємодії з об'єктами, доступні клавіші керування або інші контекстні дії. Наприклад, при наближенні до предмета система відображає повідомлення щодо можливості його підбору. Такий підхід зменшує потребу у додатковому навчанні та робить процес взаємодії більш природним. Центральний прицільний маркер використовується як орієнтир під час переміщення камерою та взаємодії з об'єктами через Raycast-механізм. Він має бути мінімалістичним, не перекривати важливі елементи сцени та залишатися помітним на будь-якому фоні - найчастіше для цього використовується невеликий хрест або точка контрастного кольору.

До **другої групи** належать меню та екранні панелі, що відображаються залежно від стану системи: головне меню, меню паузи, екран перемоги, екран поразки та панель налаштувань. Ці елементи активуються лише за відповідних умов.

Головне меню є першим елементом, з яким взаємодіє користувач після запуску продукту. Воно містить базові команди запуску гри, виходу з програми та переходу до налаштувань. З позиції проєктування важливо забезпечити просту навігацію та зрозуміле розташування кнопок. Меню паузи

виконує допоміжну функцію і використовується для тимчасового припинення активного процесу. Під час його відкриття призупиняється рух об'єктів сцени, зупиняються анімації та блокується введення керування персонажем. Меню може містити кнопки продовження гри, перезапуску рівня або виходу до головного меню. Екрани перемоги та поразки виконують функцію завершення ігрового циклу та інформують користувача про результат проходження сцени. Вони повинні бути візуально виразними, містити коротке повідомлення та набір подальших дій - повторний запуск або повернення до меню.

Особливу увагу при проектуванні було приділено композиції елементів. Основні показники доцільно розташовувати по краях екрана, щоб не перекривати центральну частину сцени, а найбільш важливі повідомлення - у центрі або у верхній частині екрана. Такий підхід відповідає загальним принципам візуальної ієрархії та покращує швидкість сприйняття інформації. Також передбачено єдиний візуальний стиль: однакові шрифти, узгоджена кольорова схема, повторювані форми кнопок та уніфіковані відступи між елементами. Наявність єдиного стилю позитивно впливає на сприйняття продукту та створює відчуття цілісності системи.

З технічної точки зору структура інтерфейсу організована у вигляді окремих панелей: HUD-панель для постійних показників, PausePanel для меню паузи, ResultPanel для екрану завершення гри. Такий підхід спрощує програмне керування видимістю елементів та підвищує зручність супроводу проєкту. Окрему роль відіграє оптимізація інтерфейсу: надмірна кількість анімованих елементів або часті оновлення тексту можуть негативно вплинути на продуктивність системи, тому інтерфейс повинен бути функціональним, але технічно економним щодо використання ресурсів.

Проектування користувацького інтерфейсу здійснювалося як комплексний процес, що поєднує ергономічні, дизайнерські та технічні вимоги. Результатом є створення зручної системи взаємодії користувача з програмним середовищем, яка забезпечує швидке сприйняття інформації, інтуїтивність використання та комфортний процес роботи з продуктом.

### 2.3. Проєктування основних функціональних компонентів

Важливим етапом створення інтерактивного програмного продукту є проєктування функціональних компонентів, які забезпечують реалізацію внутрішньої логіки системи та її стабільну роботу. Якщо користувацький інтерфейс відповідає за зовнішню взаємодію з користувачем, то функціональні компоненти формують поведінку об'єктів, реакцію системи на події, виконання ігрових сценаріїв, збереження даних та підтримку загального стану програми. Саме якість їх проєктування безпосередньо впливає на надійність, продуктивність і можливість подальшого розширення продукту.

При розробці даної системи основна увага приділялася створенню модульної архітектури, у якій кожен компонент виконує чітко визначену функцію та може бути змінений або доповнений без суттєвого втручання в інші частини програми. Такий підхід є особливо актуальним для сучасної розробки програмного забезпечення, оскільки дозволяє швидко масштабувати проєкт, виправляти помилки та впроваджувати нові можливості.

Одним із базових компонентів системи є **модуль керування персонажем**. Через нього користувач отримує можливість взаємодіяти з віртуальним середовищем, переміщуватися сценою, оглядати простір, уникати небезпеки та виконувати поставлені завдання. На етапі проєктування було визначено, що система керування повинна бути інтуїтивно зрозумілою, швидкою у реагуванні та не перевантажувати користувача зайвими діями. Для цього було використано класичну схему переміщення за допомогою клавіш клавіатури та керування камерою за допомогою миші - стандарт для більшості сучасних тривимірних продуктів, який не потребує додаткового навчання.

Для технічної реалізації модуля обрано компонент `CharacterController`, який забезпечує коректний рух персонажа, врахування колізій із поверхнями, проходження по нерівностях та роботу гравітації. Перевагою цього рішення є відсутність потреби у складному фізичному моделюванні, що позитивно впливає на продуктивність програми. Крім того, систему керування

спроєктовано таким чином, щоб у майбутньому можна було легко додати біг, присідання, стрибки або інші додаткові можливості.

Другим важливим функціональним компонентом є **система штучного інтелекту ворожих об'єктів**. Її головне призначення полягає у створенні динамічного середовища, яке реагує на дії користувача та формує інтерактивний сценарій взаємодії. Без цього компонента сцена виглядала б статичною та передбачуваною, що значно знижує зацікавленість користувача.

При проєктуванні було визначено декілька базових сценаріїв поведінки ворога: очікування, патрулювання території, виявлення гравця, переслідування та атака. Для реалізації переміщення використано NavMeshAgent, який автоматично будує маршрут до цілі та дозволяє обходити перешкоди без ручного програмування складних алгоритмів навігації. Це рішення суттєво скорочує час розробки та забезпечує природну поведінку об'єктів у тривимірному просторі. Логіку штучного інтелекту спроєктовано за принципом скінченного автомата станів: ворог у кожний момент часу перебуває в одному з можливих режимів і переходить між ними залежно від подій у середовищі. Якщо гравець входить у зону виявлення - ворог переходить у режим переслідування, якщо дистанція скорочується до мінімальної - запускається атака. Такий підхід є ефективним, зрозумілим та легко розширюється новими сценаріями поведінки.

Наступним важливим компонентом стала **система взаємодії з предметами**. У сучасних інтерактивних продуктах користувач очікує можливості взаємодіяти з оточенням: підбирати ресурси, активувати механізми, відкривати двері, отримувати нові можливості. У межах роботи спроєктовано універсальний механізм взаємодії, який може застосовуватися до будь-яких об'єктів сцени. Принцип роботи системи базується на використанні променя Raycast, що спрямовується з камери користувача вперед. Якщо промінь потрапляє в інтерактивний об'єкт, система визначає його тип і виводить відповідну підказку на екран. Після натискання клавіші взаємодії запускається потрібна дія: підбір предмета, відкриття дверей або

запуск іншого сценарію. Перевагою такого підходу є його універсальність - один механізм використовується для великої кількості різних об'єктів без дублювання коду.

Значну роль у формуванні ігрової логіки виконує **система здоров'я персонажа**. Її проектування необхідне для створення елементів ризику, потреби уникати небезпеки та контролю успішності проходження сцени. Система здоров'я реалізована як окремий компонент, який зберігає поточний рівень ресурсів персонажа, обробляє отримання пошкоджень і запускає завершення гри у разі досягнення критичного значення. Важливою вимогою була синхронізація цього компонента з інтерфейсом: після кожної зміни здоров'я відповідний індикатор на екрані автоматично оновлюється, що дозволяє користувачу миттєво оцінювати свій стан і приймати рішення щодо подальших дій.

Окреме місце в архітектурі займає **система збереження та завантаження прогресу**. Наявність такого механізму є обов'язковою вимогою до сучасних інтерактивних систем, оскільки користувач повинен мати можливість продовжити роботу з моменту завершення попередньої сесії. На етапі проектування визначено перелік параметрів, що мають зберігатися: позиція персонажа, рівень здоров'я, кількість зібраних предметів, виконані дії та інші ключові характеристики. Для запису даних використано формат JSON, який є простим, компактним і зручним для серіалізації. Збереження виконується у спеціальній системній директорії, що забезпечує сумісність програми з різними операційними системами.

Не менш важливим компонентом стала **система керування загальними станами програми**. У процесі роботи продукт може перебувати в різних режимах: головне меню, активна гра, пауза, перемога або завершення гри. Для уникнення хаотичного керування логікою спроектовано центральний менеджер станів, який відповідає за перемикання між режимами, блокування керування під час паузи, запуск відповідних екранів інтерфейсу та перезапуск сцени. Централізований підхід дозволяє зменшити кількість залежностей між

окремими частинами системи та забезпечує зручність подальшого супроводу. У разі додавання нових режимів роботи достатньо розширити логіку одного керуючого модуля.

Усі описані компоненти спроектовано як взаємопов'язану систему. Керування персонажем змінює його координати у просторі, система штучного інтелекту реагує на нове положення користувача, механізм взаємодії дозволяє працювати з предметами, система здоров'я обробляє наслідки атак ворогів, а модуль збереження фіксує поточний стан усіх підсистем. Така інтеграція створює цілісне програмне середовище з високим рівнем функціональності.

У результаті проектування сформовано набір основних функціональних компонентів, які забезпечують повноцінну роботу інтерактивного програмного продукту. Розроблена архітектура характеризується модульністю, стабільністю, логічною структурованістю та можливістю подальшого розвитку, що створює надійну основу для практичної реалізації продукту в наступному розділі кваліфікаційної роботи.

