ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ І ТЕХНОЛОГІЙ

# ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

*бакалавра*
(освітньо-кваліфікаційний рівень)

на тему *Розробка системи адаптивного штучного інтелекту для NPC у відеоіграх*

Виконав: здобувач *4* року навчання
групи *4ПР1*
напряму підготовки (спеціальності)
*121-Інженерія програмного забезпечення*
(шифр і назва напряму підготовки, спеціальності)
*Леба Данило Сергійович*
(підпис, прізвище та ініціали)

Керівник *Боскін О.О.*
(підпис, прізвище та ініціали)

Рецензент *Корніловська Н.В.*
(прізвище та ініціали)

Нормоконтролер *Комісаров О. С.*
(підпис, прізвище та ініціали)

Хмельницький – 2026

ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРЗВТЕТ

Інститут, факультет, відділення *Інформаційних технологій та дизайну*
Кафедра, циклова комісія *Програмних засобів і технологій*
Освітньо-кваліфікаційний рівень *бакалавр*
Освітньо-професійна підготовка *Програмна інженерія*
(шифр і назва)
Спеціальність *121-Інженерія програмного забезпечення*
(шифр і назва)

**ЗАТВЕРДЖУЮ**
**Завідувач кафедри** *програмних засобів і технологій*
_____ О.Є. Огнєва
«___» _____ 202_ року

# ЗАВДАННЯ

## НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

*Леби Данила Сергійовича*
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *«Розробка системи адаптивного штучного інтелекту для NPC у відеоіграх»*

керівник проекту (роботи) ст. викладач Боскін О.О.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «16» січня 2026 року №82-с

2. Строк подання здобувачем проекту(роботи) 24.06.2026 року

3. Вихідні дані до проекту (роботи) літературні та періодичні джерела, матеріали переддипломної практики

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження та аналіз предметної області
Проектування програмного забезпечення
Розробка програмного забезпечення
Тестування та впровадження програмного забезпечення

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

6. Консультанти розділів проекту (роботи)

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

7. Дата видачі завдання *10.02.2026*

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної роботи | Строк виконання етапів проекту (роботи) | Примітка |
|---|---|---|---|
| 1 | Отримання завдання | 10.02.2026 | Виконано |
| 2 | Підбір літератури | 11.02.2026-20.02.2026 | Виконано |
| 3 | Аналіз предметної області | 21.02.2026-28.02.2026 | Виконано |
| 4 | Розробка та обґрунтування завдання | 01.03.2026-10.03.2026 | Виконано |
| 5 | Розробка концептуальної моделі | 11.03.2026-20.03.2026 | Виконано |
| 6 | Моделювання та проектування системи | 21.03.2026-10.04.2026 | Виконано |
| 7 | Тестування гри | 11.04.2026-18.04.2026 | Виконано |
| 8 | Оформлення пояснювальної записки | 19.04.2026-26.05.2026 | Виконано |
| 9 | Захист кваліфікаційної роботи | 24.06.2026 | Виконано |

**Здобувач** _____ *Д.С. Леба*
(підпис) (прізвище та ініціали)

**Керівник проекту (роботи)** _____ *О.О. Боскін*
(підпис) (прізвище та ініціали)

# РЕФЕРАТ

Кваліфікаційна робота бакалавра: 62 сторінки, 19 рисунків, 5 таблиць, 20 джерел та 1 додаток.

Об’єкт дослідження – процес розроблення 3D ігрового програмного засобу з адаптивною поведінкою неігрових персонажів.

Предмет дослідження – методи, моделі та програмні засоби реалізації адаптивного штучного інтелекту для NPC у 3D міні-грі, зокрема механізми репутації, поширення чуток, зміни реакцій персонажів і визначення сюжетної лінії.

Мета роботи – розробити 3D міні-гру з адаптивним штучним інтелектом для NPC, у якій поведінка неігрових персонажів змінюється залежно від дій гравця, рівня репутації, поширення чуток та поточної сюжетної лінії.

У роботі проведено аналіз предметної галузі та виконано огляд існуючих підходів до реалізації адаптивного штучного інтелекту у відеоіграх. Обґрунтовано вибір ігрового рушія Unity та мови програмування C#. Визначено функціональні та нефункціональні вимоги до програмного засобу, спроєктовано архітектуру системи, структуру основних модулів і взаємодію між ними. Побудовано UML-діаграми, які відображають варіанти використання системи, структуру класів, взаємодію компонентів та логіку поведінки NPC.

У межах практичної частини реалізовано програмний прототип 3D міні-гри за мотивами авторського сценарію «Човняр: темні води Херсона», у якому центральну роль відіграють моральний вибір гравця, система локальної репутації, поширення чуток між NPC та перехід між гарною, нейтральною і поганою сюжетними лініями. Реалізовано механізми переміщення гравця, взаємодії з NPC, вибору варіантів звернення, зміни реакції персонажів і візуального відображення їхнього стану. Проведено тестування основних компонентів системи та проаналізовано результати їх роботи.

Практична значущість роботи полягає в тому, що розроблений програмний засіб може бути використаний як прототип для подальшого створення

повноцінної сюжетно-орієнтованої гри, а також як навчальний приклад реалізації адаптивного штучного інтелекту для NPC у середовищі Unity. Отримані результати можуть бути застосовані під час розроблення ігрових застосунків, дослідження поведінкових моделей неігрових персонажів та вдосконалення механік взаємодії гравця з ігровим середовищем.

КЛЮЧОВІ СЛОВА: UNITY, C#, 3D МІНІ-ГРА, АДАПТИВНИЙ ШТУЧНИЙ ІНТЕЛЕКТ, NPC, РЕПУТАЦІЯ, ЧУТКИ, СЮЖЕТНА ЛІНІЯ, ВЗАЄМОДІЯ З NPC, ІГРОВИЙ ПРОТОТИП.

# АНОТАЦІЯ

У кваліфікаційній роботі бакалавра розглянуто розроблення 3D міні-гри з адаптивним штучним інтелектом для неігрових персонажів. Метою роботи є створення програмного прототипу, у якому поведінка NPC змінюється залежно від дій гравця, рівня репутації, поширення чуток, фракційної належності персонажів і поточної сюжетної лінії.

У роботі проаналізовано основні підходи до моделювання поведінки NPC: скінченні автомати, дерева поведінки, GOAP, Utility AI та методи машинного навчання. Для практичної реалізації обрано гібридний rule-based підхід, який поєднує станну модель, локальну та фракційну репутацію, систему чуток і контекстне оцінювання реакцій персонажів.

Програмний засіб розроблено в середовищі Unity із використанням мови програмування C#. Реалізовано модулі переміщення гравця, взаємодії з NPC, оновлення репутації, поширення чуток, визначення сюжетної лінії та візуального відображення станів персонажів. Практичну частину виконано у вигляді прототипу 3D міні-гри за мотивами авторського сценарію «Човняр: темні води Херсона». Залежно від рішень гравця NPC можуть змінювати своє ставлення, надавати правдиві або хибні підказки, відмовляти в допомозі чи демонструвати ворожу реакцію.

Проведено модульне, інтеграційне та UI-тестування програмного засобу. Результати підтвердили коректність роботи основних механізмів адаптивного штучного інтелекту. Розроблений прототип може бути використаний як основа для подальшого створення сюжетно-орієнтованої гри та як навчальний приклад реалізації адаптивної поведінки NPC у Unity.

**Ключові слова:** UNITY, C#, NPC, АДАПТИВНИЙ ШТУЧНИЙ ІНТЕЛЕКТ, 3D МІНІ-ГРА, РЕПУТАЦІЯ, ЧУТКИ, СЮЖЕТНА ЛІНІЯ, ІГРОВИЙ ПРОТОТИП.

## ABSTRACT

The bachelor's qualification thesis focuses on the development of a 3D mini-game with adaptive artificial intelligence for non-player characters. The aim of the work is to create a software prototype in which NPC behavior changes depending on the player's actions, reputation level, rumor propagation, character faction, and the current storyline.

The thesis analyzes the main approaches to NPC behavior modeling, including finite-state machines, behavior trees, GOAP, Utility AI, and machine learning methods. For practical implementation, a hybrid rule-based approach was selected. It combines a state-based model, local and faction reputation, a rumor system, and context-based evaluation of NPC reactions.

The software was developed in Unity using the C# programming language. The implemented modules provide player movement, interaction with NPCs, reputation updates, rumor propagation, storyline evaluation, and visual representation of character states. The practical part was implemented as a prototype of a 3D mini-game based on the original scenario "Ferryman: Dark Waters of Kherson." Depending on the player's decisions, NPCs can change their attitude, provide true or false hints, refuse to help, or demonstrate hostile behavior.

Unit, integration, and UI testing of the software were conducted. The results confirmed the correct operation of the main adaptive artificial intelligence mechanisms. The developed prototype can be used as a basis for the further creation of a story-oriented game and as an educational example of implementing adaptive NPC behavior in Unity.

**Keywords:** UNITY, C#, NPC, ADAPTIVE ARTIFICIAL INTELLIGENCE, 3D MINI-GAME, REPUTATION, RUMORS, STORYLINE, GAME PROTOTYPE.

# ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ ..... 10

ВСТУП ..... 11

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ... 14

1.1. Опис та загальна характеристика предметної області ..... 14

1.2. Огляд та аналіз існуючих рішень ..... 15

1.3. Огляд методів та технологій ..... 18

1.4. Постановка задачі та визначення вимог ..... 19

1.5. Висновки до розділу 1 ..... 21

РОЗДІЛ 2. Проектування програмного забезпечення ..... 23

2.1. Вибір архітектури програмного засобу ..... 23

2.2. Архітектура програмного засобу та варіанти використання системи ..... 24

2.2.1. Загальна архітектура програмного засобу ..... 24

2.2.2. Варіанти використання системи ..... 26

2.3. Проєктування структури даних ..... 28

2.4. Проєктування поведінки системи ..... 30

2.5. Проєктування інтерфейсу користувача ..... 32

2.6. Висновки до розділу 2 ..... 34

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ..... 36

3.1. Середовище та інструменти розробки ..... 36

3.2. Реалізація ключових компонентів системи ..... 38

3.3. Реалізація збереження стану, конфігурації та налагодження системи ..... 42

3.4. Інтеграція підсистем та внутрішні інтерфейси взаємодії ..... 44

3.5. Висновок до розділу 3 ..... 46
РОЗДІЛ 4. Тестування та впровадження програмного забезпечення.... 47
4.1. Підхід до модульного тестування..... 47
4.2. Інтеграційне та UI-тестування ..... 49
4.3. Аналіз результатів тестування ..... 51
4.4. Інструкція користувача ..... 54
4.5. Локальне розгортання та запуск прототипу..... 55
4.6. Висновок до розділу 4 ..... 56
ВИСНОВКИ ..... 57
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ ..... 59

# ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

AI (*Artificial Intelligence*) – штучний інтелект.

API (*Application Programming Interface*) – прикладний програмний інтерфейс.

FSM (*Finite State Machine*) – скінченний автомат.

GOAP (*Goal-Oriented Action Planning*) – планування дій, орієнтоване на досягнення цілей.

HTTP (*Hypertext Transfer Protocol*) – протокол передавання гіпертексту.

IDE (*Integrated Development Environment*) – інтегроване середовище розробки.

JSON (*JavaScript Object Notation*) – текстовий формат обміну та зберігання даних.

LTS (*Long-Term Support*) – версія програмного забезпечення з довгостроковою підтримкою.

NPC (*Non-Player Character*) – неігровий персонаж, поведінкою якого керує програмна логіка.

ORM (*Object-Relational Mapping*) – технологія об’єктно-реляційного відображення даних.

REST (*Representational State Transfer*) – архітектурний стиль взаємодії програмних систем.

UI (*User Interface*) – інтерфейс користувача.

UML (*Unified Modeling Language*) – уніфікована мова моделювання.

ШІ – штучний інтелект.

# ВСТУП

У сучасних умовах стрімкого розвитку цифрових технологій та індустрії відеоігор особливої актуальності набуває створення програмних засобів, у яких неігрові персонажі здатні не лише виконувати заздалегідь визначені сценарії, а й адаптувати свою поведінку до дій користувача. Для сюжетно-орієнтованих ігор така властивість є особливо важливою, оскільки саме вона забезпечує відчуття динамічного світу, підвищує рівень занурення гравця та формує індивідуальний досвід проходження. У зв’язку з цим розробка 3D міні-гри з адаптивним штучним інтелектом для NPC є актуальною та практично значущою задачею.

Основою даної роботи є розроблення програмного засобу у вигляді 3D міні-гри за мотивами авторського сценарію «Човняр: темні води Херсона», у якому події розгортаються в умовах затоплених територій Херсонщини після підриву Каховської ГЕС, а поведінка персонажів і розвиток сюжетної лінії залежать від рішень гравця. У сценарії закладено систему морального вибору, репутації, довіри та взаємодії з різними фракціями, що робить його придатною основою для дослідження адаптивного AI у відеоіграх.

Актуальність теми полягає в тому, що більшість сучасних ігрових проєктів прагнуть забезпечити не лише візуальну привабливість, а й правдоподібну поведінку неігрових персонажів. NPC, які реагують на дії гравця, змінюють своє ставлення до нього, поширюють інформацію про події та впливають на подальший перебіг сюжету, формують більш глибокий і реалістичний ігровий досвід. У межах даної роботи реалізація такої поведінки є центральною ідеєю програмного засобу, оскільки саме адаптивність NPC визначає перехід між гарною, нейтральною та поганою сюжетними лініями.

Метою роботи є розробка 3D міні-гри з адаптивним штучним інтелектом для NPC, яка забезпечує зміну поведінки неігрових персонажів залежно від дій гравця, рівня репутації, поширення чуток та поточної сюжетної лінії.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– проаналізувати предметну галузь та сучасні підходи до реалізації адаптивного AI у відеоіграх;

- обґрунтувати вибір інструментів і технологій розробки, зокрема рушія Unity та мови програмування C#;
- спроєктувати архітектуру програмного засобу та структуру його основних модулів;
- розробити систему взаємодії гравця з NPC;
- реалізувати механізми репутації, поширення чуток і визначення сюжетної лінії;
- забезпечити відображення реакцій NPC на дії гравця;
- провести тестування розробленого програмного засобу та проаналізувати результати його роботи.