#### **2.4. UML-модельовання архітектури системи**

Важливим етапом проектування програмного продукту є формалізація його структури та поведінки за допомогою стандартних засобів моделювання. Одним із найбільш ефективних підходів до цього є використання UML (Unified Modeling Language), який дозволяє наочно представити архітектуру системи, взаємодію її компонентів та логіку функціонування окремих підсистем. Застосування UML-моделей сприяє підвищенню зрозумілості структури програмного продукту, спрощує процес розробки та забезпечує можливість подальшого супроводу і масштабування системи[12].

У межах даної роботи UML-модельовання використовувалось для відображення як статичної, так і динамічної структури системи. Основна увага була приділена діаграмам класів, діаграмам прецедентів (use case), діаграмам послідовностей та діаграмам станів, оскільки саме вони найбільш повно описують архітектуру інтерактивного програмного продукту[12].



Першим етапом моделювання стало створення **діаграми прецедентів**, яка дозволяє визначити основні сценарії взаємодії користувача із системою. Основним актором виступає користувач (гравець), який взаємодіє з програмним продуктом через інтерфейс. До основних прецедентів належать: запуск гри, керування персонажем, взаємодія з об'єктами, уникнення ворогів, збереження прогресу та завершення гри. Ця діаграма дозволяє сформулювати загальне уявлення про функціональні можливості системи та визначити її основні сценарії використання[18].

Наступним кроком стало побудування **діаграми класів**, яка відображає статичну структуру програмного продукту. У межах цієї діаграми визначено основні класи системи, їх атрибути та методи, а також зв'язки між ними. Ключовими класами виступають PlayerController, EnemyAI, GameManager, UIManager, SaveManager та InteractionSystem. Кожен із них відповідає за окрему функціональну підсистему, що відповідає принципам модульного програмування[18].

Зокрема, клас PlayerController реалізує логіку керування персонажем, обробляє введення користувача та координує рух об'єкта у просторі. Клас EnemyAI відповідає за поведінку ворогів і містить алгоритми виявлення, переслідування та атаки. GameManager виступає центральним елементом системи, який координує загальний стан гри та забезпечує взаємодію між компонентами. UIManager управляє елементами інтерфейсу та їх оновленням, SaveManager реалізує механізми збереження та завантаження даних, а InteractionSystem забезпечує обробку взаємодії з об'єктами сцени.

Важливим аспектом є те, що між класами встановлено слабкі зв'язки через використання інтерфейсів та подій. Це дозволяє уникнути жорсткої залежності між модулями та спрощує внесення змін у систему. Наприклад, зміна реалізації збереження даних не потребує модифікації інших компонентів за умови збереження інтерфейсу взаємодії.

Для опису динамічної поведінки системи використано **діаграми послідовностей**, які відображають порядок виклику методів і взаємодію між

об'єктами в часі. Наприклад, сценарій підбору предмета включає такі етапи: користувач натискає клавішу взаємодії, система перевіряє наявність об'єкта через Raycast, InteractionSystem передає дані в Inventory або GameManager, після чого UIManager оновлює відображення. Така деталізація дозволяє виявити потенційні помилки ще на етапі проєктування та оптимізувати логіку взаємодії компонентів.

Окрему увагу було приділено **діаграмам станів**, які використовуються для опису поведінки об'єктів із внутрішньою логікою переходів між режимами. Найбільш доцільним є застосування цього типу діаграм для моделювання поведінки ворогів: об'єкт EnemyAI переходить між станами очікування, патрулювання, переслідування та атаки залежно від дій користувача. Такий підхід дозволяє чітко структурувати логіку штучного інтелекту та уникнути складних умовних конструкцій у коді. Діаграми станів також застосовуються для моделювання загального стану гри - система може перебувати у станах «меню», «гра», «пауза», «перемога» або «поразка», і кожен із них має власну логіку обробки подій. Це забезпечує впорядкованість роботи програми та дозволяє централізовано керувати її поведінкою[18].

Використання UML-моделювання у процесі проєктування має низку переваг. По-перше, це підвищує зрозумілість структури системи як для розробника, так і для сторонніх осіб. По-друге, це дозволяє виявити логічні помилки ще до етапу реалізації, що зменшує витрати часу на їх виправлення. По-третє, UML-діаграми можуть використовуватися як документація, що супроводжує програмний продукт і полегшує його подальший розвиток.

## **РОЗДІЛ 3.**

### **РЕАЛІЗАЦІЯ ІНТЕРФЕЙСУ ТА ОСНОВНИХ ФУНКЦІОНАЛЬНИХ КОМПОНЕНТІВ**

#### **3.1. Реалізація користувацького інтерфейсу**

Після завершення етапу проєктування наступним логічним кроком є практична реалізація користувацького інтерфейсу інтерактивного програмного продукту. Саме інтерфейс виступає основним засобом взаємодії користувача з системою, тому його якість безпосередньо впливає на зручність використання, швидкість освоєння та загальне враження від програмного продукту.

У межах даної роботи реалізація інтерфейсу здійснювалася в середовищі Unity з використанням системи Canvas та бібліотеки UGUI. Такий підхід дозволяє створювати гнучкі, адаптивні та продуктивні інтерфейси, які коректно відображаються на різних екранах і не залежать від складності тривимірної сцени.

На початковому етапі було визначено структуру інтерфейсу, яка включає декілька ключових екранів та елементів: основний ігровий інтерфейс (HUD), підказки взаємодії, індикатор здоров'я, а також екрани завершення гри (перемога та поразка). Така структура забезпечує користувача всією необхідною інформацією без перевантаження екрану зайвими елементами.

Основою інтерфейсу є Canvas, який виступає контейнером для всіх UI-елементів. Для забезпечення коректного масштабування було використано режим Screen Space - Overlay, що дозволяє інтерфейсу автоматично підлаштовуватися під роздільну здатність екрану. Додатково застосовано компонент Canvas Scaler з режимом Scale With Screen Size, що гарантує однакове відображення елементів на різних пристроях.

Одним із ключових елементів інтерфейсу є індикатор здоров'я персонажа. Він реалізований у вигляді Slider, який динамічно змінює своє значення залежно від поточного стану персонажа. Для забезпечення наочності

було обрано горизонтальне розташування індикатора у верхній частині екрана. Зміна значення відбувається через прив'язку до системи здоров'я, що забезпечує автоматичне оновлення без додаткових викликів з боку користувача.

Важливою частиною інтерфейсу є система підказок взаємодії. Вона реалізована у вигляді текстового елемента, який з'являється в центрі або нижній частині екрана у момент, коли користувач може взаємодіяти з об'єктом. Текст підказки формується динамічно залежно від типу об'єкта (наприклад, «Натисніть E, щоб підібрати предмет»). Це значно покращує зрозумілість ігрового процесу та знижує потребу в додатковому навчанні користувача.

Окрему роль відіграють екрани завершення гри. Було реалізовано два основні варіанти: екран поразки (Game Over) та екран перемоги (Victory). Обидва екрани створені як окремі панелі в Canvas, які за замовчуванням є прихованими. Вони активуються через GameManager у відповідний момент часу. Кожен із цих екранів містить текстове повідомлення та кнопки для перезапуску гри або виходу до головного меню.

Для реалізації інтерактивності кнопок використано стандартні компоненти Button, які підтримують обробку подій натискання. Логіка кнопок пов'язана з менеджером гри, що дозволяє централізовано обробляти дії користувача. Особливу увагу під час реалізації було приділено оптимізації інтерфейсу. Усі UI-елементи були згруповані в ієрархії Canvas, що дозволяє зменшити кількість обчислень під час рендерингу. Крім того, було мінімізовано використання складних анімацій, що позитивно вплинуло на продуктивність.

Ще одним важливим аспектом є адаптивність інтерфейсу. Завдяки використанню анкерів (anchors) та гнучкого масштабування елементи інтерфейсу зберігають своє розташування незалежно від роздільної здатності екрану. Це дозволяє використовувати програмний продукт на різних пристроях без додаткового налаштування.

У процесі реалізації також було враховано принципи UX/UI дизайну: мінімалізм, читабельність, логічне розташування елементів та швидкий доступ до важливої інформації[9]. Інтерфейс не перевантажений зайвими деталями, що дозволяє користувачу зосередитися на основному ігровому процесі[17].

У результаті реалізації було створено сучасний, функціональний та зручний користувацький інтерфейс, який забезпечує ефективну взаємодію користувача з програмним продуктом. Реалізовані рішення відповідають сучасним вимогам до інтерактивних систем і можуть бути використані як основа для подальшого розвитку та вдосконалення програмного продукту[15].

### 3.2. Реалізація компонента керування персонажем

Реалізація компонента керування персонажем є одним із ключових етапів створення інтерактивного програмного продукту, оскільки саме цей компонент забезпечує безпосередню взаємодію користувача з ігровим середовищем. У межах даної кваліфікаційної роботи було реалізовано систему керування від першої особи з використанням сучасних можливостей середовища Unity, зокрема компонентів **CharacterController** та **Input System**.