Об’єкт дослідження – програмні засоби у сфері відеоігор з адаптивною поведінкою неігрових персонажів.

Предмет дослідження – методи, засоби та технології розробки 3D ігрового застосунку з адаптивним штучним інтелектом для NPC, системою репутації та нелінійною сюжетною взаємодією.

Методи дослідження. Під час виконання роботи використовувалися методи аналізу предметної галузі, проєктування програмного забезпечення, об’єктно-орієнтованого програмування, моделювання поведінки NPC, побудови компонентної архітектури в середовищі Unity, а також методи модульного та інтеграційного тестування. Для практичної реалізації застосовано рушій Unity, мову програмування C#, механізми MonoBehaviour, ScriptableObject, логіку взаємодії між компонентами, а також UML-діаграми для проєктування структури та поведінки системи.

Мета і задачі дослідження полягають у створенні програмного прототипу, який демонструє можливість реалізації адаптивної поведінки NPC у 3D міні-грі. Розроблений застосунок повинен забезпечувати переміщення персонажа, взаємодію з NPC, вибір варіантів звернення, зміну ставлення неігрових персонажів до гравця, а також відображення переходу між різними сюжетними лініями залежно від морального вибору.

Наукова новизна одержаних результатів полягає в розробленні прототипу 3D міні-гри, у якій реалізовано адаптивний підхід до поведінки NPC на основі

поєднання системи локальної репутації, чуток, фракційної належності та контекстної взаємодії з гравцем. На відміну від статичних моделей поведінки, запропонований підхід дозволяє змінювати реакції NPC відповідно до накопичених дій користувача та поточного стану сюжетної лінії.

Практичне значення одержаних результатів полягає в тому, що створений програмний засіб може бути використаний як основа для подальшої розробки повноцінної сюжетно-орієнтованої гри, а також як демонстраційний приклад реалізації адаптивного AI у середовищі Unity. Отримані результати можуть бути корисними для навчальних цілей, дослідження ігрових механік та подальшого вдосконалення поведінкових моделей NPC.

Структура: робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел, додатків. Кваліфікаційна робота бакалавра складається 62 сторінок, 19 рисунків, 5 таблиць, 20 джерел та 1 додаток.

# РОЗДІЛ 1. ДОСЛІДЖЕННЯ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1. Опис та загальна характеристика предметної області

На відміну від статичних ігрових сценаріїв, у таких системах значущу роль відіграє не лише візуальне середовище та набір механік, а й здатність NPC реагувати на дії гравця, змінювати ставлення до нього, поширювати інформацію про події та впливати на подальший перебіг сюжету. У межах цієї роботи предметна галузь охоплює поєднання ігрового наративу, поведінкового моделювання NPC, системи репутації та механізмів морального вибору.

Основою проєкту є авторський сценарій гри «Човняр: темні води Херсона», у якому події розгортаються після підриву Каховської ГЕС у затоплених районах Херсона та прилеглих населених пунктах. Головний герой, Ігар Коваленко, шукає доньку Оленку. Сюжет побудовано як нелінійну систему, де допомога іншим, нейтральна поведінка або мародерство безпосередньо впливають на доступ до інформації, стосунки з NPC і кінцевий результат проходження. У документі сценарію прямо закладено механіку морального компаса, репутації та різних сюжетних ліній. Це робить придатною основою для дослідження адаптивного AI у відеоіграх.

Сюжетна основа програмного засобу базується на авторському сценарії гри “Човняр: темні води Херсона”, авторське право на який зареєстровано свідоцтвом № 138131 від 29.08.2025 року. За авторством Боскіна О.О, Голобородько О.С., Кулініч-Швець Д.Д. та Хижнякова Г.Г.[1]

Актуальність теми полягає в тому, що сучасні ігрові застосунки дедалі частіше вимагають від NPC не заздалегідь заскриптованої, а контекстно-залежної поведінки. Для сюжетно-орієнтованих ігор особливо важливо, щоб неігрові персонажі реагували на попередні дії гравця, змінювали доступність діалогів, допомоги, торгівлі чи сюжетних підказок. У сценарії «Човняр: темні води Херсона» така потреба проявляється безпосередньо: на «гарній» лінії герой отримує довіру й підказки, тоді як на «поганій» лінії NPC можуть уникати його, відмовляти в допомозі або свідомо вводити в оману. Через що

задача розроблення адаптивного штучного інтелекту для NPC є не допоміжною, а центральною для реалізації даного ігрового проєкту.

У контексті розроблюваного програмного засобу NPC розглядаються не лише як декоративні елементи ігрової сцени або джерела заздалегідь визначених реплік, а як програмні агенти, які мають власний внутрішній стан і здатні змінювати поведінку залежно від подій у навколишньому середовищі. До такого стану належать рівень довіри до головного героя, належність до певної фракції, наявність отриманих чуток, оцінка попередніх дій гравця та поточна сюжетна лінія. Сукупність цих параметрів дає змогу формувати контекстно-залежні реакції персонажів: надавати корисну інформацію, відмовляти в допомозі, демонструвати недовіру, поширювати відомості серед інших NPC або змінювати ставлення до героя після його вчинків.

Важливою особливістю предметної області є взаємозв’язок між локальними рішеннями користувача та загальним розвитком ігрового сюжету. Окрема дія гравця не повинна залишатися ізольованою подією. Її наслідки накопичуються в системі репутації, передаються між персонажами через механізм чуток і впливають на подальшу доступність діалогів та сюжетних підказок. Завдяки цьому формується модель динамічного ігрового середовища, у якому поведінка NPC змінюється поступово. Результат проходження залежить від послідовності моральних виборів користувача. Такий підхід відповідає загальній концепції прототипу, де передбачено локальну репутацію, поширення чуток і перехід між гарною, нейтральною та поганою сюжетними лініями.

### 1.2. Огляд та аналіз існуючих рішень

Під час проєктування адаптивного штучного інтелекту для NPC доцільно розглянути кілька поширених підходів до моделювання поведінки агентів. До найбільш відомих належать скінченні автомати (Finite State Machines), дерева поведінки (Behavior Trees), планування на основі цілей (GOAP), Utility AI та підходи на основі машинного навчання, зокрема reinforcement learning. Усі

вони по-різному вирішують проблему вибору дій NPC, мають різну складність реалізації, різний рівень пояснюваності та різну придатність до прототипування в межах навчального проєкту.

Скінченні автомати (FSM) є одним із найпростіших і найстаріших підходів для ігрового AI. Їхня перевага полягає у простоті реалізації та зрозумілій структурі переходів між станами, наприклад: neutral, suspicious, hostile, flee. Водночас зі зростанням кількості умов і подій FSM стають менш гнучкими, важче масштабуються та ускладнюють повторне використання логіки. Тому цей підхід добре працює для невеликих прототипів, але не завжди є достатнім як єдина архітектура для складної сюжетної взаємодії.[2]

Behavior Trees побудовані як ієрархічні дерева рішень і вважаються модульним та реактивним способом організації поведінки агентів. У дослідженнях з поведінкових дерев підкреслюється, що вони дають кращу модульність і розширюваність, ніж FSM. До того ж, спрощують повторне використання окремих поведінкових блоків. Для ігрового застосування це означає, що NPC легше описувати через ієрархії дій: спостереження, вибір реакції, пересування, втеча, взаємодія тощо.[3]

GOAP (Goal-Oriented Action Planning) орієнтований не на жорстко визначений ланцюг переходів, а на побудову плану дій для досягнення цілі. У роботах Jeff Orkin GOAP описано як архітектуру прийняття рішень, що дозволяє агенту обирати не лише наступну дію, а й спосіб досягнення бажаного стану світу. Тому такий підхід особливо корисний у динамічних середовищах, де NPC має адаптуватися до змін контексту. Однак для невеликого прототипу GOAP часто виявляється складнішим в реалізації й налагодженні, ніж більш локальні rule-based системи.[4]

Utility AI оцінює можливі дії через систему корисності. Кожна потенційна дія отримує числову оцінку, а агент обирає варіант з найкращою поточною доцільністю. Такий підхід добре підходить для ситуацій, де потрібно уникнути різких переходів типу «так/ні» та плавніше враховувати контекст: рівень довіри, небезпеку, цінність ресурсу, емпатію NPC, локальну репутацію

героя. Саме тому Utility AI є особливо цікавим для систем, де поведінка персонажа залежить від сукупності багатьох факторів, а не від однієї події.[5]

Підходи на основі машинного навчання, зокрема reinforcement learning, в Unity можуть бути реалізовані через ML-Agents Toolkit. Офіційна документація Unity зазначає, що ML-Agents дозволяє перетворювати Unity-сцени на навчальні середовища й тренувати поведінку агентів за допомогою алгоритмів машинного навчання, після чого інтегрувати навчені моделі назад у гру. Водночас для кваліфікаційної роботи бакалавра такий підхід має суттєві витрати на підготовку середовища, налаштування навчання, збір достатніх сценаріїв і контроль відтворюваності результатів.[6]

З огляду до нашої проблематики доцільно обрати гібридний rule-based підхід, який поєднує просту станну модель NPC, систему локальної репутації, поширення чуток і контекстний вибір реакції на основі набору вагових правил. Це є кращим саме для навчального прототипу, тому що він достатньо прозорий для аналізу, легко відлагоджується, не потребує довготривалого навчання моделей, добре узгоджується з сюжетною логікою «гарної» і «поганої» ліній та дозволяє поступово нарощувати складність без повного переписування системи.

Тому гібридний rule-based підхід є ближчим до поєднання FSM + Utility-style evaluation, ніж до повного GOAP або reinforcement learning.

При виборі підходу до реалізації адаптивної поведінки NPC важливо враховувати не лише функціональні можливості окремих рішень, а й особливості конкретного ігрового проєкту. Для сюжетно-орієнтованої 3D міні-гри визначальними критеріями є прозорість логіки прийняття рішень. Це є можливістю швидкого налагодження, передбачуваність реакцій персонажів і простота подальшого розширення системи. У межах прототипу необхідно забезпечити залежність поведінки NPC від кількох взаємопов’язаних факторів: локальної репутації гравця, належності персонажа до певної фракції, отриманих чуток, рівня довіри та поточної сюжетної лінії. Так обране рішення

повинно дозволяти комбінувати різні умови без надмірного ускладнення програмної реалізації.

Порівняння розглянутих підходів показує, що використання лише одного методу може бути недостатнім для досягнення поставленої мети. Чистий FSM є зручним для опису базових станів NPC, однак зі збільшенням кількості умов його структура стає складнішою. Behavior Trees краще підходять для ієрархічної організації поведінки, але для невеликого прототипу можуть вимагати додаткової інфраструктури. GOAP та reinforcement learning надають ширші можливості для побудови складних моделей поведінки, проте ускладнюють налагодження та перевірку результатів. В той же час, поєднання станної моделі з ваговим оцінюванням контексту дає змогу зберегти зрозумілу логіку переходів і водночас враховувати сукупний вплив дій гравця на реакції NPC.

### 1.3. Огляд методів та технологій

Для реалізації програмного засобу обрано ігровий рушій Unity та мову програмування C#. Unity 2022.3 LTS є придатною основою для розроблення 2D і 3D ігор. Він має офіційний User Manual, Scripting API та набір стандартних інструментів редактора для роботи зі сценами, фізикою, матеріалами, UI, анімацією та скриптами. Крім того, Unity позиціонується як рушій для створення багатоплатформових ігрових застосунків, що є важливою перевагою з точки зору подальшого розвитку проєкту за межами дипломного прототипу.[7][8]

Вибір саме Unity обґрунтовується кількома чинниками. По-перше, цей рушій дає змогу швидко створювати прототипи, що особливо важливо для навчального проєкту, де необхідно спочатку перевірити логіку адаптивного AI, а вже потім переходити до деталізованої графіки. По-друге, Unity має офіційно підтримуваний інструментарій для навігації агентів: у 2022.3 використовується пакет AI Navigation, який надає високорівневі NavMesh-

компоненти для побудови та використання навігації NPC. По-третє, Unity має розвинену екосистему документації, прикладів і пакетів, що істотно спрощує реалізацію прототипу та його подальше розширення.[9][10]

Для програмної реалізації логіки системи обрано C#, тому що це сучасна, відкрита, кросплатформна, об’єктно-орієнтована мова програмування платформи .NET. Документація Microsoft підкреслює її type safety, підтримку узагальнень, pattern matching, async-моделі та інші мовні засоби, які корисні для побудови структурованого й підтримуваного коду. Для даної роботи це має практичне значення. Поведінка NPC, система репутації, взаємодія з UI та логіка місії природно подаються через класи, перерахування, керуючі структури та подієві зв’язки.[11][12]

Обрання зв’язки Unity + C# також узгоджується з архітектурою прототипу, який уже був окреслений у межах практичної частини: використання ScriptableObject для профілів NPC, MonoBehaviour для поведінкових компонентів, простих 3D-примітивів для прототипування, NavMesh-навігації для переміщення агентів і C#-скриптів для реалізації систем репутації, чуток та сюжетних гілок. Зазначене поєднання є достатньо потужним для розроблення 3D міні-гри й водночас достатньо простим для демонстрації ключової дослідницької ідеї – адаптивного штучного інтелекту NPC.

### 1.4. Постановка задачі та визначення вимог

Метою випускної кваліфікаційної роботи є розроблення програмного засобу у вигляді 3D міні-гри, в якій реалізовано адаптивний штучний інтелект для неігрових персонажів(NPC). Практична реалізація має базуватися на сюжетному сценарії гри «Човняр: темні води Херсона» та зосереджуватися на демонстрації того, як NPC змінюють свою поведінку залежно від дій гравця, його репутації, локального контексту та обраної сюжетної лінії. Для прототипування як основну сцену доцільно використовувати місію «Олешки», оскільки саме вона найбільш виразно поєднує волонтерів, мародерів,

цивільних, хибні або правдиві підказки, а також моральний вибір між «гарною» та «поганою» лініями.

Потрібно розробити 3D програмний застосунок у середовищі Unity з використанням мови C#, який реалізує прототип сюжетно-орієнтованої гри з адаптивною поведінкою NPC на основі систем репутації, чуток, фракційної належності та контекстної взаємодії з гравцем.

До функціональних вимог програмного засобу доцільно віднести:

- можливість переміщення ігрового персонажа в межах сцени;
- наявність кількох типів NPC, зокрема волонтерів, мародерів і цивільних;
- можливість взаємодії гравця з NPC на близькій дистанції;
- наявність кількох варіантів звернення або вибору під час взаємодії;
- зміну реакції NPC залежно від локальної репутації героя;
- реалізацію поширення чуток як механізму передачі інформації між NPC;
- візуальне відображення поточного ставлення NPC до гравця;
- визначення поточної сюжетної лінії як гарної, нейтральної або поганої;
- надання NPC різних відповідей залежно від поточного стану системи;
- можливість подальшого розширення прототипу до повноцінної місійної сцени.

До нефункціональних вимог доцільно віднести:

- зрозумілість та наочність роботи системи адаптивного AI;
- модульність програмної реалізації;
- можливість налагодження та тестування окремих підсистем;
- прийнятну продуктивність на типовому персональному комп’ютері;
- розширюваність архітектури для майбутнього додавання нових NPC, діалогів, умов та місій;
- простоту інтерфейсу прототипу, достатню для демонстрації логіки взаємодії.

Оскільки не закладається окремий кібербезпековий або мережевий функціонал, вимоги безпеки не виділяються як окремий великий блок. Натомість доречно враховувати базові вимоги до коректності обробки станів,

стабільності виконання, відсутності критичних помилок під час взаємодії з NPC та збереження внутрішньої цілісності логіки переходів між сюжетними лініями.

До обмежень і умов реалізації слід віднести

- використання Unity 2022.3.10f1 як цільового середовища розроблення;
- використання C# як мови програмування;
- орієнтацію на прототип, а не на повноцінний комерційний реліз;
- використання спрощених 3D-об’єктів на ранньому етапі розроблення замість деталізованих моделей;
- акцент на демонстрації адаптивного AI, а не на візуальній завершеності продукту;
- реалізацію в межах однієї ключової сцени з можливістю подальшого розширення.

### 1.5. Висновки до розділу 1

В цьому розділі було досліджено та проведено аналіз предметної області на основі авторського сюжету гри.

Проведений аналіз показав, що для задачі розроблення адаптивного AI NPC у межах дипломного проєкту найбільш доцільним є гібридний rule-based підхід, що поєднує станну модель, локальну репутацію, систему чуток і вагове оцінювання варіантів поведінки. На відміну від reinforcement learning, такий підхід є простішим у реалізації та відтворюванні, а порівняно з чистим FSM – краще масштабується для сюжетно-орієнтованої взаємодії з NPC.

Вибір Unity як ігрового рушія та C# як основної мови програмування зумовлений потребою у швидкому прототипуванні, наявністю офіційно підтримуваних інструментів для створення 3D-сцен, UI та навігації NPC, а також зручністю реалізації компонентної архітектури програмного засобу. Таке поєднання технологій є достатнім для розроблення дипломного прототипу з адаптивним AI NPC і забезпечує можливість подальшого функціонального розширення проєкту.

У межах випускної кваліфікаційної роботи поставлено задачу розроблення 3D прототипу сюжетно-орієнтованої гри з адаптивним штучним інтелектом для NPC. Визначені функціональні та нефункціональні вимоги орієнтовані на демонстрацію зміни поведінки NPC залежно від морального вибору, репутації та контексту дій гравця. Обрані обмеження реалізації відповідають формату кваліфікаційної роботи та забезпечують практичну перевірку запропонованого підходу.

# РОЗДІЛ 2. ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 2.1. Вибір архітектури програмного засобу

Для реалізації випускної кваліфікаційної роботи обрано монолітну компонентно-орієнтовану архітектуру в межах одного застосунку, розробленого у середовищі Unity. Це доцільно, оскільки програмний засіб не передбачає розподіленої серверної інфраструктури, окремих мікросервісів або хмарного оброблення даних. Всі основні підсистеми функціонують локально в межах єдиної ігрової сцени та спільного середовища виконання.

Обрана архітектура враховує специфіку предметної області, оскільки розроблюваний програмний засіб є не класичною інформаційною системою. 3D ігровим застосунком із сюжетною логікою, моральним вибором та адаптивною поведінкою неігрових персонажів. У сценарії гри «Човняр: темні води Херсона» центральне місце займають система морального компаса, репутація героя, довіра з боку NPC, доступ до сюжетних підказок та різні варіанти розвитку подій залежно від дій гравця. Це означає, що архітектура програмного засобу повинна підтримувати часту взаємодію між підсистемами керування гравцем, системою NPC, логікою репутації, поширенням чуток і визначенням поточної сюжетної лінії.

Монолітна архітектура в цьому випадку є доцільною з кількох причин. По-перше, вона спрощує реалізацію прототипу, всі модулі функціонують у межах одного проєкту та можуть безпосередньо взаємодіяти між собою. По-друге, такий підхід добре підходить для навчального проєкту, де важливо не масштабування на велику кількість клієнтів, а наочна демонстрація логіки системи. По-третє, монолітна структура спрощує налагодження, тестування й поступове розширення функціоналу, що є важливим для поетапної розробки: спочатку у вигляді прототипу на простих фігурах, а потім у вигляді повноціннішої сцени.

На рівні внутрішньої організації застосунок має компонентно-орієнтовану структуру, характерну для Unity. Основні функції розподіляються

між окремими скриптами та компонентами, кожен із яких виконує чітко визначену роль. Наприклад, одні компоненти відповідають за переміщення гравця, інші – за виявлення NPC поряд. Окремі модулі керують репутацією, поширенням чуток, визначенням реакцій NPC і відображенням інформації в інтерфейсі. Це забезпечує модульність, локалізацію відповідальності та зручність подальшої модифікації окремих частин системи без потреби кардинально змінювати всю архітектуру.

З урахуванням поставленої задачі до ключових переваг обраної архітектури належать:

- простота реалізації в межах дипломного проєкту;
- можливість швидкого прототипування;
- зручність тестування окремих підсистем;
- модульність внутрішньої логіки;
- придатність до подальшого розширення;
- відповідність вимогам 3D ігрового застосунку з адаптивним AI.

Для реалізації програмного засобу доцільно використовувати монолітну компонентно-орієнтовану архітектуру, яка найкраще відповідає цілям випускної кваліфікаційної роботи, формату прототипу та обраним технологіям розроблення.

## 2.2. Архітектура програмного засобу та варіанти використання системи

### 2.2.1. Загальна архітектура програмного засобу

Архітектура програмного засобу побудована навколо взаємодії кількох основних функціональних модулів, які разом забезпечують логіку ігрового процесу та адаптивну поведінку NPC. На концептуальному рівні система складається з підсистем керування гравцем, взаємодії з NPC, зберігання і аналізу репутації, поширення чуток, вибору поведінки NPC, визначення поточної сюжетної лінії та інтерфейсного відображення результатів.

Центральним елементом системи є підсистема керування гравцем, яка забезпечує переміщення персонажа сценою та ініціює взаємодії з об’єктами середовища. Саме дії гравця запускають подальші зміни в поведінковій моделі NPC. Після наближення героя до неігрового персонажа активується підсистема взаємодії з NPC, яка визначає доступність діалогу, варіанти звернення та результат взаємодії.

Окреме місце займає підсистема адаптивного AI NPC, яка враховує фракційну належність персонажа, поточний стан довіри, локальну репутацію героя, а також отримані чутки про його дії. У результаті NPC може змінювати власну реакцію – від дружньої до ворожої. Це відповідає сюжетній логіці сценарію, де на «гарній» лінії герой отримує підказки та допомогу, а на «поганій» – стикається з недовірою, брехнею або агресією.

Для підтримки такої поведінки використовується підсистема репутації, яка накопичує інформацію про дії гравця в межах конкретної локації та щодо конкретних груп NPC. Вона зберігає локальну репутацію героя, окремі оцінки серед волонтерів, мародерів і цивільних, а також узагальнений напрям сюжетного дрейфу. Її робота безпосередньо впливає на подальші реакції NPC і доступність певних варіантів діалогу.

Із репутаційною підсистемою тісно пов’язана підсистема чуток, яка імітує поширення інформації між NPC. Після значущої дії гравця формується подія-чутка, яка може бути передана іншим персонажам і вплинути на їх ставлення до героя. Такий механізм є важливим для демонстрації адаптивного AI, так як зміна поведінки NPC стає наслідком не жорстко заскриптованої події, а накопичення соціального контексту в межах локації.

Ще одним важливим модулем є підсистема оцінювання сюжетної лінії, яка аналізує накопичені значення репутації та морального дрейфу та визначає, чи рухається гравець гарною, нейтральною чи поганою лінією. Цей модуль забезпечує зв’язок між локальними рішеннями гравця та загальною напрямленістю проходження.

Узагальнену взаємодію користувача із системою відображає інтерфейсний модуль, який повідомляє про доступність NPC поруч, дозволяє обирати варіанти звернення, виводить відповідь NPC та відображає поточний стан сюжетної лінії. На етапі прототипування інтерфейс має спрощений вигляд і слугує насамперед для демонстрації логіки роботи системи, а не для фінального художнього оформлення.

Отже, архітектура програмного засобу є багатокомпонентною внутрішньо, але монолітною за способом розгортання. Вона забезпечує поєднання ігрової логіки, поведінкової моделі NPC та сюжетної адаптації, що є необхідним для реалізації обраної теми дипломного проєкту.

### 2.2.2. Варіанти використання системи

Для опису функціонального призначення програмного засобу доцільно застосувати UML Use Case діаграму, яка відображає основні варіанти використання системи з точки зору користувача. У розроблюваному застосунку головним зовнішнім актором є гравець, який взаємодіє з системою через керування персонажем, дослідження сцени, ініціювання взаємодії з NPC та виконання дій, що впливають на репутацію й розвиток сюжету.

Основними варіантами використання системи є:

- запуск гри;
- керування персонажем;
- переміщення сценою;
- наближення до NPC;
- ініціювання взаємодії;
- вибір одного з варіантів звернення;
- отримання відповіді NPC;
- зміна репутації героя;
- формування або поширення чутки;

– зміна поточного стану NPC;
– зміна сюжетної лінії;
– отримання підказки або хибної інформації.

У межах застосунку ці варіанти використання не є ізольованими. Наприклад, сценарій взаємодії з NPC включає одразу кілька пов’язаних підпроцесів: гравець підходить до персонажа, відкриває меню взаємодії, обирає варіант звернення, після чого система змінює значення репутації, поширює відповідну чутку, перераховує реакції NPC та повертає відповідь, яка може змінити подальший перебіг проходження. Use Case діаграма(див. рис. 2.1) для цього програмного засобу має не лише показувати перелік функцій, а й підкреслювати логічний зв’язок між діями гравця та адаптивною поведінкою середовища.

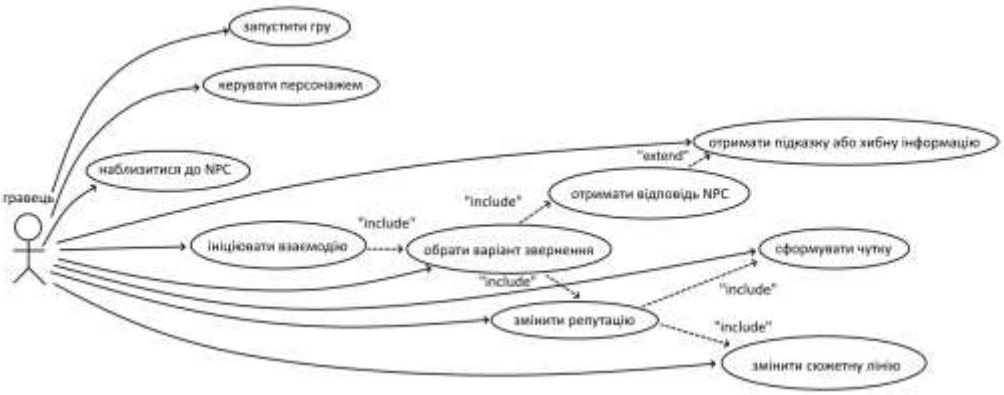



**Рисунок 2.1. Use Case діаграма програмного засобу**

Найбільш важливим варіантом використання є взаємодія з NPC, оскільки саме через нього в повній мірі проявляється адаптивний характер системи. Після вибору типу звернення – умовно доброзичливого або агресивного – неігровий персонаж надає різну відповідь залежно від своєї фракції, накопичених чуток, рівня довіри до героя та поточної сюжетної лінії. Це дозволяє відобразити одну з ключових особливостей сценарію – залежність доступу до інформації про Оленку та подальших шансів героя від його моральних рішень.

Ще одним важливим варіантом використання є зміна сюжетної лінії. Формально гравець не обирає її через окреме меню, проте своїми діями постійно впливає на її формування. Якщо він допомагає волонтерам, рятує цивільних або поводиться коректно у взаємодії, система схиляється до гарної лінії. Якщо ж він обирає мародерські, грубі або насильницькі дії, формується погана лінія. Саме такий механізм створює зв’язок між мікрорішеннями гравця та макрорезультатом проходження.

Use Case модель програмного засобу відображає його основне функціональне призначення: надати гравцеві можливість взаємодіяти з динамічним ігровим середовищем, у якому NPC адаптують свою поведінку відповідно до його дій, а сюжетна лінія змінюється внаслідок накопиченого морального вибору.

### 2.3. Проєктування структури даних

Так як, розроблюваний програмний засіб не передбачає використання реляційної бази даних, для опису структури даних доцільно застосувати UML Class Diagram, а не ER-діаграму. Основні сутності системи існують у вигляді класів C# і конфігураційних об’єктів Unity, зокрема ScriptableObject-профілів. Саме тому об’єктно-орієнтоване подання структури є найбільш природним для даного проєкту.

У центрі моделі даних знаходиться клас NpcMemory, який зберігає поточний стан NPC, посилання на профіль персонажа, відомі чутки та актуальну реакцію на гравця. Конфігураційні характеристики NPC зберігаються в класі NpcProfileSO, який використовується як зовнішній профіль і містить параметри емпатії, довіри, ворожості, можливості торгівлі та поширення чуток.