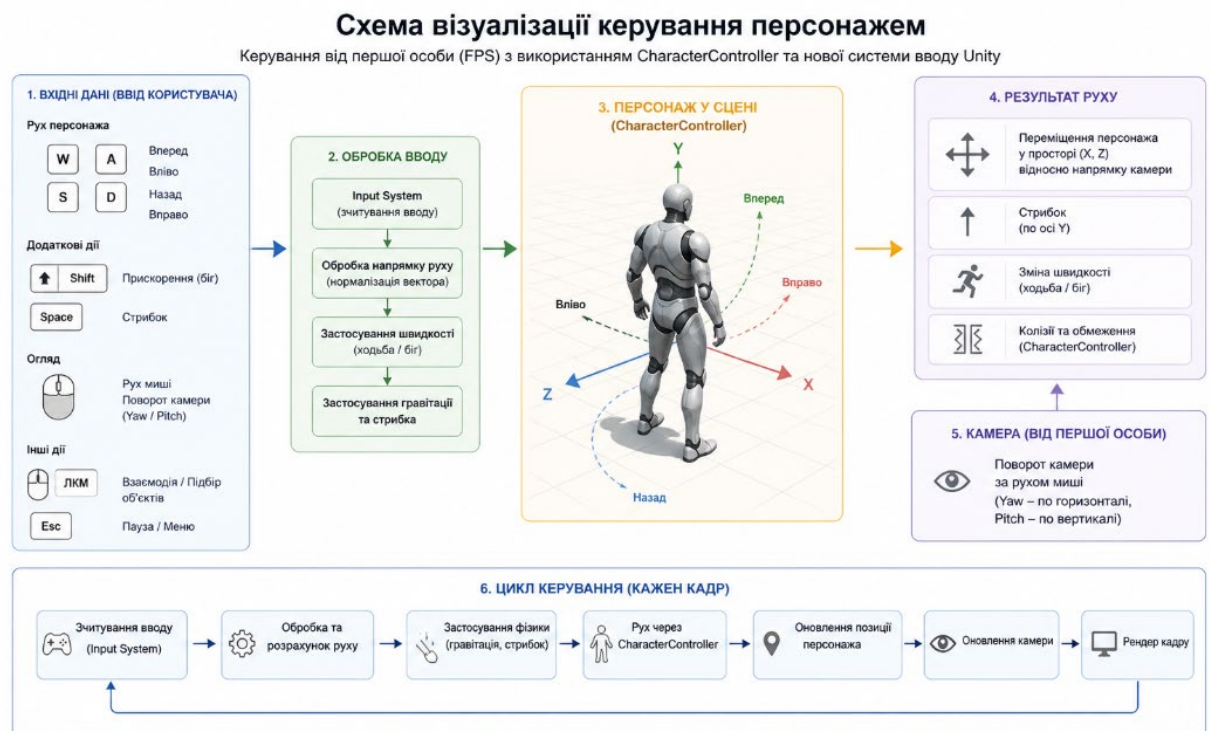


Рисунок 1 Керування персонажем

CharacterController - це спеціалізований компонент Unity, призначений для переміщення персонажа у тривимірному просторі без використання повноцінної фізичної моделі, яка базується на Rigidbody[5]. Його основна перевага полягає в тому, що він забезпечує контрольоване та передбачуване переміщення об'єкта, що особливо важливо для ігор від першої особи.

На відміну від фізичних об'єктів, CharacterController не підпадає під вплив сил автоматично. Це означає, що гравітація, стрибки та інші аспекти руху повинні реалізовуватися програмно. Такий підхід дозволяє повністю контролювати поведінку персонажа та уникати небажаних ефектів, наприклад ковзання або нестабільних зіткнень. Основною функцією цього компонента є метод Move(), який приймає вектор переміщення і виконує рух з урахуванням колізій. Завдяки цьому забезпечується коректна взаємодія з поверхнями, стінами та іншими об'єктами сцени. Також компонент автоматично визначає, чи знаходиться персонаж на землі (isGrounded), що значно спрощує реалізацію стрибків і гравітації.

У реалізованій системі керування CharacterController використовується для:

- обробки переміщення персонажа;
- визначення контакту з поверхнею;
- реалізації стрибків і падіння;
- забезпечення стабільної колізії без складних фізичних розрахунків.

Таким чином, цей компонент дозволяє зосередитися на логіці гри, а не на низькорівневих деталях фізики.

Не менш важливим є використання Input System - сучасної системи введення в Unity, яка прийшла на зміну застарілому Input Manager[6]. Вона забезпечує більш гнучкий та масштабований підхід до обробки дій користувача.

Основною ідеєю Input System є абстракція дій користувача у вигляді «дій» (Actions), які можуть бути прив'язані до різних пристроїв введення[6].

Наприклад, дія «Move» може відповідати як клавішам WASD на клавіатурі, так і стику геймпада. Це дозволяє легко адаптувати систему керування під різні платформи без зміни логіки програми.

У реалізованому компоненті керування використовується клас `PlayerInput`, який забезпечує доступ до налаштованих дій. Зокрема, були визначені такі дії:

- `Move` - для переміщення персонажа;
- `Look` - для керування камерою;
- `Jump` - для виконання стрибка;
- `Sprint` - для активації бігу.

Зчитування введення здійснюється через метод `ReadValue`, який повертає значення у відповідному форматі (наприклад, `Vector2` для руху або огляду). Це дозволяє отримувати дані незалежно від конкретного пристрою введення.

Важливою перевагою `Input System` є підтримка подій (events), що дозволяє реагувати на дії користувача в момент їх виникнення, а не лише опитувати стан кожен кадр. У даній реалізації використовується комбінований підхід: для руху застосовується постійне зчитування значень, а для стрибка - перевірка тригера (triggered), що дозволяє точно визначити момент натискання клавіші.

Ще однією перевагою є можливість централізованого налаштування керування через редактор Unity. Це означає, що зміна клавіш або додавання нових пристроїв введення не потребує внесення змін у програмний код.

Основою реалізації виступає клас `PlayerControllerWithCamera`, який поєднує в собі логіку руху персонажа, керування камерою, обробку введення користувача та анімацію. Важливою особливістю є використання атрибутів `[RequireComponent]`, які гарантують наявність необхідних компонентів (`CharacterController`, `Animator`, `PlayerInput`) на об'єкті, що запобігає помилкам під час виконання програми. На етапі ініціалізації (метод `Awake`) відбувається отримання посилань на всі необхідні компоненти системи. Зокрема,

ініціалізуються `CharacterController`, який відповідає за фізичне переміщення персонажа, `Animator` для керування анімаціями та `PlayerInput`, що забезпечує роботу з новою системою введення. Також виконується прив'язка дій користувача до відповідних змінних (`moveAction`, `lookAction`, `jumpAction`, `sprintAction`), що дозволяє гнучко налаштовувати керування без зміни коду. Додатково в цьому методі здійснюється налаштування курсора: він блокується в центрі екрана та приховується, що є стандартною практикою для ігор від першої особи. Це забезпечує коректну роботу керування камерою та підвищує рівень занурення користувача.

Метод **Start** реалізує важливу функцію інтеграції із системою збереження. Якщо в системі існують збережені дані, позиція персонажа автоматично відновлюється:

```
if (SaveSystem.instance != null)
{
    transform.position = SaveSystem.instance.LoadGame();
}
```

Це демонструє взаємодію компонента керування з іншими підсистемами, зокрема із системою збереження прогресу.

Основна логіка роботи компонента реалізована в методі `Update`, який викликається кожен кадр і відповідає за обробку руху та огляду:

```
void Update()
{
    HandleMovement();
    HandleLook();
}
```

Метод `HandleMovement` реалізує переміщення персонажа у просторі. Він зчитує введення користувача у вигляді двовимірного вектора, що відповідає напрямку руху (вперед/назад, ліворуч/праворуч). На основі цього формується тривимірний вектор руху відносно орієнтації персонажа:

$$\text{Vector3 move} = \text{transform.right} * \text{input.x} + \text{transform.forward} * \text{input.y};$$



Додатково враховується можливість бігу через множник швидкості (*runMultiplier*), що активується при натисканні відповідної клавіші. Це дозволяє динамічно змінювати швидкість пересування залежно від дій користувача.

Особливу увагу приділено реалізації гравітації та стрибка. Якщо персонаж знаходиться на землі, вертикальна швидкість скидається, що запобігає накопиченню значення. При натисканні клавіші стрибка встановлюється початкова вертикальна швидкість:

```
if (jumpAction != null && jumpAction.triggered)
{
    verticalVelocity = jumpForce;
}
```

У випадку перебування в повітрі до вертикальної швидкості застосовується гравітація, що створює реалістичну фізичну поведінку.

Переміщення персонажа виконується через метод *CharacterController.Move*, який враховує всі обчислені параметри:

```
controller.Move(move * targetSpeed * Time.deltaTime);
```

Важливою складовою є синхронізація руху з анімаціями. Швидкість руху та напрямок передаються в *Animator*, що дозволяє плавно змінювати анімації залежно від дій користувача. Для цього використовується інтерполяція (*Mathf.Lerp*), яка забезпечує плавність переходів і покращує візуальне сприйняття.

Метод *HandleLook* відповідає за керування камерою та орієнтацією персонажа. Він зчитує рух миші та перетворює його на обертання по горизонталі (поворот персонажа) і вертикалі (нахил камери). Для уникнення неприродних положень використовується обмеження кута нахилу:

```
cameraPitch = Mathf.Clamp(cameraPitch, -80f, 80f);
```

Це запобігає надмірному обертанню камери та забезпечує комфортне управління.