Реалізація адаптивної поведінки вимагає виділення окремих сервісних класів. ReputationManager відповідає за накопичення локальної та фракційної репутації, RumorSystem – за створення та поширення чуток, StoryPathEvaluator – за визначення поточної сюжетної лінії. Для реалізації взаємодії між гравцем

і NPC використовуються класи PlayerInteractionDetector, PlayerInteractionController, InteractableNpc та InteractionResult.

Такий розподіл чітко розмежовує дані, поведінку та службову логіку. Частина класів зберігає стан, частина виконує керуючі функції, а частина реалізує адаптивну зміну поведінки NPC. Як результат, формується структура, придатна до подальшого розширення без радикальної перебудови застосунку.

На рисунку 2.2 показано зв’язки між основними класами програмного засобу. Діаграма демонструє, що конфігураційні дані NPC зберігаються окремо від їхньої поточної пам’яті, а взаємодія з гравцем, система репутації та система чуток оформлені у вигляді незалежних, але взаємопов’язаних сервісних компонентів.

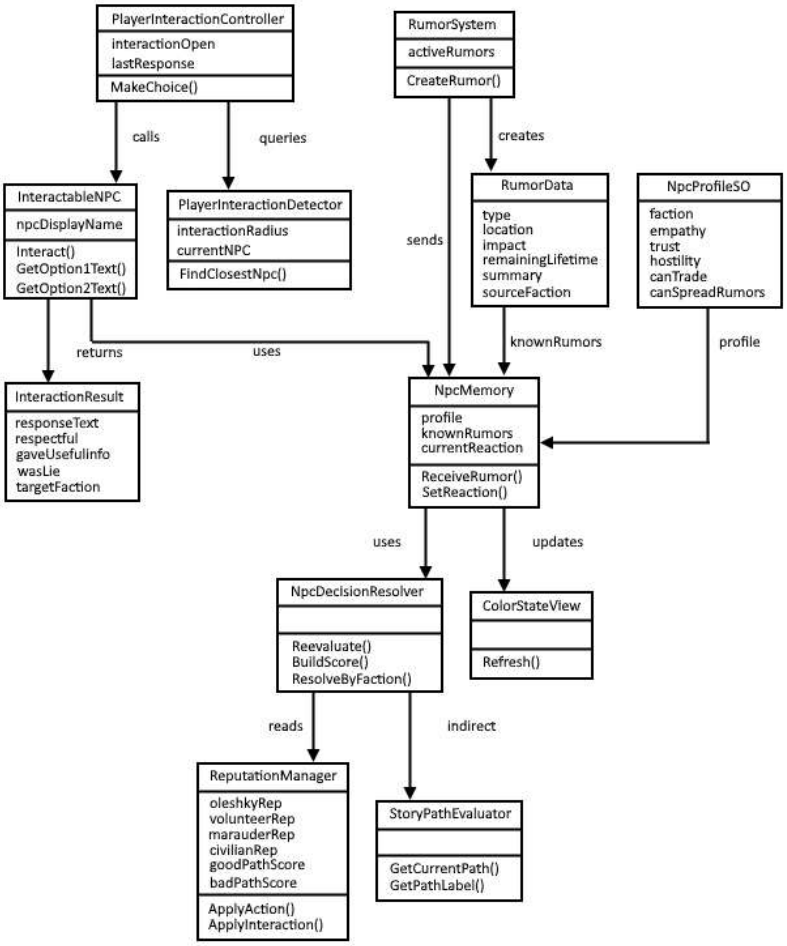



**Рисунок 2.2. Діаграми класів основних класів програмного засобу**

## 2.4. Проєктування поведінки системи

Поведінкове проєктування є ключовим для даного застосунку, так як саме воно відображає динаміку адаптивного штучного інтелекту NPC. Для опису логіки роботи системи доцільно використати поєднання Sequence Diagram, Activity Diagram та State Diagram. Таке поєднання дозволяє одночасно показати часову послідовність викликів, алгоритм обробки події та переходи між поведінковими станами.

Sequence Diagram фіксує послідовність взаємодії між гравцем, системою виявлення NPC, контролером взаємодії, самим NPC, підсистемами репутації й чуток та інтерфейсом. Це дає змогу простежити повний ланцюг обробки однієї значущої події, починаючи від натискання клавіші і завершуючи зміною реакції NPC та оновленням UI.

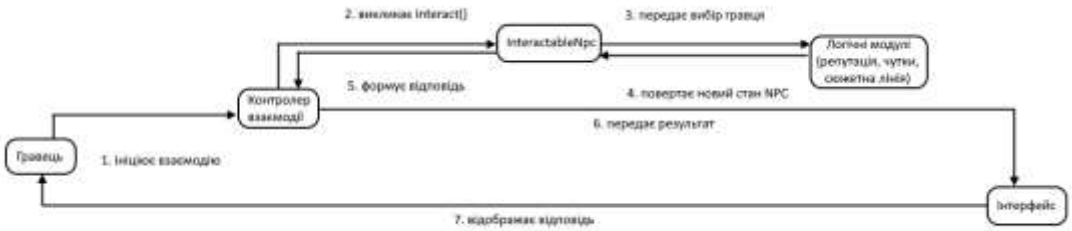



**Рисунок 2.3. Sequence Diagram сценарію взаємодії гравця з NPC**

Рисунок 2.3 ілюструє основний часовий сценарій роботи системи. Після наближення до NPC гравець ініціює взаємодію, обирає один із варіантів звернення, а система послідовно виконує зміну репутації, створення чутки, повторну оцінку реакції NPC та відображення результату в інтерфейсі.

Activity Diagram використовується для відображення загального алгоритму роботи сценарію. На відміну від Sequence Diagram, вона показує не об’єкти системи, а логіку проходження процесу: від появи NPC у радіусі взаємодії до оновлення репутації та формування відповіді.(див. рис. 2.4)


**Рисунок 2.4. Activity Diagram сценарію взаємодії з NPC**

Для безпосереднього опису поведінки NPC доцільно використати State Diagram(див. рис. 2.5). Вона показує, у яких станах може перебувати неігровий персонаж і за яких умов виконуються переходи між ними. Це є особливо корисним для демонстрації адаптивного AI, оскільки дає змогу наочно

показати, як дії гравця та соціальний контекст впливають на зміну ставлення NPC.

Базовим станом для більшості NPC є Neutral. Далі залежно від накопичених дій гравця, рівня довіри, типу фракції та отриманих чуток NPC може перейти в стани Friendly, Suspicious, RefuseHelp, Lie, Hostile або Flee. Наявність таких переходів прямо підтримує реалізацію гарної, нейтральної та поганої сюжетних ліній.

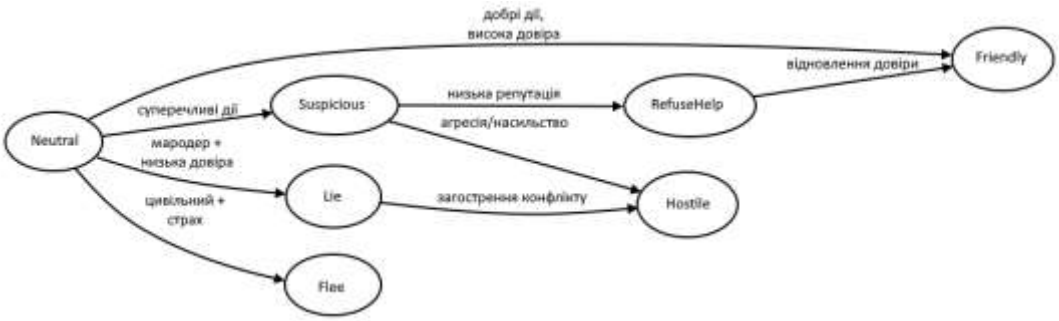



**Рисунок 2.5. State Diagram зміни станів NPC**

## 2.5. Проєктування інтерфейсу користувача

На етапі прототипування інтерфейс користувача виконує насамперед функцію демонстрації логіки системи, а не остаточного художнього оформлення. Ось чому в розроблюваному застосунку використовується спрощений інтерфейс, який дозволяє бачити поточний стан NPC, обирати варіант звернення, отримувати відповідь та відслідковувати зміну сюжетної лінії.

Інтерфейсна модель включає кілька основних елементів: підказку про доступність NPC поруч, вікно вибору дії, область відображення відповіді NPC, блок поточної сюжетної лінії та debug-панель з репутацією і списком активних чуток. Така структура інтерфейсу забезпечує достатню інформативність для тестування системи й одночасно не перевантажує користувача зайвими деталями.

Для представлення інтерфейсних рішень доцільно використати wireframe-макети. Можуть зафіксувати композицію елементів, логіку навігації та спосіб відображення результатів взаємодії до переходу до більш деталізованого графічного оформлення.

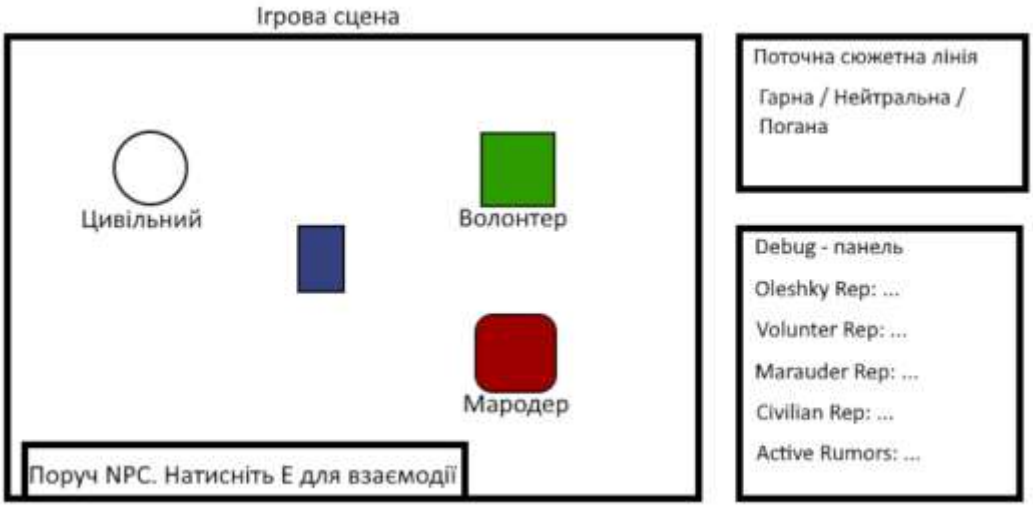



**Рисунок 2.6. Wireframe головної сцени прототипу**

На рисунку 2.6 показано загальну структуру головної сцени прототипу. Центральну область займає ігровий простір із гравцем і NPC, а допоміжні інформаційні блоки розміщено праворуч та в нижній частині екрана. Зокрема, передбачено окрему зону для відображення поточної сюжетної лінії та debug-панель для перевірки стану репутації й активних чуток.


**Рисунок 2.7. Wireframe вікна взаємодії з NPC**

Wireframe, наведений на рисунку 2.7, відображає структуру діалогу з NPC. У межах вікна показується ім’я або тип персонажа, його поточний стан, два варіанти звернення та текстова відповідь після вибору дії. Це безпосередньо підтримує перевірку адаптивної реакції NPC на поведінку гравця.

### 2.6. Висновки до розділу 2

У другому розділі було обґрунтовано архітектурні та проєктні рішення для розроблення програмного засобу у вигляді 3D ігрового прототипу з адаптивним штучним інтелектом NPC. Визначено, що для реалізації проєкту доцільно використовувати монолітну компонентно-орієнтовану архітектуру в межах Unity-застосунку.

На основі UML та wireframe-діаграм описано компонентну структуру системи, варіанти використання, класову модель, сценарії взаємодії, алгоритм обробки подій, переходи між поведінковими станами NPC та загальний вигляд інтерфейсу користувача. Представлена модель забезпечує логічну узгодженість між підсистемами репутації, чуток, сюжетної лінії та адаптивної реакції NPC.

Отримані результати створюють цілісну основу для переходу до практичної реалізації програмного засобу, опис якої доцільно навести в наступному розділі випускної кваліфікаційної роботи.

# РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1. Середовище та інструменти розробки

Основним середовищем реалізації проєкту обрано Unity 2022.3.10f1 LTS. Зумовлено це тим, що LTS-версія рушія орієнтована на стабільну розробку, містить повний набір інструментів для роботи зі сценами, фізикою, матеріалами, анімацією, інтерфейсом користувача та скриптовою логікою. Для даного кваліфікаційного проєкту це особливо важливо, оскільки прототип має поєднувати 3D сцену, переміщення гравця, взаємодію з NPC, візуалізацію станів і механізми адаптивного AI в одному середовищі.

Базовою мовою програмування є C#, який природно інтегрується з Unity та дозволяє реалізувати об’єктно-орієнтовану модель взаємодії між підсистемами. Для даного проєкту C# використано для створення скриптів керування гравцем, модулів репутації, системи чуток, оцінювання сюжетної лінії та логіки NPC. Серед переваг C# для цієї задачі слід виділити типобезпечність, підтримку об’єктно-орієнтованого підходу, зручну роботу з класами, перерахуваннями, колекціями та серіалізацією.

Для написання та редагування коду доцільно використовувати Visual Studio 2022 або Visual Studio Community, оскільки це середовище надає зручне автодоповнення, налагодження, підсвічування помилок, інтеграцію з Git та підтримку C#-проєктів [13]. Використання IDE набагато спрощує підтримку великої кількості взаємопов’язаних скриптів, що важливо для поетапного нарощування функціональності.

Для контролю версій і впорядкування змін у кодовій базі використовується Git. Застосування Git дозволяє фіксувати стан проєкту після кожного стабільного етапу, повертатися до попередніх версій, працювати з гілками розробки та відокремлювати експериментальні зміни від основної гілки [14]. Для навчального проєкту це важливо не менше, ніж для промислової розробки, так як дає змогу безпечно рефакторити логіку, тестувати різні варіанти поведінки NPC і не втрачати працездатні стани системи.

На рівні інструментів Unity у проєкті використовуються стандартні вбудовані та пакетні засоби. Для переміщення гравця по поверхні у прототипі застосовано Character Controller – компонент, орієнтований на керування персонажем без повноцінної фізичної симуляції Rigidbody [15]. Для конфігурації даних NPC використовується ScriptableObject, що дозволяє зберігати профілі фракцій, рівні емпатії, довіри, агресії та інші параметри як окремі assets проєкту [16]. Для навігації агентів і подальшого розширення поведінки NPC використовується пакет AI Navigation, який забезпечує підтримку NavMesh, агентів, перешкод та зв’язків між ділянками навігації [17].

Організація репозиторію побудована за принципом функціонального розподілу: окремо виділяються сцени, скрипти, профілі NPC, матеріали, префаби та артефакти інтерфейсу. Усередині папки Scripts логіка розділяється на підкаталоги AI, Systems, UI та Player, що полегшує орієнтацію в коді та відповідає компонентній структурі застосунку. Така організація узгоджується з архітектурними рішеннями попереднього розділу та забезпечує простоту подальшого масштабування прототипу.

Усі інструменти та їх призначення вказані у таблиці 3.1

**Таблиця 3.1. інструменти та середовища необхідні для розробки**

| Інструмент | Призначення | Примітка |
|---|---|---|
| Unity 2022.3.10f1 LTS | Основний рушій розробки | Сцени, компоненти, скрипти, матеріали |
| C# | Реалізація логіки застосунку | Скрипти керування, репутації, AI, UI |
| Visual Studio 2022 | IDE та налагодження | Редагування, дебаг, навігація по коду |

продовження табл. 3.1.

| | | |
|---|---|---|
| Git | Контроль версій | Фіксація змін, повернення до стабільних станів |
| AI Navigation | Навігація NPC | NavMesh, агенти, перешкоди |
| ScriptableObject | Конфігурація даних | Профілі NPC та параметри поведінки |

### 3.2. Реалізація ключових компонентів системи

Практична реалізація програмного засобу виконувалась відповідно до проєктних рішень, сформованих у розділі 2. Підсистеми застосунку були розділені за функціональним призначенням: керування гравцем, виявлення та взаємодія з NPC, зберігання репутації, поширення чуток, оцінювання сюжетної лінії, вибір реакції NPC та відображення станів у UI. Це дало змогу реалізовувати логіку поетапно: спочатку на простих примітивах і кольорах, а потім – розширювати її до більш завершеної сцени.

Компонент SimplePlayerMover відповідає за переміщення гравця по площині сцени. Для цього використовується CharacterController, який обробляє рух об’єкта без моделювання інерції фізичного тіла, що наведено у лістінгу 3.1. У прототипі це рішення є переважним, оскільки головним завданням було не створення складної фізичної моделі персонажа, а забезпечення керованого переміщення між NPC і тестовими зонами взаємодії.

**Лістинг 3.1. Фрагмент реалізації переміщення гравця**

```
[RequireComponent(typeof(CharacterController))]
public class SimplePlayerMover : MonoBehaviour
{
    [SerializeField] private float moveSpeed = 5f;
    [SerializeField] private float gravity = -9.81f;
    private CharacterController controller;
    private Vector3 velocity;
```

```
        private       void       Awake()       =>       controller       =
GetComponent<CharacterController>();

        private void Update()
        {
            float h = Input.GetAxisRaw("Horizontal");
            float v = Input.GetAxisRaw("Vertical");
            Vector3 move = new Vector3(h, 0f, v).normalized;

            controller.Move(move * moveSpeed * Time.deltaTime);

            if (controller.isGrounded && velocity.y < 0f)
                velocity.y = -2f;

            velocity.y += gravity * Time.deltaTime;
            controller.Move(velocity * Time.deltaTime);
        }
    }
```

Взаємодія з NPC реалізована через зв’язку PlayerInteractionDetector, PlayerInteractionController, InteractableNpc і InteractionUI. Перший компонент визначає найближчого неігрового персонажа в заданому радіусі, другий керує відкриттям та завершенням взаємодії, третій формує варіанти звернення і відповіді залежно від фракції та стану NPC, а четвертий відповідає за виведення підказок і текстових блоків у прототипному інтерфейсі.

Система репутації реалізована як окремий менеджер, що акумулює значення локальної репутації в Олешках, ставлення волонтерів, мародерів і цивільних, а також напрям сюжетного дрейфу через окремі значення goodPathScore та badPathScore, що наведено в лістінгу 3.2. Такий підхід дозволив позбутися ситуацій, коли одна добра дія миттєво компенсувала серію поганих вчинків.

**Лістинг 3.2. Фрагмент логіки оновлення репутації після взаємодії**

```
public void ApplyInteraction(FactionType targetFaction, bool respectful)
{
    if (respectful)
    {
        switch (targetFaction)
        {
            case FactionType.Volunteer:
                oleshkyRep += 5f;
                volunteerRep += 10f;
                goodPathScore += 0.75f;
                break;
            case FactionType.Civilian:
                oleshkyRep += 5f;
                civilianRep += 10f;
                goodPathScore += 0.75f;
```

```
                    break;
                }
            }
            else
            {
                switch (targetFaction)
                {
                    case FactionType.Volunteer:
                        oleshkyRep -= 12f;
                        volunteerRep -= 15f;
                        badPathScore += 1.5f;
                        break;
                    case FactionType.Civilian:
                        oleshkyRep -= 10f;
                        civilianRep -= 15f;
                        badPathScore += 1.25f;
                        break;
                }
            }
            ClampAll();
        }
```

Підсистема чуток реалізує передачу інформації про значущі дії гравця між NPC. Після події створюється об’єкт RumorData, який містить тип чутки, її короткий опис, силу впливу та час життя. Далі RumorSystem передає копію цієї чутки всім зареєстрованим NPC, а їхній локальний стан оновлюється через NpcMemory.

Ключовим компонентом адаптивного AI виступає NpcDecisionResolver. Він не просто зчитує одне значення репутації, а формує сумарну оцінку на основі кількох факторів: локальної репутації в Олешках, фракційної репутації, особистого профілю NPC, наявних чуток та поточної сюжетної лінії, що відображено у лістінгу 3.3. На основі цього розрахунку NPC переходить у стани Friendly, Neutral, Suspicious, RefuseHelp, Lie, Flee або Hostile.

**Лістинг 3.3. Спрощений фрагмент оцінювання стану NPC**

```
        private float BuildScore()
        {
            NpcProfileSO profile = memory.Profile;
            float score = 0f;

            score                                                    +=
ReputationManager.Instance.GetLocationRep(LocationId.Oleshky);
            score += ReputationManager.Instance.GetFactionRep(profile.faction) *
0.6f;
            score += (profile.trust - 50f) * 0.4f;
            score -= (profile.hostility - 50f) * 0.3f;

            if  (memory.HasRumor(RumorType.HelpedAnimal)  &&  profile.empathy  >=
60f)
```

```
            score += 15f;

        if (memory.HasRumor(RumorType.RobbedCivilian))
            score -= 25f;

        return score;
    }
```

Для того, щоб зробити реакції NPC наочними без великих витрат на графічні ресурси, у прототипі застосовано спеціальний компонент ColorStateView. Його призначення полягає у зміні кольору простих фігур відповідно до стану NPC. Наприклад, зелений колір відображає дружню реакцію, жовтий – підозру, а червоний – ворожість. Саме така візуалізація дозволила швидко перевірити коректність логіки адаптивного AI на ранньому етапі, це не витрачає час на анімації та повноцінні моделі персонажів.

Таким чином, реалізовані модулі у таблиці 3.2 відповідають архітектурним рішенням, описаним у попередньому розділі. Підсистема керування гравцем відповідає модулю Player Controller, менеджер репутації – компоненту обробки соціального стану, система чуток – підсистемі поширення інформації, а зв’язка NpcMemory, NpcDecisionResolver і ColorStateView – підсистемі AI NPC.

**Таблиця 3.2. Проєктні модулі**

| Проєктний модуль | Реалізація у коді | Призначення |
|---|---|---|
| Player Controller | SimplePlayerMover, CharacterController | Переміщення персонажа сценою |
| NPC Interaction | PlayerInteractionDetector, PlayerInteractionController, InteractableNpc | Пошук NPC поруч і запуск діалогу |

Продовження табл. 3.2.

| | | |
|---|---|---|
| Reputation Module | ReputationManager | Локальна та фракційна репутація, moral drift |

| | | |
|---|---|---|
| Rumor Module | RumorSystem, RumorData, NpcRegistry | Створення та поширення чуток |
| NPC AI | NpcMemory, NpcDecisionResolver, ColorStateView | Вибір і візуалізація реакції NPC |
| Story Path | StoryPathEvaluator | Визначення гарної, нейтральної або поганої лінії |
| UI Module | InteractionUI, DebugOverlay | Відображення вибору, відповіді та службової інформації |

### 3.3. Реалізація збереження стану, конфігурації та налагодження системи

Оскільки в роботі розроблюваний програмний засіб є локальним ігровим прототипом, у його поточній версії не використовується реляційна база даних, ORM-модель або зовнішнє сховище даних. Натомість для зберігання конфігураційних параметрів застосовано ScriptableObject assets, а для поточного стану системи – поля менеджерів, серіалізовані змінні та об’єкти runtime-логіки. Можна вважати такий підхід є виправданим, оскільки дозволяє зосередитись на демонстрації адаптивної поведінки NPC, не ускладнюючи прототип зайвою інфраструктурою.

Профілі NPC зберігаються у вигляді окремих ScriptableObject assets. У них розміщуються фракція, рівень емпатії, довіри, агресії, можливість торгівлі та поширення чуток, що наведено у лістінгу 3.4. Перевага цього рішення полягає в тому, що дані відокремлюються від сценових об’єктів, перевикористовуються декількома NPC і можуть налаштовуватись без зміни вихідного коду.

Поточний стан системи зберігається в менеджерах таких як ReputationManager і RumorSystem. У межах виконання сцени ці об’єкти працюють як централізовані сховища даних, через які проходять усі події, пов’язані з репутацією, чутками та сюжетним дрейфом. Цього достатньо для односценового прототипу й водночас добре масштабується до майбутнього JSON-збереження або сейв-системи.

Для налагодження системи реалізовано окремий сервісний інтерфейс DebugOverlay, який виводить на екран поточну сюжетну лінію, числові значення репутації, активні чутки та службові гарячі клавіші. Це дозволяє швидко виявляти помилки балансу, конфлікти вводу та некоректні переходи між станами NPC. Також у процесі розробки активно використовуються журнали повідомлень Unity Console, серіалізовані поля в Inspector і кольорова візуалізація станів NPC.

**Лістинг 3.4. Фрагмент опису конфігурації NPC через ScriptableObject**

```
[CreateAssetMenu(menuName = "AI/NPC Profile", fileName = "NPC_Profile")]
public class NpcProfileSO : ScriptableObject
{
    public FactionType faction;
    [Range(0, 100)] public float empathy = 50f;
    [Range(0, 100)] public float trust = 50f;
    [Range(0, 100)] public float hostility = 50f;
    public bool canTrade = false;
    public bool canSpreadRumors = true;
}
```

Додатковим елементом організації стану є структура репозиторію, де вихідний код, профілі, префаби, матеріали й сцени розділено за функціональними каталогами. Що дає можливість підтримувати проєкт у зрозумілому стані та зменшує ризик втрати або дублювання даних.

### 3.4. Інтеграція підсистем та внутрішні інтерфейси взаємодії

На відміну від типових прикладних систем, у яких окремо реалізується REST або GraphQL API, у даному проєкті взаємодія відбувається усередині одного Unity-застосунку. Тому цей підрозділ присвячено не зовнішньому API, а внутрішнім інтерфейсам взаємодії між модулями. Фактично йдеться про внутрішній прикладний контракт між компонентами системи: які скрипти викликають один одного, які дані передаються між ними та в якій послідовності формується реакція середовища на дії гравця.

Ключовим інтеграційним сценарієм є взаємодія гравця з NPC наведеному у лістінгу 3.5. Ланцюг викликів починається з PlayerInteractionDetector, який фіксує, що поруч із гравцем знаходиться доступний InteractableNpc. Далі PlayerInteractionController відкриває взаємодію та передає інформацію до InteractionUI. Після вибору варіанту звернення запускається метод Interact, який у свою чергу оновлює репутацію героя, створює чутку про подію, передає її до інших NPC, викликає переоцінювання станів і повертає текстову відповідь.

**Лістинг 3.5. Фрагмент внутрішнього інтерфейсу взаємодії з NPC**

```
public InteractionResult Interact(bool respectful)
{
    InteractionResult result = new InteractionResult
    {
        respectful = respectful,
        targetFaction = memory.Profile.faction
    };


PlayerActionTracker.Instance.RegisterInteraction(memory.Profile.faction,
respectful);

        NpcDecisionResolver resolver = GetComponent<NpcDecisionResolver>();
        if (resolver != null)
            resolver.Reevaluate();

        NpcReactionType reactionAfter = memory.CurrentReaction;
        result.responseText   =   BuildResponse(reactionAfter,   respectful,
result);

        return result;
    }
```

На рівні інтеграції даних важливу роль відіграє RumorSystem, який після виклику CreateRumor проходить по всіх зареєстрованих NPC через NpcRegistry і передає кожному копію об’єкта RumorData. Після цього NpcMemory зберігає чутку, а NpcDecisionResolver виконує повторну оцінку стану. Таким чином, внутрішня взаємодія модулів побудована не навколо зовнішніх HTTP-запитів, а навколо подій, викликів методів і передачі об’єктів у пам’яті в межах одного процесу.

Суттєвою перевагою такої інтеграції є низька затримка, відсутність залежності від мережі та повна керованість усіх переходів у коді. З практичної точки зору це означає, що розробник може швидко відслідковувати причинно-наслідковий ланцюг: дія гравця – зміна репутації – створення чутки – переоцінка NPC – оновлення кольору – зміна сюжетної лінії.

Разом із тим така інтеграція вимагає акуратного розподілу відповідальності. Саме тому в ході рефакторингу були розділені функції зберігання репутації, оцінювання сюжетної лінії, UI-відображення та контролю взаємодії. Це дозволило уникнути надмірної зв’язаності коду та створити стабільнішу основу для майбутнього розширення проєкту, зокрема додавання руху NPC за NavMesh, повноцінної місійної логіки та збереження стану між сценами.