Додатково реалізовано систему згладжування поворотів і "мертвої зони", що усуває дрібні коливання при незначних рухах миші. Це підвищує точність керування та покращує користувацький досвід.

Важливою особливістю є інтеграція з системою паузи. Перед обробкою огляду перевіряється стан гри:

```
if (pauseManager.isPaused) return;
```

Це дозволяє повністю блокувати керування під час паузи, що є необхідною умовою коректної роботи інтерфейсу.

Окрім компонента руху, важливу роль у керуванні персонажем відіграє система здоров'я, реалізована в класі `PlayerHealth`. Вона відповідає за обробку пошкоджень, оновлення інтерфейсу та завершення гри. Зокрема, при отриманні шкоди виконується зменшення значення здоров'я та перевірка на досягнення критичного рівня:

```
health -= damage;  
if (health <= 0)  
{  
    Die();  
}
```

У випадку смерті персонажа вимикається компонент керування та викликається метод завершення гри через менеджер:

```
controller.enabled = false;  
GameManager.Instance.GameOver();
```

Це демонструє тісну інтеграцію між компонентами керування, системою здоров'я та глобальним менеджером гри.

### **3.3. Реалізація системи штучного інтелекту ворогів**

Однією з ключових складових інтерактивного програмного продукту є система штучного інтелекту, яка відповідає за поведінку ворожих об'єктів. Саме вона створює динамічність ігрового процесу, змушує користувача адаптуватися до змін середовища та підвищує загальний рівень складності. У межах даної роботи було реалізовано систему штучного інтелекту ворогів із

використанням навігаційної системи Unity (NavMesh) та власної логіки виявлення і переслідування гравця.

Реалізація штучного інтелекту в середовищі Unity базується на поєднанні декількох рівнів, кожен із яких відповідає за окремий аспект поведінки неігрових персонажів. Такий підхід дозволяє створювати гнучкі, масштабовані та реалістичні системи, які забезпечують динамічну взаємодію з користувачем.

Базовим рівнем реалізації ШІ в Unity є система навігації, зокрема NavMesh. Вона дозволяє персонажам будувати маршрути, обходити перешкоди та коректно переміщуватися в тривимірному просторі[4]. Навігаційна сітка формується на основі геометрії сцени та визначає області, доступні для пересування. Компонент NavMeshAgent використовує цю сітку для автоматичного прокладання оптимального маршруту до цілі, враховуючи перешкоди та топологію середовища. Хоча система NavMesh не визначає безпосередньо поведінку персонажа, вона є фундаментом для всіх подальших рішень, оскільки без коректного руху будь-який інтелект втрачає сенс. Саме завдяки цьому інструменту розробник може уникнути реалізації складних алгоритмів пошуку шляху, таких як  $A^*$ [7], і зосередитися на логіці поведінки.

Наступним рівнем є реалізація логіки прийняття рішень. У класичному підході для цього використовуються кінцеві автомати станів (Finite State Machines). У такій моделі персонаж перебуває в одному з можливих станів, наприклад: очікування, патрулювання, переслідування або взаємодія. Перехід між станами відбувається на основі визначених умов, таких як виявлення гравця або досягнення певної точки. Перевагою цього підходу є його простота та зрозумілість. Він добре підходить для реалізації базових сценаріїв поведінки, що було використано і в даній роботі при створенні логіки ворогів. Проте зі зростанням складності гри та кількості станів виникає проблема масштабованості: кількість переходів між станами значно збільшується, що ускладнює підтримку та розширення системи.



Рисунок 2. Finite State Machines

Саме тому в сучасних проєктах дедалі частіше застосовуються поведінкові дерева (Behavior Trees). Цей підхід дозволяє будувати ієрархічну структуру прийняття рішень, де кожен вузол відповідає за певну умову або дію. Поведінкові дерева забезпечують більшу гнучкість, повторне використання логіки та кращу читабельність складних сценаріїв[16]. Вони дозволяють уникнути «вибуху станів», характерного для кінцевих автоматів, і значно спрощують реалізацію складної поведінки NPC.

Окрім руху та прийняття рішень, важливою складовою штучного інтелекту є система сприйняття. Саме вона забезпечує зв'язок між діями гравця та реакцією ігрових персонажів. Без цього компоненту навіть найскладніша логіка поведінки залишатиметься неактивною.

У Unity система сприйняття зазвичай реалізується за допомогою комбінації декількох методів. Найпростішим є перевірка відстані між персонажем і гравцем, яка дозволяє визначити, чи знаходиться об'єкт у зоні досяжності. Додатково використовується перевірка кута огляду, що формує поле зору персонажа і обмежує область, у якій він може «бачити» гравця.

Для більш точної перевірки застосовуються променеві обчислення (Raycast), які дозволяють визначити наявність перешкод між NPC і ціллю. Це

забезпечує реалістичну поведінку, коли персонаж не реагує на гравця, якщо той знаходиться за стіною або іншим об'єктом. Також можуть використовуватися тригерні колайдери, які спрацьовують при входженні гравця в певну область.

Комбінація цих підходів дозволяє створити ефект «сприйняття» - персонажі не просто рухаються за заданим алгоритмом, а реагують на зміну ситуації в середовищі. Саме це формує відчуття живого ігрового світу та підвищує рівень занурення користувача[16].

Можливості Unity для реалізації штучного інтелекту охоплюють три основні рівні: навігацію, логіку прийняття рішень і систему сприйняття. Їх поєднання дозволяє створювати складні та адаптивні поведінкові моделі, які можуть змінюватися залежно від дій користувача. У межах даної роботи ці підходи були реалізовані у спрощеному вигляді, що забезпечило баланс між складністю реалізації та ефективністю роботи програмного продукту. Основою реалізації виступає клас `EnemyAI_NavMesh`, який керує поведінкою ворога в реальному часі. Для забезпечення переміщення використовується компонент `NavMeshAgent`, що дозволяє автоматично будувати маршрут до цілі, обходити перешкоди та забезпечувати природну навігацію в тривимірному просторі. Це значно спрощує реалізацію складних алгоритмів руху та дозволяє зосередитися на логіці поведінки.

На початковому етапі (метод `Start`) відбувається отримання посилання на компонент `NavMeshAgent`, після чого система готова до роботи. Основна логіка реалізована в методі `Update`, який виконується кожен кадр і відповідає за визначення стану ворога та вибір відповідної поведінки. Центральним елементом системи є механізм виявлення гравця. Для цього використовується метод `DetectPlayer`, який аналізує три ключові параметри: відстань до гравця, кут огляду та наявність перешкод між ворогом і ціллю. Спочатку обчислюється напрямок до гравця та відстань між об'єктами. Далі визначається кут між напрямком руху ворога та напрямком на гравця. Це дозволяє реалізувати обмежене поле зору, що наближене до реальної

поведінки. Якщо гравець знаходиться в межах допустимої дистанції та кута огляду, додатково виконується перевірка за допомогою променя (Raycast), який дозволяє визначити, чи немає перешкод між ворогом і гравцем. Якщо перешкод немає, ворог «бачить» гравця та переходить у режим переслідування. У цей момент зберігається остання відома позиція гравця, що використовується для подальшого пошуку. Коли ворог бачить гравця, він активує рух і встановлює точку призначення через метод `SetDestination`, що змушує його рухатися безпосередньо до цілі:

```
agent.SetDestination(player.position);
```

Якщо ж гравець виходить із поля зору, система переходить у режим пошуку. У цьому стані ворог рухається до останньої зафіксованої позиції гравця та протягом певного часу намагається його знайти. Для цього використовується таймер (`searchTimer`), який поступово зменшується. Якщо за відведений час гравець не був повторно виявлений, ворог припиняє існування:

```
if (searchTimer <= 0f)
{
    Destroy(gameObject);
}
```

Такий підхід дозволяє оптимізувати продуктивність, оскільки непотрібні об'єкти видаляються зі сцени.

Окрім логіки руху та пошуку, важливим елементом є система нанесення шкоди. Вона реалізована в окремому класі `EnemyDamage`, який обробляє зіткнення ворога з гравцем. При входженні в тригер перевіряється, чи об'єкт має тег "Player", після чого викликається метод отримання шкоди:

```
player.TakeDamage(damage);
```

У даній реалізації значення шкоди встановлено на рівні 100, що означає миттєве знищення персонажа при контакті. Це створює високий рівень напруги та змушує користувача уникати ворогів.

Додатковим компонентом системи є механізм генерації ворогів, реалізований у класі `EnemySpawner`. Він відповідає за появу нових ворогів у

випадку, якщо попередній був знищений. Логіка роботи базується на таймері: якщо ворог відсутній, запускається відлік часу, після чого створюється новий об'єкт у випадковій позиції поблизу гравця. Позиція генерації визначається як випадкове зміщення відносно позиції гравця:

$$\text{Vector3 spawnPos} = \text{player.position} + \\ \text{new Vector3}(\text{Random.Range}(-10f, 10f), 0, \text{Random.Range}(-10f, 10f));$$

Після створення ворога йому передається посилення на гравця, що дозволяє системі штучного інтелекту одразу почати роботу. Важливою особливістю реалізованої системи є її модульність. Логіка виявлення, переслідування, пошуку та нанесення шкоди розділена на окремі компоненти, що значно спрощує підтримку та розширення функціональності. Наприклад, у майбутньому можна додати нові стани поведінки (патрулювання, втеча, групова взаємодія) без суттєвої зміни існуючої структури.