Типовий сценарій внутрішньої взаємодії модулів можна подати у вигляді такої послідовності:

1) гравець наближається до NPC;

2) детектор взаємодії фіксує доступного NPC;

3) інтерфейс відкриває меню вибору дії;

4) після вибору варіанту звернення оновлюється репутація;

5) формується чутка про подію;

6) усі NPC отримують оновлений соціальний контекст;

7) їхній стан перераховується;

8) змінюється візуальне представлення реакції;

9) система переоцінює поточну сюжетну лінію;

10) гравець отримує відповідь NPC.

### 3.5. Висновок до розділу 3

У розділі було розглянуто практичну реалізацію програмного засобу у вигляді 3D прототипу в Unity. Показано, що для розроблення системи адаптивного AI NPC було використано компонентну організацію коду, де окремі модулі відповідають за керування гравцем, репутацію, поширення чуток, вибір реакцій NPC, визначення сюжетної лінії та виведення результатів у користувацький інтерфейс.

Також обґрунтовано коригування структури розділу порівняно зі стандартним шаблоном: замість реалізації бази даних та прикладного веб-API описано підсистеми збереження конфігурацій, внутрішнє керування станом і інтеграцію локальних модулів в межах одного Unity-застосунку. Це відповідає специфіці проєкту, який є локальною ігровою системою, орієнтованою на демонстрацію адаптивної поведінки NPC, а не на роботу з зовнішнім сервером або реляційним сховищем.

Отримана реалізація підтверджує відповідність архітектурним рішенням, сформованим у розділі 2, і створює основу для подальшого розширення прототипу до повноцінної місійної сцени з рухом NPC, додатковими сценаріями взаємодії та поглибленою сюжетною логікою.

# РОЗДІЛ 4. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 4.1. Підхід до модульного тестування

Модульне тестування в розробленому проєкті орієнтоване на перевірку окремих скриптів і логічних блоків, що не потребують повного запуску всієї ігрової сцени. Такий підхід є доцільним для компонентів, які виконують локальні обчислення та визначають зміни стану системи за вхідними параметрами. До таких компонентів належать менеджер репутації, оцінювач сюжетної лінії, логіка обробки чуток, а також окремі допоміжні структури, що описують результати взаємодії.

Оскільки прототип реалізовано в Unity, для модульного тестування доцільно використовувати Unity Test Framework. У межах цього підходу Edit Mode тести застосовуються для перевірки чистої логіки класів і методів, тоді як Play Mode тести доцільні для тих сценаріїв, де потрібно перевірити поведінку об’єктів у контексті сцени, корутин або життєвого циклу компонентів [18]. Для перевірки очікуваних результатів використовуються стандартні твердження NUnit, зокрема перевірки на рівність, істинність умов та відсутність null-посилань [19].

На рівні модульного тестування насамперед перевіряються методи, що змінюють репутацію гравця, визначають поточну сюжетну лінію, зберігають та обробляють чутки, а також розраховують реакції NPC на підставі репутаційних параметрів. Типовим прикладом перевірки є встановлення того, чи правильно після агресивної взаємодії з волонтером знижується локальна репутація та чи зміщується стан системи в бік поганої сюжетної лінії.

Окремо необхідно було перевірити ті місця, де під час налагодження виникали фактичні дефекти. Деякі з них викладені у вигляді таблиці 4.1. До них належали конфлікт гарячих клавіш між діалоговим вибором і debug-діями, некоректне повернення до гарної лінії після серії негативних рішень, дублювання окремих компонентів на NPC-об’єктах та відсутність призначених

профілів NPC. Ось чому модульне тестування було зорієнтоване не лише на перевірку позитивних сценаріїв, а й на відтворення вже виявлених проблемних випадків.

**Таблиця 4.1. Тести на виявлення фактичних дефектів**

| Код тесту | Об’єкт перевірки | Очікуваний результат | Фактичний результат |
|---|---|---|---|
| MT-01 | ApplyAction(HelpedAnimal) | До репутації додаються позитивні значення, good-path зростає | Очікування підтверджено |
| MT-02 | ApplyAction(AttackedVolunteer) | VolunteerRep та OleshkyRep зменшуються, bad-path зростає | Очікування підтверджено |
| MT-03 | ApplyInteraction(Volunteer, false) | Погіршується ставлення волонтерів і цивільних | Очікування підтверджено |
| MT-04 | StoryPathEvaluator після серії негативних дій | Поточний шлях визначається як поганий | Очікування підтверджено після виправлення логіки |
| MT-05 | RumorData.Clone() | Створюється копія чутки без втрати полів | Очікування підтверджено |
| MT-06 | NpcMemory.ReceiveRumor() | Чутка додається до пам’яті NPC і запускається переоцінка стану | Очікування підтверджено |

### 4.2. Інтеграційне та UI-тестування

Інтеграційне тестування спрямоване на перевірку взаємодії між окремими модулями системи. Для розробленого прототипу найбільш важливим є ланцюг «гравець - взаємодія з NPC - зміна репутації - створення чутки - переоцінка стану NPC - зміна кольору і текстової відповіді». Виключно цей сценарій демонструє ключову особливість проєкту – адаптивну реакцію неігрових персонажів на дії користувача.

На відміну від модульного тестування, інтеграційна перевірка потребує активної сцени. Тут перевіряються взаємодія скриптів між собою, коректне зчитування даних з профілів NPC, робота детектора наближення, відкриття вікна взаємодії, передача вибраного варіанта до контролера взаємодії та подальше оновлення реакції NPC.

Для UI-тестування в межах прототипу основною метою є не художня оцінка інтерфейсу, а перевірка його функціональної коректності. Було важливо впевнитися, що підказка про NPC поруч з’являється вчасно, вікно вибору дії відкривається за натисканням клавіші взаємодії, а відповідь NPC відображається після натискання одного з доступних варіантів. Додатково перевіряється індикатор поточної сюжетної лінії, а також debug-панель, яка допомагає відстежувати зміну репутації та активних чуток, результати яких відображені й таблиці 4.2.

Важливо продемонструвати реального стану програми під час перевірки основних сценаріїв. Наведено скріншот стартового тестового середовища (див. рис. 4.1), після пояснення взаємодії з NPC – на скріншот відкритого вікна з двома варіантами вибору (див. рис. 4.2), а після аналізу зміни реакції – на скріншот різнокольорових NPC станів та оновленого індикатора сюжетної лінії (див. рис. 4.3).

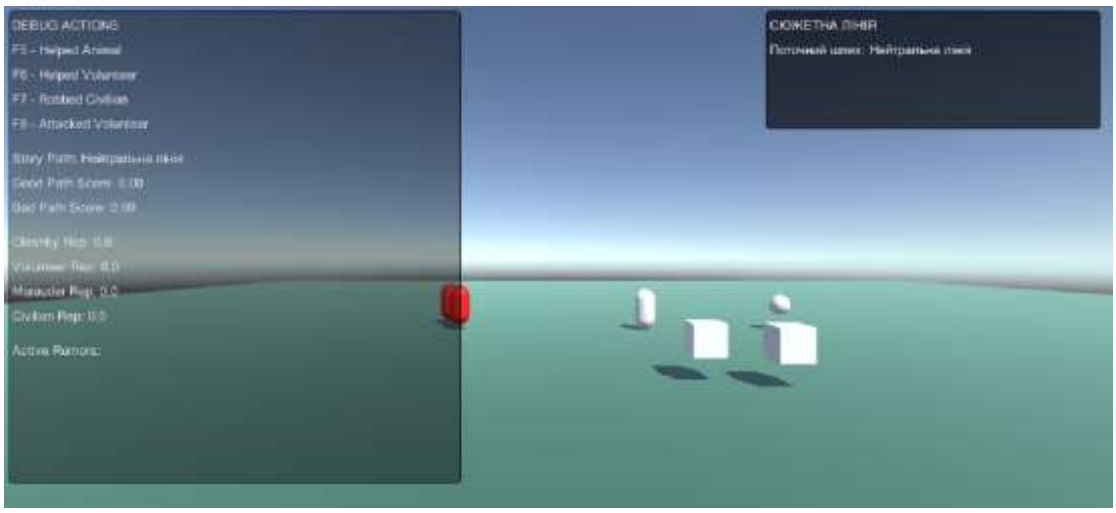



**Рисунок 4.1. Головна тестова сцена програми з Player, NPC різних типів та debug-панеллю**

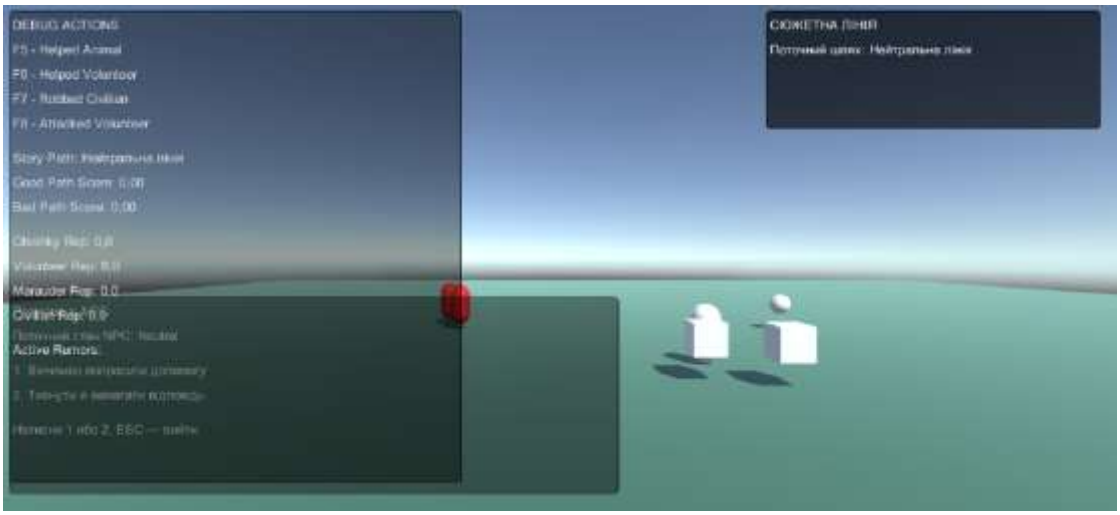



**Рисунок 4.2. Вікно взаємодії з NPC із двома варіантами звернення та текстовою відповіддю**

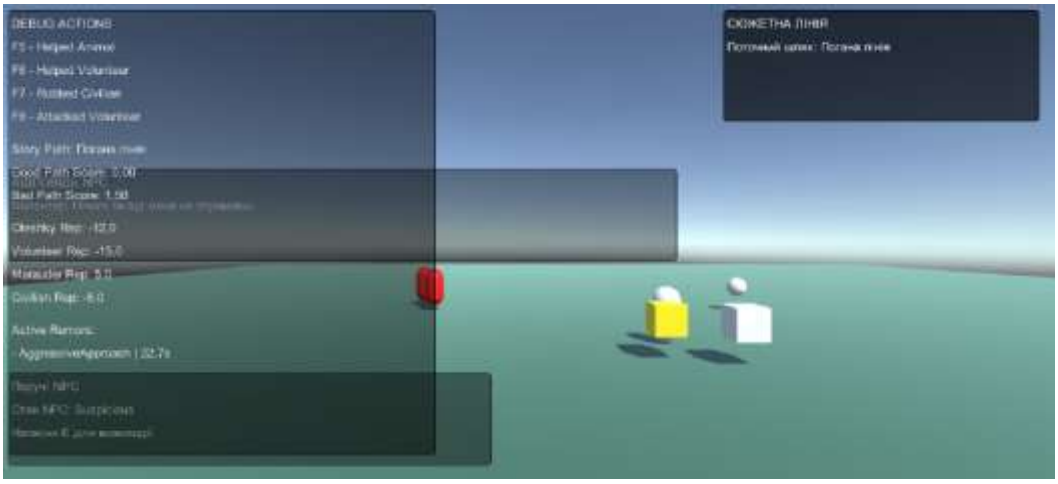



**Рисунок 4.3. Приклад зміни кольору NPC і показника сюжетної лінії після негативної взаємодії**

**Таблиця 4.2. Тестування взаємодій в різних сценаріях**

| Код тесту | Сценарій інтеграції | Очікуваний результат | Фактичний результат |
|---|---|---|---|
| IT-01 | Підійти до волонтера на позитивній репутації | З'являється підказка про взаємодію, відкривається діалог | Працює коректно |
| IT-02 | Обрати ввічливий варіант взаємодії з волонтером | Гравець отримує корисну відповідь, репутація зростає | Працює коректно |
| IT-03 | Обрати агресивний варіант взаємодії з волонтером | Репутація знижується, NPC реагує негативно | Працює коректно після усунення конфлікту клавіш |
| IT-04 | Взаємодія з мародером на поганій лінії | NPC не обов'язково стає дружнім, але може брехати або толерувати агресію | Працює коректно |
| IT-05 | Зміна локальної репутації після серії поганих дій | Індикатор переходить у погану лінію | Працює коректно після корекції ваг path-score |
| IT-06 | Відкрити UI та натиснути 1/2 у діалозі | Діалогові клавіші не запускають debug-дії | Працює коректно після перенесення debug-клавіш на F5–F8 |

### 4.3. Аналіз результатів тестування

За результатами виконаних перевірок можна зробити висновок, що загальна логіка прототипу працює стабільно, а ключові проектні рішення, сформульовані в попередньому розділі, отримали практичне підтвердження. Найважливішим результатом тестування є те, що система адаптивного AI

дійсно змінює поведінку NPC залежно від дій гравця, а не працює як фіксований набір наперед визначених реплік.

Під час тестування(див. табл 4.3) були виявлені кілька суттєвих дефектів. Перший із них полягав у конфлікті між debug-клавішами та клавішами вибору реплік у діалозі, через що негативна взаємодія могла випадково зараховуватися як позитивна дія. Другий дефект полягав у недостатньо стабільній логіці визначення поганої сюжетної лінії: навіть після серії негативних рішень система інколи поверталася до гарної лінії через надмірний вплив окремих позитивних параметрів. Третя проблема стосувалася конфігурації об’єктів сцени – деякі NPC не мали призначених профілів або містили дубльовані компоненти, що ускладнювало інтерпретацію результатів.

Після виправлення зазначених дефектів було отримано більш прогнозовану модель роботи системи. Перенесення debug-дій на окремі клавіші усунуло побічні спрацювання під час діалогу, введення окремих накопичувальних good-path та bad-path показників зробило сюжетну лінію стійкішою до випадкових коливань, а перегляд налаштувань компонентів на сцені дозволив забезпечити коректну роботу профілів NPC.

**Таблиця 4.3. дефекти та їх усунення**

| Код дефекту | Опис | Прояв | Спосіб усунення |
|---|---|---|---|
| D-01 | Конфлікт клавіш 1/2 між діалогом і debug-діями | Поганий вибір інколи зараховувався як добра дія | Debug-клавіші перенесено на F5–F8, додано блокування під час відкритого діалогу |

Продовження табл 4.3

| | | | |
|---|---|---|---|
| D-02 | Нестабільне визначення поганої сюжетної лінії | Після серії негативних дій система могла повернутись до гарної лінії | Введено окремі good-path / bad-path накопичувачі та оновлено оцінювач сюжетної лінії |
| D-03 | Порожнє поле Profile у частини NPC | NPC працював некоректно або не змінював стан | Створено та призначено ScriptableObject-профілі |
| D-04 | Дублювання NpcDecisionResolver на об’єкті | Логіка переоцінки стану працювала непередбачувано | Зайві компоненти видалено, сцену повторно перевірено |

Підсумовуючи результати перевірки, можна стверджувати, що розроблений прототип відповідає основним функціональним вимогам: забезпечує переміщення гравця по поверхні, підтримує виявлення NPC поблизу, дозволяє обирати альтернативні варіанти звернення, змінює репутацію та сюжетну лінію залежно від вчинків користувача, а також коректно відображає зміну стану NPC через колір і текстову відповідь. Таким чином, тестування підтвердило придатність реалізованого рішення для демонстрації основної ідеї дипломного проєкту.

### 4.4. Інструкція користувача

Розроблений застосунок є локальним Unity-прототипом, орієнтованим на демонстрацію логіки адаптивного штучного інтелекту NPC. Для запуску користувачеві не потрібні спеціальні знання у сфері програмування або адміністрування систем, однак необхідно мати персональний комп’ютер із встановленою операційною системою Windows та можливістю запуску застосунків, зібраних у Unity.

Запустити виконуваний файл зібраного застосунку або відкрити сцену проєкту в Unity.

Після завантаження сцени використовувати клавіші W, A, S, D для переміщення персонажа по площині.

Підійти до одного з NPC доти, доки на екрані не з’явиться підказка про доступну взаємодію (див. рис. 4.1).

Натиснути клавішу E для відкриття вікна взаємодії з NPC (див. рис. 4.2).

Вибрати один із двох варіантів звернення за допомогою клавіш 1 або 2.

Ознайомитися з відповіддю NPC та звернути увагу на зміну кольору об’єкта й показника поточної сюжетної лінії (див. рис. 4.3.-4.4.).

За потреби в режимі налагодження скористатися клавішами F5–F8 для моделювання окремих дій, що змінюють репутацію та чутки.

Для завершення роботи вийти із застосунку стандартним способом через системне вікно.

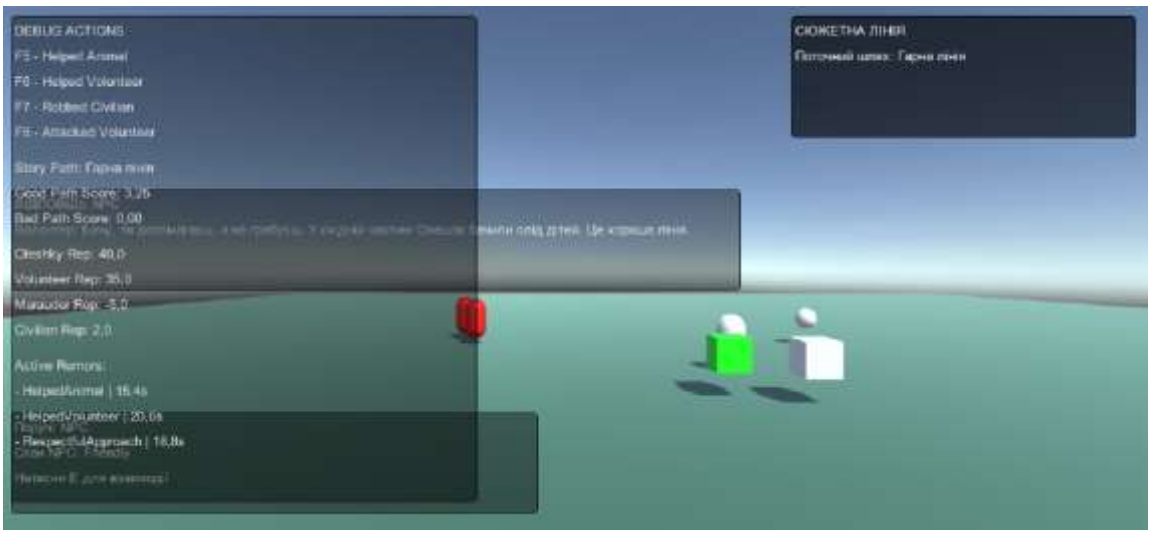



**Рисунок 4.4. Екран роботи застосунку після запуску з відображенням індикатора сюжетної лінії та відповіді NPC**

### 4.5. Локальне розгортання та запуск прототипу

Оскільки в межах кваліфікаційного проєкту розроблено локальний Unity-застосунок без серверної частини, процедура впровадження зводиться до локального розгортання та формування виконуваної збірки. На етапі розробки це виконується безпосередньо в Unity через меню налаштувань збірки [3].

Типовий порядок локального розгортання включає такі кроки: відкриття проєкту у відповідній версії Unity, перевірку підключених сцен, вибір цільової платформи, формування папки збірки та тестовий запуск зібраного файлу. Для випускної кваліфікаційної роботи достатньо локального розгортання на цільовому комп’ютері, оскільки прототип не використовує мережевих сервісів, віддалених баз даних або зовнішніх API.

Відкрити проєкт у Unity 2022.3.10f1 або сумісній версії 2022.3 LTS.

Переконатися, що тестова сцена додана до списку Scenes In Build.

У меню File → Build Settings вибрати цільову платформу Windows.

Натиснути Build та вказати каталог для збереження зібраного застосунку.

Після завершення збірки запустити виконуваний файл і виконати базову перевірку працездатності.

За необхідності повторити цикл правок, нової збірки та повторного тестування. Також можна запустити exe-файл у каталозі застосунку (див. рис. 4.5).

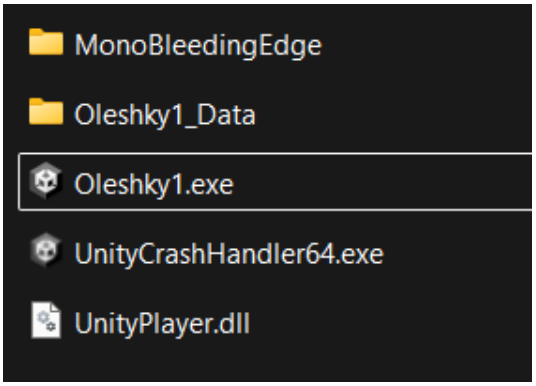



**Рисунок 4.5. Каталог зі зібраним локальним застосунком Unity**

### 4.6. Висновок до розділу 4

У розділі було розглянуто підхід до тестування та впровадження розробленого програмного засобу. З урахуванням того, що реалізований продукт є локальним 3D-прототипом у середовищі Unity, структура тестування була адаптована без акценту на перевірку баз даних, REST API чи серверних сервісів. Основну увагу приділено модульній перевірці логічних компонентів, інтеграційній взаємодії між підсистемами, коректності інтерфейсної поведінки, а також перевірці відповідності фактичної реалізації проектним рішенням попередніх розділів.

Результати тестування показали, що після усунення виявлених дефектів прототип стабільно реалізує основний функціонал: дозволяє переміщуватися сценою, взаємодіяти з NPC, змінювати репутацію, формувати чутки та відображати різні стани неігрових персонажів залежно від попередніх дій гравця. Запропонована інструкція користувача та опис локального розгортання дають змогу відтворити роботу прототипу та використати його для подальшої апробації в межах дипломного проєкту.

# ВИСНОВКИ

У результаті виконання випускної кваліфікаційної роботи було розроблено прототип програмного засобу у вигляді 3D міні-гри з елементами адаптивного штучного інтелекту для неігрових персонажів. Основна увага в роботі була приділена реалізації такої системи взаємодії, за якої NPC змінюють свою поведінку залежно від дій гравця, накопиченої репутації, поширення чуток та поточної сюжетної лінії. Практична цінність розробки полягає в тому, що створений програмний прототип демонструє можливість поєднання сюжетно-орієнтованого підходу з адаптивною AI-логікою в межах ігрового застосунку, створеного в середовищі Unity.

У першому розділі було проаналізовано предметну галузь, розглянуто особливості ігрових застосунків з адаптивною поведінкою NPC, обґрунтовано актуальність теми, вибір Unity та C#, а також сформульовано постановку задачі й основні вимоги до програмного засобу.

У другому розділі було виконано проєктування системи: обрано архітектуру програмного засобу, визначено основні модулі, описано структуру даних, взаємодію компонентів і поведінку системи за допомогою UML-діаграм. Також було розглянуто проєктування інтерфейсу користувача для демонстрації основних можливостей прототипу.

У третьому розділі описано процес розробки програмного забезпечення, використані інструменти та реалізацію ключових модулів системи. Особливу увагу було приділено реалізації механізмів переміщення гравця, взаємодії з NPC, системи репутації, поширення чуток, визначення сюжетної лінії та зміни реакцій NPC.

У четвертому розділі було розглянуто тестування та впровадження програмного засобу. Наведено основні тестові сценарії, проаналізовано результати перевірки роботи системи, виявлені недоліки та способи їх усунення. Також було подано інструкцію користувача та описано особливості запуску й використання прототипу.

У підсумку поставлену мету випускної кваліфікаційної роботи можна вважати досягнутою. Розроблений програмний засіб реалізує базову модель адаптивного штучного інтелекту для NPC, яка враховує дії гравця, локальну репутацію, чутки та належність NPC до певної фракції. У результаті неігрові персонажі змінюють своє ставлення до героя, а сам перебіг взаємодії демонструє перехід між гарною, нейтральною та поганою сюжетними лініями. Отриманий прототип може слугувати основою для подальшого розвитку гри, розширення кількості місій, ускладнення поведінки NPC, переходу від тестових об’єктів до повноцінних моделей та поглиблення сюжетної складової.

Перспективами подальшого розвитку роботи є вдосконалення поведінкової моделі NPC, додавання переміщення агентів залежно від стану, розширення системи діалогів, інтеграція повноцінної місійної логіки для ключових локацій сценарію, а також покращення користувацького інтерфейсу й візуального оформлення гри.

Посилання на повну версію коду у GitHub: URL: https://github.com/GuYdukevich/Oleshky_scene_proj

# ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

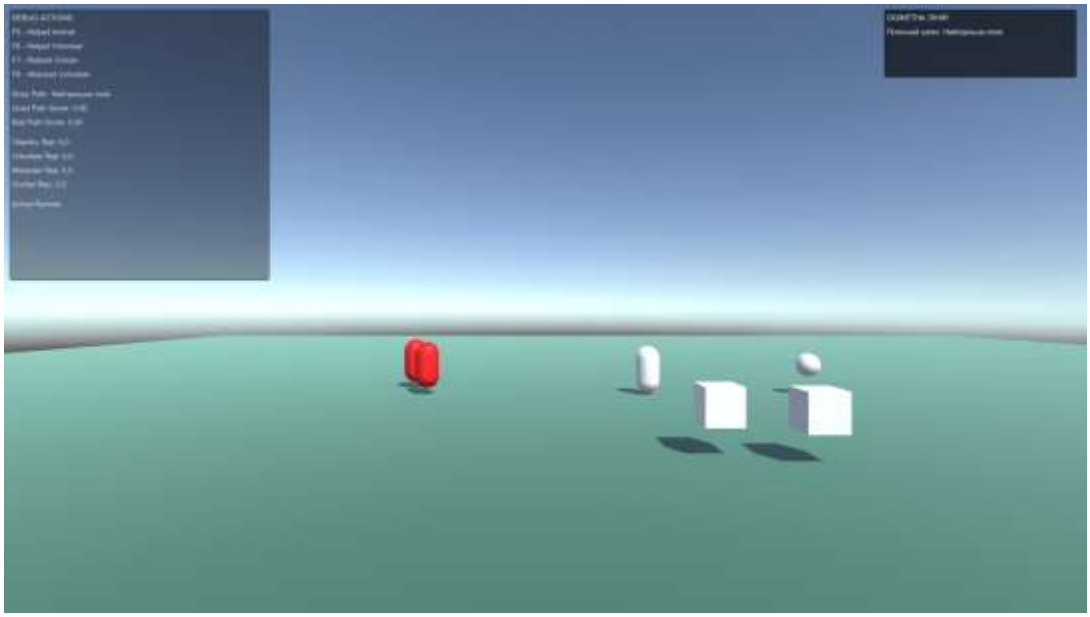

1. Авторське свідоцтво. Сценарій гри «Човняр: темні води Херсона» URL: https://iprop-ua.com/cr/sbetzcx0/ (дата звернення: 11.02.2026).
2. Finite State Machine in Game Development. URL: https://www.researchgate.net/publication/355518086_Finite_State_Machine_in_Game_Development(дата звернення: 12.02.2026).
3. A Survey of Behavior Trees in Robotics and AI. URL: https://arxiv.org/abs/2005.05842 (дата звернення: 12.02.2026).
4. Three States and a Plan: The A.I. of F.E.A.R.. URL: https://www.gamedevs.org/uploads/three-states-plan-ai-of-fear.pdf (дата звернення: 13.02.2026).
5. Improving AI Decision Modeling Through Utility Theory. URL: https://media.gdcvault.com/gdc10/slides/MarkDill_ImprovingAIUtilityTheory.pdf (дата звернення: 13.02.2026).
6. ML-Agents Overview. URL: https://docs.unity3d.com/Packages/com.unity.ml-agents%403.0/manual/index.html (дата звернення: 14.02.2026).
7. https://docs.unity3d.com/2022.3/Documentation/Manual/UnityManual.html (дата звернення: 03.06.2026).
8. Unity User Manual 2022.3 (LTS). URL: https://docs.unity3d.com/ScriptReference/index.html (дата звернення: 14.02.2026).
9. A guide on using the new AI Navigation package in Unity 2022 LTS and above. URL: https://discussions.unity.com/t/a-guide-on-using-the-new-ai-navigation-package-in-unity-2022-lts-and-above/371872 (дата звернення: 14.02.2026).
10. Changelog AI Navigation. URL: https://docs.unity3d.com/Packages/com.unity.ai.navigation%401.1/changelog/CHANGELOG.html (дата звернення: 14.02.2026).
11. About C#. URL: https://dotnet.microsoft.com/en-us/languages/csharp (дата звернення: 16.02.2026).