## РОЗДІЛ 4

### ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ПРОГРАМНОГО ПРОДУКТУ

#### 4.1. Організація процесу тестування

Після завершення реалізації інтерактивного програмного продукту важливим етапом є проведення тестування, основною метою якого є перевірка коректності роботи системи, виявлення помилок та оцінка стабільності функціонування всіх компонентів. Тестування дозволяє визначити, наскільки програмний продукт відповідає поставленим вимогам, а також оцінити якість реалізованих механізмів взаємодії, продуктивність та зручність використання.

У межах даної роботи тестування проводилося поетапно та охоплювало як окремі функціональні компоненти, так і систему в цілому. Основна увага приділялася перевірці:

- коректності керування персонажем;
- стабільності роботи штучного інтелекту;
- правильності взаємодії з об'єктами;
- функціонуванню системи здоров'я;
- роботі інтерфейсу користувача;
- механізму збереження та завантаження даних;
- загальної продуктивності програмного продукту.

Тестування виконувалося у середовищі Unity Editor із використанням режиму Play Mode, що дозволило оперативно перевіряти роботу окремих компонентів та аналізувати поведінку системи в реальному часі. Додатково використовувалися вбудовані інструменти Unity Profiler, які дають змогу оцінити навантаження на процесор, використання пам'яті та швидкість рендерингу сцени.

На початковому етапі проводилося модульне тестування окремих компонентів системи. Такий підхід дозволяє локалізувати помилки на ранньому етапі та спростити процес їх виправлення. Наприклад, окремо



перевірялася система керування персонажем: коректність руху, обробка гравітації, робота стрибків та плавність керування камерою. Під час тестування компонента керування було встановлено, що система стабільно реагує на введення користувача та забезпечує плавне переміщення персонажа без різких змін координат. Також було перевірено роботу обмеження кута огляду камери, що запобігає некоректному обертанню та покращує комфорт користувача.

Наступним етапом стало тестування системи штучного інтелекту. Перевірялася коректність роботи NavMeshAgent, виявлення гравця, переслідування та система нанесення пошкоджень. Особлива увага приділялася перевірці поведінки ворога при втраті гравця з поля зору. У результаті тестування було підтверджено, що система коректно переходить між станами та не створює конфліктів у роботі AI.

Окремо тестувалася система взаємодії з об'єктами. Перевірялася точність роботи Raycast, відображення підказок та коректність активації інтерактивних елементів. Було встановлено, що система стабільно визначає об'єкти в полі зору користувача та правильно виконує задані сценарії взаємодії.

Для перевірки системи збереження виконувалося тестування запису та відновлення даних. Зокрема, перевірялося:

- збереження позиції персонажа;
- відновлення стану після перезапуску;
- коректність роботи JSON-файлу;
- стабільність роботи при відсутності файлу збереження.

У ході тестування було підтверджено, що система успішно відновлює попередній стан гри та не викликає критичних помилок навіть у випадку пошкодження або видалення файлу збереження.

Важливою складовою тестування стала перевірка користувацького інтерфейсу. Аналізувалася адаптивність UI при різних роздільних здатностях екрана, швидкість оновлення індикаторів та коректність роботи кнопок меню.

У результаті тестування встановлено, що інтерфейс коректно масштабується та забезпечує комфортну взаємодію незалежно від параметрів екрана.

Після завершення модульного тестування було проведено інтеграційне тестування, метою якого стала перевірка взаємодії між окремими компонентами системи. У межах цього етапу аналізувалося:

- взаємодія системи здоров'я та UI;
- зв'язок AI із системою пошкоджень;
- передача даних між SaveSystem та GameManager;
- взаємодія компонентів керування й анімації.

Інтеграційне тестування підтвердило, що всі компоненти функціонують узгоджено та не створюють конфліктів під час одночасної роботи.

Окрім функціонального тестування, було проведено оцінку продуктивності програмного продукту. Для цього аналізувалася частота кадрів (FPS), навантаження на процесор та використання оперативної пам'яті. Тестування проводилося на середньому за продуктивністю комп'ютері, що дозволило оцінити ефективність оптимізації системи.

У результаті було встановлено, що програмний продукт стабільно працює із частотою понад 60 FPS, а використання NavMesh та компонентної архітектури дозволило мінімізувати навантаження на систему. Також було підтверджено, що використання CharacterController є менш ресурсоємним порівняно з повноцінною фізичною моделлю Rigidbody.

Важливим аспектом стала перевірка зручності використання. Для цього оцінювалася інтуїтивність інтерфейсу, швидкість реакції системи та загальна комфортність взаємодії користувача з програмним продуктом. У результаті тестування було визначено, що реалізована структура інтерфейсу є зрозумілою та не потребує додаткового навчання користувача.

Проведене тестування підтвердило працездатність, стабільність та ефективність реалізованого програмного продукту. Усі основні функціональні компоненти коректно взаємодіють між собою, а система в цілому відповідає

поставленим вимогам та може використовуватися як основа для подальшого розвитку й удосконалення.

#### **4.2. Аналіз результатів тестування та оцінка ефективності системи**

Після завершення процесу тестування було проведено комплексний аналіз отриманих результатів, метою якого стала оцінка ефективності реалізованого програмного продукту, визначення рівня його стабільності, продуктивності та відповідності поставленим функціональним вимогам. Аналіз результатів дозволяє не лише підтвердити працездатність системи, але й оцінити доцільність використаних технологічних рішень, архітектурних підходів і методів реалізації окремих компонентів.

Одним із головних критеріїв оцінювання стала стабільність роботи програмного продукту. Під час багаторазового запуску та тривалого тестування не було виявлено критичних помилок, які могли б призвести до аварійного завершення програми або втрати даних користувача. Усі основні функціональні компоненти - система керування персонажем, штучний інтелект, система взаємодії, інтерфейс та механізм збереження - працювали коректно та взаємодіяли між собою без конфліктів.

Особливу увагу було приділено оцінці системи керування персонажем. У результаті тестування встановлено, що використання компонента `CharacterController` забезпечило плавне переміщення та стабільну взаємодію з ігровим середовищем. На відміну від фізичної моделі, заснованої на `Rigidbody`, даний підхід дозволив уникнути небажаних фізичних ефектів, таких як ковзання або нестабільна поведінка при зіткненнях. Крім того, використання сучасної системи введення `Input System` забезпечило швидку реакцію на дії користувача та спростило підтримку різних пристроїв керування.

У ході аналізу системи штучного інтелекту було підтверджено ефективність використання `NavMesh` та компоненту `NavMeshAgent`. Вороги коректно визначали маршрут до гравця, обходили перешкоди та реагували на зміну положення цілі в режимі реального часу. Завдяки використанню

кінцевого автомата станів вдалося реалізувати зрозумілу та стабільну модель поведінки без перевантаження системи складними алгоритмами.

Позитивним результатом стала також ефективна робота системи сприйняття. Комбінація перевірки дистанції, кута огляду та Raycast дозволила створити реалістичну реакцію ворогів на дії користувача. У процесі тестування було встановлено, що NPC не реагують на гравця через перешкоди, що підвищує рівень достовірності поведінки штучного інтелекту.

Аналіз системи взаємодії з об'єктами показав, що використання Raycast є ефективним рішенням для реалізації інтерактивності в тривимірному просторі. Система стабільно визначає об'єкти перед користувачем і дозволяє виконувати взаємодію без затримок або помилкових спрацьовувань. Відображення текстових підказок додатково покращило зрозумілість ігрового процесу та позитивно вплинуло на зручність використання.

Окрему увагу було приділено оцінці системи збереження даних. У результаті тестування підтверджено, що використання JSON для серіалізації інформації є ефективним та достатньо швидким рішенням. Система успішно відновлювала стан гри після повторного запуску програми, а також коректно обробляла ситуації, пов'язані з відсутністю або пошкодженням файлу збереження. Це забезпечило високий рівень надійності роботи програмного продукту.

Важливим критерієм оцінювання стала продуктивність системи. Під час тестування аналізувалися:

- частота кадрів (FPS);
- навантаження на центральний процесор;
- використання оперативної пам'яті;
- стабільність рендерингу сцени.

У результаті було встановлено, що програмний продукт стабільно підтримує частоту понад 60 кадрів за секунду навіть при одночасній роботі системи AI, фізики та інтерфейсу. Це свідчить про ефективність використаної архітектури та достатній рівень оптимізації.

- Позитивний вплив на продуктивність забезпечили:
- використання NavMesh замість складних алгоритмів пошуку шляху;
- застосування CharacterController замість фізичних Rigidbody;
- компонентна архітектура Unity;
- розділення логіки на окремі модулі;
- мінімізація кількості зайвих обчислень у методі Update.

Також було проведено оцінку зручності використання програмного продукту. Інтерфейс продемонстрував достатній рівень адаптивності та коректно відображався при різних роздільних здатностях екрана. Розташування елементів UI забезпечує швидкий доступ до важливої інформації та не перевантажує екран другорядними компонентами.

Аналіз результатів тестування дозволив визначити і певні обмеження реалізованої системи. Зокрема, поведінка штучного інтелекту реалізована у спрощеному вигляді та не враховує складні сценарії групової взаємодії або адаптивного навчання. Крім того, система збереження підтримує лише базовий набір параметрів і не реалізує декілька слотів збереження.