12. C# documentation. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 16.02.2026).
13. Visual Studio: IDE and Code Editor for Software Development. URL: https://visualstudio.microsoft.com/ (дата звернення: 12.03.2026).
14. Pro Git book. URL: https://git-scm.com/book/en/v2 (дата звернення: 13.03.2026).
15. Character Controller component reference. URL: https://docs.unity3d.com/2022.3/Documentation/Manual/class-CharacterController.html (дата звернення: 14.03.2026).
16. ScriptableObject. URL: https://docs.unity3d.com/Manual/class-ScriptableObject.html (дата звернення: 14.03.2026).
17. AI Navigation 1.1.7. URL: https://docs.unity3d.com/Packages/com.unity.ai.navigation@1.1/manual/index.html (дата звернення: 15.03.2026).
18. Unity Technologies. Test Framework. URL: https://docs.unity3d.com/2022.3/Documentation/Manual/com.unity.test-framework.html (дата звернення: 11.04.2026).
19. NUnit Documentation. Assertions. URL: https://docs.nunit.org/articles/nunit/writing-tests/assertions/assertions.html (дата звернення: 11.04.2026).
20. Unity Technologies. Build Settings. URL: https://docs.unity3d.com/2022.3/Documentation/Manual/BuildSettings.html (дата звернення: 12.04.2026).

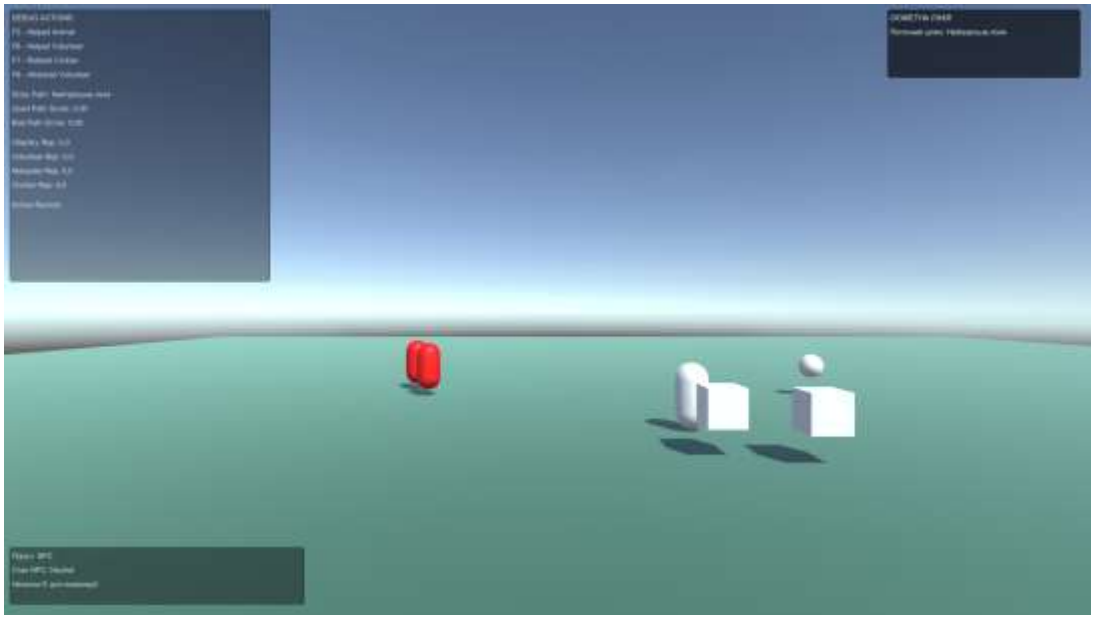
# ДОДАТОК А

Скріншоти з сцени гри

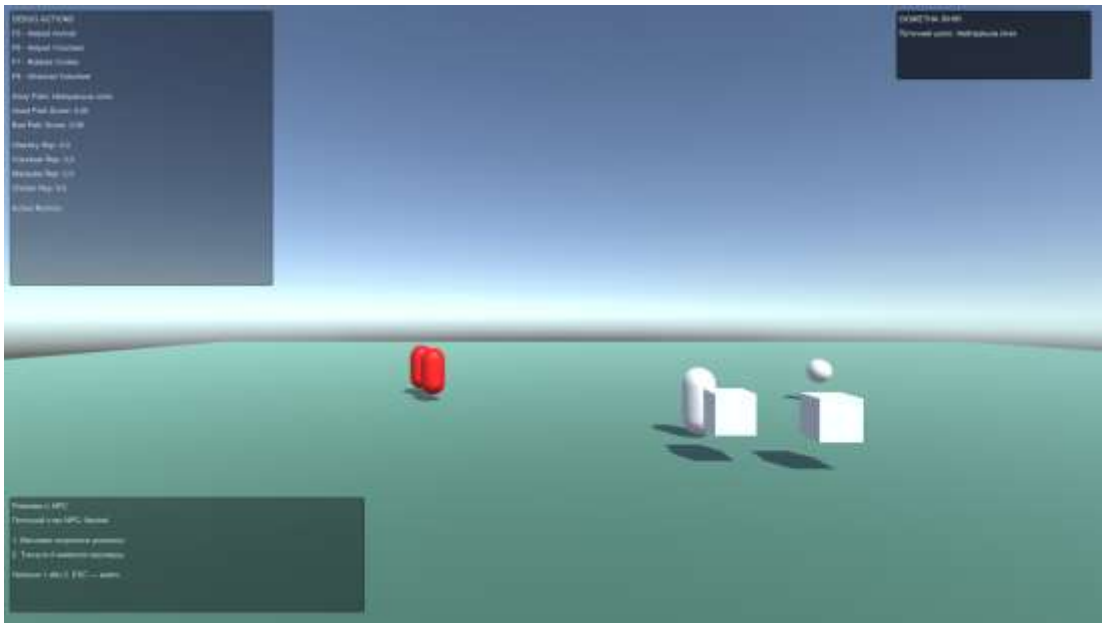
## А.1. Стартова сцена



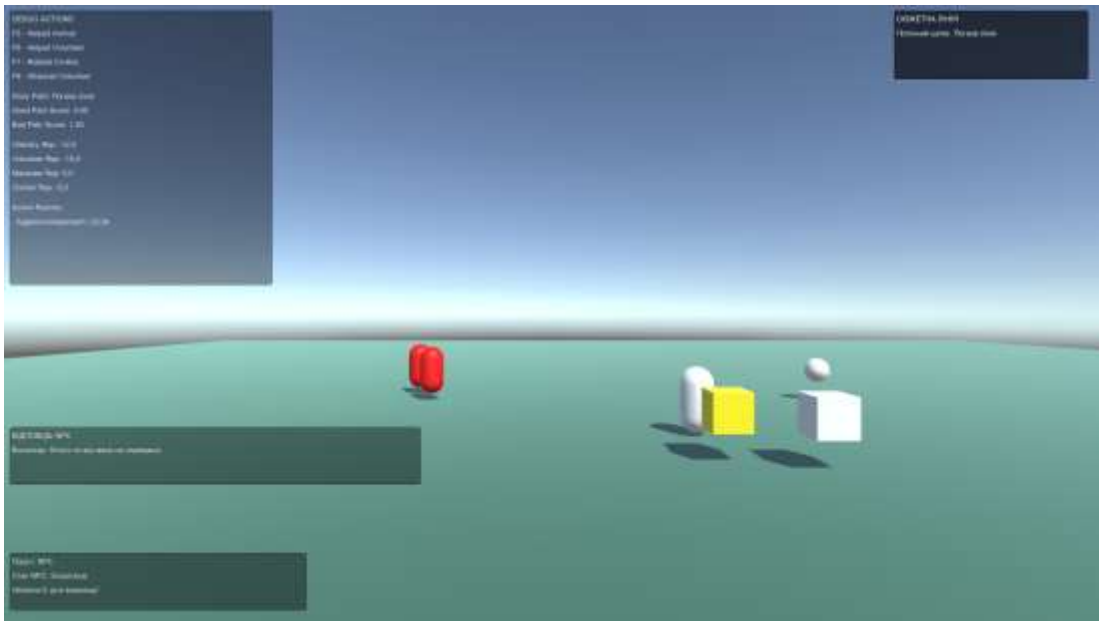
## А.2. Наближення гравця до NPC



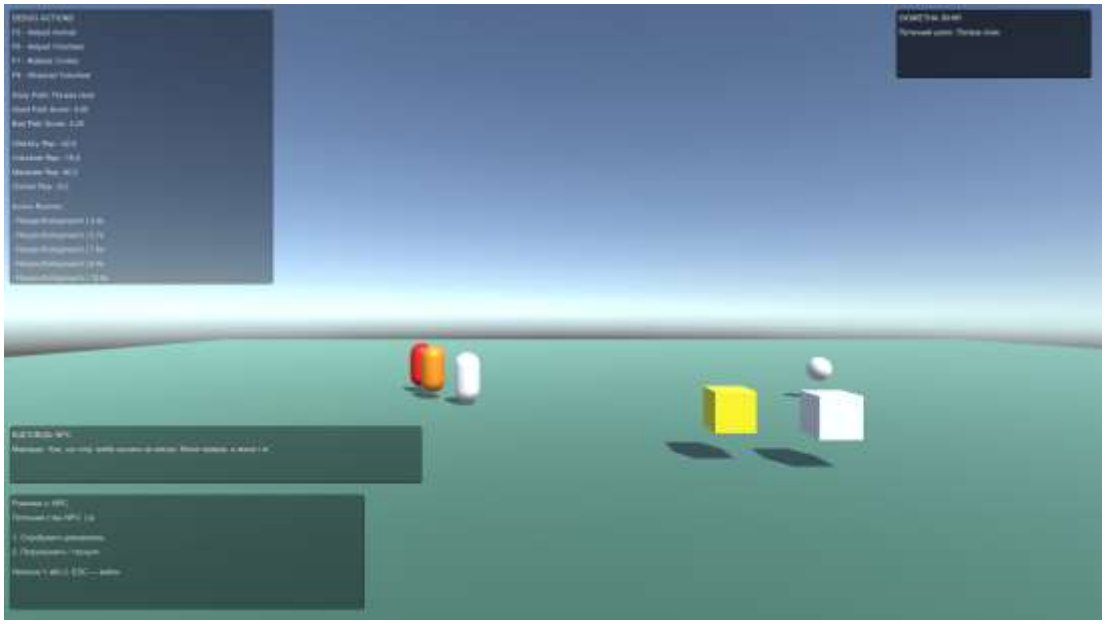
## А.3. Взаємодія з NPC



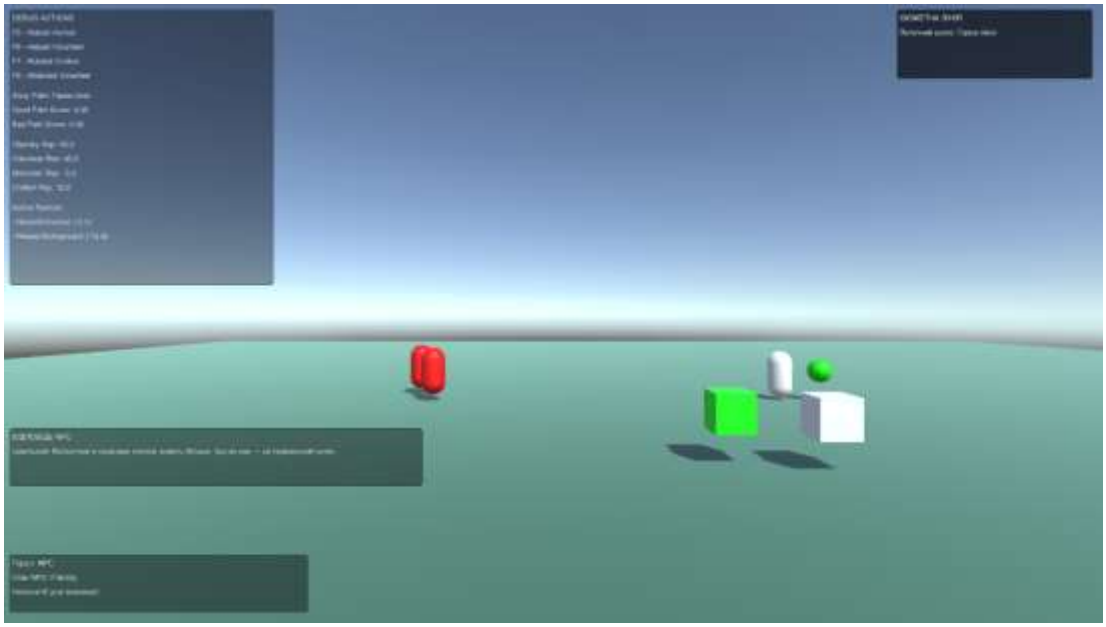
## А.4. Реакція волонтера на звернення при поганій лінії



## А.5. Реакція мародера на звернення при поганій лінії



## А.6. Реакція волонтера на звернення при гарній лінії



## А.7. Реакція цивільного на звернення при гарній лінії