Проте зазначені обмеження не впливають критично на працездатність програмного продукту та можуть бути усунуті в процесі подальшого розвитку системи. За результатами тестування реалізовані технологічні рішення є ефективними і забезпечують коректність роботи всіх основних функціональних компонентів.

#### **4.3. Перспективи розвитку та вдосконалення програмного продукту**

Розроблений інтерактивний програмний продукт є функціонально завершеною системою, яка реалізує основні механізми взаємодії користувача з віртуальним середовищем, проте сучасні тенденції розвитку програмного забезпечення та ігрових технологій передбачають постійне вдосконалення архітектури, розширення функціональних можливостей та підвищення рівня

інтерактивності. Саме тому важливим етапом аналізу є визначення перспектив подальшого розвитку програмного продукту.

Одним із найбільш очевидних напрямів удосконалення є розширення системи штучного інтелекту. У поточній реалізації використовується базова модель поведінки ворогів, заснована на кінцевому автоматі станів та системі навігації NavMesh. Такий підхід забезпечує стабільну роботу та достатній рівень інтерактивності, проте в майбутньому система може бути доповнена більш складними алгоритмами поведінки.

Зокрема, перспективним напрямом є використання поведінкових дерев (Behavior Trees), які дозволяють створювати більш гнучкі та масштабовані моделі прийняття рішень. На відміну від кінцевих автоматів, поведінкові дерева краще підходять для реалізації складних сценаріїв взаємодії та значно спрощують підтримку великої кількості станів. Крім того, можливим є впровадження систем машинного навчання або адаптивного штучного інтелекту, який зможе змінювати поведінку залежно від дій користувача[16].

Іншим важливим напрямом розвитку є вдосконалення системи взаємодії з об'єктами. У поточній реалізації вона забезпечує базову взаємодію через Raycast та текстові підказки. Надалі система може бути розширена шляхом реалізації повноцінного інвентарю, системи предметів, механіки створення об'єктів (crafting), а також складніших сценаріїв взаємодії з навколишнім середовищем.

Перспективним є також розвиток системи збереження даних. На даний момент реалізовано базове збереження прогресу через JSON-файли, однак у майбутньому система може підтримувати:

- декілька слотів збереження;
- автоматичне збереження у контрольних точках;
- хмарну синхронізацію даних;
- шифрування файлів збереження;
- резервне копіювання прогресу.

Такі вдосконалення підвищать надійність системи та забезпечать кращий користувацький досвід.

Окрему увагу можна приділити розвитку користувацького інтерфейсу. У поточній версії інтерфейс реалізовано за принципами мінімалізму та функціональності, однак сучасні інтерактивні системи часто використовують анімовані елементи, адаптивний дизайн та складні візуальні ефекти. Подальший розвиток UI може включати:

- анімацію елементів інтерфейсу;
- систему налаштувань користувача;
- підтримку локалізації;
- інтерактивні меню;
- розширені HUD-елементи.

Це дозволить підвищити рівень комфорту та зробити взаємодію з програмним продуктом більш сучасною та привабливою.

Важливим напрямом є оптимізація продуктивності. Хоча результати тестування показали стабільну роботу системи, при масштабуванні проєкту може виникнути необхідність у додатковій оптимізації. Зокрема, перспективним є:

- використання object pooling для повторного використання об'єктів;
- оптимізація роботи AI;
- зменшення навантаження на Update;
- використання асинхронного завантаження сцен;
- впровадження системи LOD для складних моделей.

Це дозволить підтримувати стабільну продуктивність навіть при збільшенні кількості об'єктів та складності сцени.

Ще одним важливим напрямом розвитку є розширення функціональності програмного продукту. У майбутньому система може бути доповнена:

- новими типами ворогів;
- системою рівнів;
- сюжетними елементами;
- системою завдань;
- кооперативним або мережевим режимом;
- підтримкою мобільних платформ або VR.

Особливо перспективним є впровадження багатокористувацького режиму, оскільки сучасні інтерактивні програмні продукти все частіше орієнтуються на мережеву взаємодію. Для цього можуть бути використані Unity Netcode або Photon Unity Networking[7].

Крім того, можливим є розвиток графічної складової проєкту. Використання сучасних можливостей Unity, таких як HDRP або URP, дозволить покращити якість освітлення, тіней та візуальних ефектів. Це підвищить рівень реалістичності та загальне сприйняття програмного продукту.

Не менш важливою є перспектива використання розробленої архітектури як основи для інших програмних продуктів. Завдяки модульній структурі система може бути адаптована для створення:

- навчальних симуляторів;
- VR-додатків;
- тренажерів;
- інтерактивних візуалізацій;
- ігрових проєктів різних жанрів[15].

Таким чином, розроблений програмний продукт має значний потенціал для подальшого розвитку та вдосконалення. Реалізована модульна архітектура забезпечує гнучкість системи та створює основу для впровадження нових технологій і функціональних можливостей. Це дозволяє розглядати створений програмний продукт не лише як завершений результат кваліфікаційної роботи,



але і як платформу для подальших досліджень та практичної розробки інтерактивних систем.

#### **4.4. Висновки до розділу**

В ході роботи було проведено тестування розробленого інтерактивного програмного продукту, виконано аналіз результатів його роботи та оцінено ефективність використаних технологічних рішень. Основною метою даного етапу стало підтвердження працездатності системи, перевірка стабільності функціонування всіх компонентів та визначення перспектив подальшого розвитку програмного продукту.

У процесі тестування перевірялась робота основних функціональних підсистем: керування персонажем, системи штучного інтелекту, механізму взаємодії з об'єктами, користувацького інтерфейсу та системи збереження даних. Отримані результати підтвердили, що всі компоненти працюють коректно та забезпечують стабільну взаємодію між собою без критичних помилок або конфліктів.

Особливу увагу було приділено перевірці системи керування персонажем. Використання компонентів CharacterController та Input System дозволило забезпечити плавне та стабільне керування, швидку реакцію на дії користувача та коректну роботу камери. Також підтверджено ефективність використання NavMesh та NavMeshAgent для реалізації навігації та поведінки ворогів.

Під час аналізу роботи штучного інтелекту встановлено, що реалізована модель на основі кінцевого автомата станів забезпечує коректну реакцію NPC на дії користувача. Система сприйняття, побудована на перевірці дистанції, кута огляду та Raycast, створює достатній рівень інтерактивності та реалістичності поведінки ворогів.

Результати тестування також підтвердили ефективність реалізованої системи взаємодії з об'єктами. Використання Raycast дозволило забезпечити точне визначення інтерактивних елементів сцени та реалізувати зручний механізм взаємодії без перевантаження інтерфейсу.

Оцінка системи збереження даних показала, що використання JSON-серіалізації забезпечує швидке та стабільне збереження стану гри. Система коректно відновлює прогрес користувача та обробляє ситуації, пов'язані з відсутністю або пошкодженням файлів збереження.

Окремо проводилась оцінка продуктивності програмного продукту. За результатами тестування встановлено, що система стабільно працює із частотою понад 60 кадрів за секунду та не створює надмірного навантаження на апаратні ресурси. Це підтверджує ефективність використаної архітектури та достатній рівень оптимізації.

У межах розділу також визначено перспективи подальшого розвитку програмного продукту. Основними напрямками вдосконалення є розширення системи штучного інтелекту, впровадження поведінкових дерев, розвиток системи взаємодії, покращення інтерфейсу користувача, оптимізація продуктивності та реалізація багатокористувацького режиму.

Результати проведеного тестування підтвердили, що розроблений інтерактивний програмний продукт відповідає поставленим вимогам, забезпечує стабільну роботу всіх функціональних компонентів та має достатній потенціал для подальшого розвитку. Реалізована система може бути використана як основа для створення більш складних інтерактивних програмних рішень, а також для подальших досліджень у сфері розробки програмного забезпечення та ігрових технологій.

## ВИСНОВКИ

У результаті виконання роботи проведено комплексне дослідження сучасних підходів до створення інтерактивних програмних систем, проаналізовано особливості використання ігрових рушіїв та реалізовано власний програмний продукт із використанням сучасних технологій розробки програмного забезпечення.

Актуальність обраної теми обумовлена стрімким розвитком інтерактивних цифрових технологій, комп'ютерної графіки та систем реального часу. Сучасні програмні продукти дедалі частіше потребують реалізації складних механізмів взаємодії користувача із цифровим середовищем, адаптивного інтерфейсу, систем штучного інтелекту та оптимізованої архітектури. Особливо важливими є питання продуктивності, масштабованості та зручності використання, оскільки саме ці характеристики значною мірою визначають якість програмного забезпечення та рівень взаємодії користувача із системою.

У межах теоретичної частини роботи досліджено сучасний стан розвитку інтерактивних програмних продуктів, проаналізовано основні підходи до створення ігрових і симуляційних систем, а також розглянуто особливості компонентно-орієнтованої архітектури Unity. Встановлено, що використання ігрового рушія Unity є доцільним рішенням для створення інтерактивних систем завдяки широкому набору вбудованих інструментів, підтримці фізики, систем навігації, користувацького інтерфейсу та кросплатформеності.

Під час виконання роботи було проведено аналіз основних технологій, які використовуються для реалізації функціональних компонентів інтерактивних систем. Зокрема, було досліджено:

- принципи роботи системи NavMesh та можливості її використання для навігації NPC;
- особливості компонентів CharacterController та Rigidbody;

- сучасну систему введення Input System;
- механізми організації користувацького інтерфейсу в Unity;
- методи реалізації систем штучного інтелекту;
- підходи до створення систем взаємодії та збереження даних.

У процесі дослідження встановлено, що компонентно-орієнтований підхід, який використовується в Unity, дозволяє ефективно розділяти логіку системи на незалежні модулі. Це значно спрощує підтримку програмного продукту, дозволяє масштабувати систему та повторно використовувати окремі компоненти.

Практична частина роботи була присвячена проєктуванню та реалізації інтерактивного програмного продукту. На етапі проєктування сформовано загальну архітектуру системи, визначено взаємозв'язки між окремими компонентами та спроектовано основні функціональні модулі. У результаті реалізовано систему керування персонажем, яка забезпечує:

- плавне переміщення у тривимірному просторі.
- підтримку стрибків і гравітації.
- керування камерою від першої особи.
- обробку введення користувача.
- підтримку анімацій руху.

Для реалізації системи керування використано CharacterController та Input System. Такий підхід дозволив забезпечити стабільність руху персонажа та спростити обробку введення з різних типів пристроїв.

Окрему увагу приділено реалізації системи штучного інтелекту ворогів. У межах роботи створено систему поведінки NPC із використанням NavMeshAgent та кінцевого автомата станів. Реалізовано механізми:

- патрулювання.
- виявлення гравця.
- переслідування.
- нанесення пошкоджень.

- пошуку останньої позиції цілі.

Для системи сприйняття використовувались перевірка дистанції, кут огляду та Raycast, що дозволило створити базову модель «зору» NPC та забезпечити реалістичну реакцію ворогів на дії користувача.

Також реалізовано систему взаємодії з об'єктами, яка дозволяє користувачу активувати інтерактивні елементи сцени. Використання Raycast забезпечило простий та ефективний механізм визначення об'єктів у полі зору гравця. Додатково створено систему текстових підказок, яка покращує зручність використання програмного продукту.

Значну увагу приділено реалізації користувацького інтерфейсу. Створено HUD-елементи, систему відображення здоров'я та меню взаємодії. Інтерфейс розроблено з урахуванням принципів мінімалізму та адаптивності, що забезпечує комфортну взаємодію користувача із системою.

Для забезпечення цілісності ігрового процесу реалізовано систему збереження даних із використанням JSON-серіалізації, яка дозволяє зберігати:

- позицію персонажа.
- поточний стан гри.
- параметри здоров'я.
- інші важливі дані.

Система автоматично перевіряє наявність файлу збереження та відновлює попередній стан після запуску програми, що забезпечує зручність використання та дозволяє користувачу продовжити роботу після завершення попередньої сесії.

Після завершення реалізації проведено комплексне тестування програмного продукту. Перевірялися:

- стабільність роботи системи.
- коректність взаємодії компонентів.
- продуктивність.
- робота інтерфейсу.

- функціонування AI.
- система збереження даних.

Результати тестування показали, що всі функціональні компоненти працюють стабільно та коректно взаємодіють між собою. Система забезпечує плавне керування персонажем, коректну роботу штучного інтелекту та стабільну продуктивність навіть при одночасній роботі декількох підсистем. Оцінка ефективності реалізованих технологічних рішень підтвердила доцільність використання Unity як середовища розробки інтерактивних систем. Використання NavMesh дозволило спростити реалізацію навігації, CharacterController забезпечив стабільне переміщення персонажа, а Input System зробила систему введення більш гнучкою та сучасною. У процесі виконання роботи також було визначено перспективи подальшого розвитку програмного продукту. Основними напрямками вдосконалення можуть стати:

- використання поведінкових дерев.
- реалізація адаптивного AI.
- створення інвентарю.
- розширення системи взаємодії.
- впровадження мережевого режиму.
- підтримка VR та мобільних платформ.
- покращення графічної складової.
- оптимізація продуктивності великих сцен.

Отже, у результаті було досягнуто поставленої мети та виконано всі визначені завдання. Розроблений програмний продукт демонструє ефективність використання сучасних технологій Unity для створення систем реального часу та підтверджує можливість реалізації складних функціональних компонентів у межах компонентно-орієнтованої архітектури. Практична цінність роботи полягає у створенні працездатного програмного продукту, який може бути використаний як основа для подальших досліджень, навчальних проєктів або розробки більш складних інтерактивних систем.

Отримані результати також можуть бути використані під час вивчення дисциплін, пов'язаних із програмуванням, комп'ютерною графікою, розробкою ігрових систем та програмною інженерією.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Глинський Я. М. Основи програмування мовою C# / Я. М. Глинський. – Львів : СПД Глинський, 2021. – 320 с.
2. Ковальчук В. В. Об'єктно-орієнтоване програмування : навчальний посібник / В. В. Ковальчук. – Київ : Видавництво Ліра-К, 2020. – 256 с.
3. Unity Technologies. Unity User Manual 6.3 LTS. – Режим доступу: Unity Documentation.
4. Unity Technologies. Navigation and Pathfinding in Unity. – Режим доступу: Unity NavMesh Documentation.
5. Unity Technologies. Character Controller Component. – Режим доступу: Unity CharacterController Documentation.
6. Unity Technologies. Input System Package. – Режим доступу: Unity Input System Documentation.
7. Gregory J. Game Engine Architecture / Jason Gregory. – 3rd ed. – Boca Raton : CRC Press, 2018. – 1136 p.
8. Nystrom R. Game Programming Patterns / Robert Nystrom. – USA : Genever Benning, 2014. – 354 p.
9. Шаховська Н. Б. Проектування інформаційних систем : навчальний посібник / Н. Б. Шаховська, В. В. Литвин. – Львів : Магнолія, 2021. – 380 с.
10. Pressman R. Software Engineering : A Practitioner's Approach / Roger Pressman, Bruce Maxim. – 9th ed. – New York : McGraw-Hill, 2019. – 880 p.
11. Gamma E. Design Patterns : Elements of Reusable Object-Oriented Software / Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. – Boston : Addison-Wesley, 1995. – 395 p.
12. Fowler M. UML Distilled : A Brief Guide to the Standard Object Modeling Language / Martin Fowler. – 3rd ed. – Boston : Addison-Wesley, 2004. – 208 p.
13. Тихомиров В. В. Розробка ігрових додатків на Unity : практичний курс / В. В. Тихомиров. – Харків : Факт, 2022. – 288 с.
14. Unity Technologies. Universal Render Pipeline Documentation.



15. Ткаченко О. М. Проєктування та розробка комп'ютерних ігор : навчальний посібник / О. М. Ткаченко. – Київ : КПП ім. Ігоря Сікорського, 2021. – 210 с.
16. Millington I. AI for Games / Ian Millington. – 3rd ed. – Boca Raton : CRC Press, 2019. – 922 p.
17. Nielsen J. Usability Engineering / Jakob Nielsen. – Boston : Academic Press, 1994. – 362 p.
18. Шмуллер Дж. Освой самостоятельно UML 2 за 24 часа / Джозеф Шмуллер. – Москва : Вільямс, 2005. – 416 с.

## 19. Додаток А.

Реалізація керування персонажем

```
using UnityEngine;
using UnityEngine.InputSystem;

[RequireComponent(typeof(CharacterController))]
[RequireComponent(typeof(Animator))]
[RequireComponent(typeof(PlayerInput))]
public class PlayerControllerWithCamera : MonoBehaviour
{
    [Header("Параметри руху")]
    public float walkSpeed = 3f;
    public float runMultiplier = 2f;
    public float jumpForce = 2f;
    public float gravity = -9.81f;
    public float smoothAnimationSpeed = 8f;

    [Header("Параметри камери")]
    public Transform playerCamera;
    public float mouseSensitivity = 2f;

    private CharacterController controller;
    private Animator animator;
    private PlayerInput playerInput;

    private InputAction moveAction;
    private InputAction lookAction;
    private InputAction jumpAction;
    private InputAction sprintAction;

    private float verticalVelocity;
    private float cameraPitch = 0f;
    private bool isJumping = false;
    private float turnValue = 0f;
    [Header("Pause")]
    public PauseManager pauseManager;

    void Start()
    {
        if (SaveSystem.instance != null)
        {
            transform.position = SaveSystem.instance.LoadGame();
        }
    }
}
```

```

}

void Awake()
{
    controller = GetComponent<CharacterController>();
    animator = GetComponent<Animator>();
    playerInput = GetComponent<PlayerInput>();

    moveAction = playerInput.actions["Move"];
    lookAction = playerInput.actions["Look"];
    jumpAction = playerInput.actions["Jump"];
    sprintAction = playerInput.actions["Sprint"]; // Shift

    if (playerCamera == null)
    {
        playerCamera = GetComponentInChildren<Camera>()?.transform;
        if (playerCamera == null)
            Debug.LogError("PlayerCamera не найдена!");
    }

    Cursor.lockState = CursorLockMode.Locked;
    Cursor.visible = false;
}

void Update()
{
    HandleMovement();
    HandleLook();
}

void HandleMovement()
{
    if (moveAction == null || controller == null) return;

    Vector2 input = moveAction.ReadValue<Vector2>();
    bool isRunning = sprintAction != null && sprintAction.IsPressed();

    float targetSpeed = walkSpeed * (isRunning ? runMultiplier : 1f);
    Vector3 move = transform.right * input.x + transform.forward * input.y;

    // Гравітація и стрибок
    if (controller.isGrounded)
    {

```

```

        if (verticalVelocity < 0)
            verticalVelocity = -2f;

        if (jumpAction != null && jumpAction.triggered)
        {
            verticalVelocity = jumpForce;
            isJumping = true;
            animator.SetTrigger("Jump");
        }
    }
    else
    {
        verticalVelocity += gravity * Time.deltaTime;
    }

    if (controller.isGrounded && isJumping && verticalVelocity < 0)
    {
        isJumping = false;
    }

    move.y = verticalVelocity;
    controller.Move(move * targetSpeed * Time.deltaTime);

    // ---- Анімації ----
    float inputMagnitude = Mathf.Clamp01(input.magnitude);
    float targetAnimationSpeed = inputMagnitude * (isRunning ? 2f : 1f);
    float currentSpeed = Mathf.Lerp(animator.GetFloat("Speed"),
targetAnimationSpeed, Time.deltaTime * smoothAnimationSpeed);

    float direction = 0f;
    if (input.x > 0.1f) direction = 1f;
    else if (input.x < -0.1f) direction = -1f;

    animator.SetFloat("Speed", currentSpeed);
    animator.SetFloat("Direction", Mathf.Lerp(animator.GetFloat("Direction"),
direction, Time.deltaTime * smoothAnimationSpeed));
}

void HandleLook()
{
    if (pauseManager.isPaused) return;
    if (lookAction == null || playerCamera == null)
        return;

```

```

Vector2 lookInput = lookAction.ReadValue<Vector2>();
float mouseX = lookInput.x * mouseSensitivity;
float mouseY = lookInput.y * mouseSensitivity;

// Поворот камери (вверх/вниз)
cameraPitch -= mouseY;
cameraPitch = Mathf.Clamp(cameraPitch, -80f, 80f);
playerCamera.localEulerAngles = Vector3.right * cameraPitch;

// Поворот гравця (вліво/вправо)
transform.Rotate(Vector3.up * mouseX);

// ---- Анімація повороту ----
float speed = animator.GetFloat("Speed");
float targetTurn = 0f;

if (speed < 0.1f)
{
    // Нормалізація руху миші
    targetTurn = Mathf.Clamp(mouseX / 8f, -1f, 1f);
}

if (Mathf.Abs(mouseX) < 0.05f)
    targetTurn = 0f;

turnValue = Mathf.Lerp(turnValue, targetTurn, Time.deltaTime * 10f);

turnValue = Mathf.Clamp(turnValue, -1f, 1f);

animator.SetFloat("Turn", turnValue);
}

}

```

## Додаток В

Реалізація управлінням додатку

```
using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;
using System.IO;

public class GameManager : MonoBehaviour
{
    public static GameManager Instance;

    [Header("UI Screens")]
    public GameObject gameOverText;
    public GameObject youWonText;

    [Header("Save / Load References – заповни або знайдуться автоматично")]
    [SerializeField] private PlayerHealth playerHealth;
    [SerializeField] private Transform playerTransform;
    [SerializeField] private PickupSystem pickupSystem;

    [Header("Save Settings")]
    [SerializeField] private bool autoLoadOnStart = true;
    private string savePath;
    private bool gameEnded = false;

    void Awake()
    {
        if (Instance == null)
        {
            Instance = this;
            DontDestroyOnLoad(gameObject);
        }
        else
        {
            Destroy(gameObject);
            return;
        }

        savePath = Path.Combine(Application.persistentDataPath, "savegame.json");

        // Підписуємося на подію завантаження будь-якої сцени
        SceneManager.sceneLoaded += OnSceneLoaded;
    }
}
```

```

void OnDestroy()
{
    // Відписуємося, щоб не було витоків
    SceneManager.sceneLoaded -= OnSceneLoaded;
}

private void OnSceneLoaded(Scene scene, LoadSceneMode mode)
{
    // Завантажуємо дані тільки коли потрапляємо в ігрову сцену
    if (scene.name == "Main" || scene.name == "GameScene" || scene.name ==
"SampleScene")
    {
        // Знаходимо посилання на гравця (якщо не призначені в інспекторі)
        FindPlayerReferences();

        if (autoLoadOnStart && File.Exists(savePath))
        {
            LoadGame();
        }
    }
}

private void FindPlayerReferences()
{
    // Якщо посилання ще null – шукаємо за тегом або ім'ям
    if (playerTransform == null || playerHealth == null || pickupSystem == null)
    {
        GameObject playerObj = GameObject.FindWithTag("Player"); // ← постав тег
"Player" на MainChar!
        // або GameObject.Find("MainChar");

        if (playerObj != null)
        {
            playerTransform = playerObj.transform;
            playerHealth = playerObj.GetComponent<PlayerHealth>();
            pickupSystem = playerObj.GetComponent<PickUpSystem>();
        }

        if (playerTransform == null) Debug.LogError("Не знайдено гравця
(playerTransform)!");
        if (playerHealth == null) Debug.LogError("Не знайдено PlayerHealth!");
    }
}

```

```

        if (pickUpSystem == null) Debug.LogWarning("PickUpSystem не знайдено –
лічильник предметів не завантажиться");
    }
}

public void SaveGame()
{
    FindPlayerReferences();

    if (playerHealth == null || playerTransform == null)
    {
        Debug.LogWarning("Не призначено/не знайдено PlayerHealth або
playerTransform!");
        return;
    }

    SaveData data = new SaveData
    {
        playerHealth = playerHealth.health,
        collectedItems = pickUpSystem != null ? pickUpSystem.collectedCount : 0,
        playerPosX = playerTransform.position.x,
        playerPosY = playerTransform.position.y,
        playerPosZ = playerTransform.position.z,
        playerRotY = playerTransform.eulerAngles.y
    };

    string json = JsonUtility.ToJson(data, true);
    File.WriteAllText(savePath, json);
    Debug.Log($"Збережено в: {savePath}\nДані: {json}");
}

public void LoadGame()
{
    if (!File.Exists(savePath))
    {
        Debug.Log("Файл збереження не знайдено: " + savePath);
        return;
    }

    string json = File.ReadAllText(savePath);
    Debug.Log("Зчитаний JSON:\n" + json);

    SaveData data = JsonUtility.FromJson<SaveData>(json);

```



```

        Debug.Log($"Завантажено дані: HP={data.playerHealth},
Items={data.collectedItems},
Pos={data.playerPosX:F2},{data.playerPosY:F2},{data.playerPosZ:F2},
RotY={data.playerRotY:F2}");

        FindPlayerReferences();

        if (playerHealth != null)
        {
            playerHealth.health = data.playerHealth;
            Debug.Log($"Здоров'я встановлено на {playerHealth.health}");
        }
        else
        {
            Debug.LogError("playerHealth == null при завантаженні!");
        }

        if (pickUpSystem != null)
        {
            pickUpSystem.collectedCount = data.collectedItems;
            Debug.Log($"Предметів завантажено: {pickUpSystem.collectedCount}");
        }
        else
        {
            Debug.LogWarning("pickUpSystem == null - лічильник не завантажено");
        }

        if (playerTransform != null)
        {
            CharacterController cc =
playerTransform.GetComponent<CharacterController>();

            if (cc != null)
            {
                cc.enabled = false;
                Debug.Log("CharacterController вимкнено для телепортації");
            }

            // Встановлюємо позицію та поворот
            playerTransform.position = new Vector3(data.playerPosX, data.playerPosY,
data.playerPosZ);

```

```

        Vector3 currentEuler = playerTransform.eulerAngles;
        playerTransform.eulerAngles = new Vector3(currentEuler.x,
data.playerRotY, currentEuler.z);

        if (cc != null)
        {
            cc.enabled = true;

            // Це ключовий "поштовх", щоб CC підхопив нову позицію
            cc.Move(Vector3.zero); // або cc.Move(new Vector3(0, -0.001f, 0));
якщо є проблеми з землею

            Debug.Log("CharacterController увімкнено + cc.Move(Vector3.zero)
виконано. Позиція: " + playerTransform.position);
        }

        Debug.Log($"Позиція гравця завантажена та застосована:
{playerTransform.position}");
    }
    else
    {
        Debug.LogError("playerTransform == null - позицію не завантажено!");
    }
}

public void DeleteSave()
{
    if (File.Exists(savePath))
    {
        File.Delete(savePath);
        Debug.Log("Збереження видалено");
    }
}

public void GameOver()
{
    if (gameEnded) return;
    gameEnded = true;
    if (gameOverText != null) gameOverText.SetActive(true);
    StartCoroutine(EndGameRoutine());
}

```

```

public void WinGame()
{
    if (gameEnded) return;
    gameEnded = true;
    if (youWonText != null) youWonText.SetActive(true);
    StartCoroutine(EndGameRoutine());
}

IEnumerator EndGameRoutine()
{
    yield return new WaitForEndOfFrame();
    Time.timeScale = 0f;
    ClearScene();
}

void ClearScene()
{
    GameObject[] allObjects = FindObjectsOfType<GameObject>();
    foreach (GameObject obj in allObjects)
    {
        if (obj.GetComponent<Canvas>()) continue;
        if (obj == this.gameObject) continue;
        if (obj == gameOverText) continue;
        if (obj == youWonText) continue;
        if (obj.GetComponent<Camera>()) continue;
        Destroy(obj);
    }
}

public void GoToGame()
{
    SceneManager.LoadScene("Main");
}

public void QuitGame()
{
    Application.Quit();
}
}

```