МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Факультет
комп'ютерних наук та технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка

до дипломного проєкту (роботи)
магістра
(ступінь вищої освіти)

на тему **КОМП'ЮТЕРНА ГРА "ВТЕЧА" НА ПЛАТФОРМІ UNREAL ENGINE 5 З ВИКОРИСТАННЯМ КОМАНДНОГО МЕТОДУ РОЗРОБКИ**

Виконав: студент 2 курсу, групи КНТ-514м
спеціальності
123 Комп'ютерна інженерія
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерні системи та мережі

НІКІТЧЕНКО С.О.
(ПРІЗВИЩЕ та ініціали)
Керівник КИРИЧЕК Г.Г.
(ПРІЗВИЩЕ та ініціали)
Рецензент ХАРИТОНОВ О.Б.
(ПРІЗВИЩЕ та ініціали)

2025

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук та технологій
Кафедра «Комп'ютерні системи та мережі»
Ступінь вищої освіти магістерський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)

Освітня програма (спеціалізація) Комп'ютерні системи та мережі
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ
Зав. кафедри Кудерметов Р.К.
_____
" 15 " Жовтня 2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

НІКІТЧЕНКО Сергія Олександровича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Комп'ютерна гра "Втеча" на платформі Unreal Engine 5 з використанням командного методу розробки
керівник проєкту (роботи) кан. техн. наук, доцент, КИРИЧЕК Галина Григорівна
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом вищого навчального закладу від " 30 " жовтня 2025 року №491
2. Строк подання студентом проєкту (роботи) 23 грудня 2025 р
3. Вихідні дані до проєкту (роботи) наявні описані вимоги до комп'ютерної гри «Втеча» у жанрі survival escape, вибраний рушій Unreal Engine 5, визначені обмеження щодо продуктивності гри, вказані використовувані технології та інструменти: мова програмування C++, система візуального сценаріювання Blueprint, система контролю версій (Git), інтегроване середовище розробки (Visual Studio), формати ігрових ресурсів та проєктних файлів, цільова платформа (ПК на базі ОС Windows)
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1) Аналіз існуючих рішень і технологій; 2) Технології, засоби та інструменти розробки гри «Втеча»; 3) Проєктування комп'ютерної гри; 4) Реалізація гри «Втеча»; 5) Дослідження та тестування системи
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5	КИРИЧЕК Г.Г., доцент		
нормоконтроль	ЩЕРБАК Н.В., ст. викл.		

7. Дата видачі завдання 01 жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз жанру survival escape, сучасних ігор цього	05.10.2025 р.	
	типу та інструментів Unreal Engine 5, перегляд		
	результатів, виділення недоліків та переваг;		
2	Проєктування архітектури гри «Втеча»;	10.10.2025 р.	
3	Дослідження алгоритмів роботи основних підсистем;	15.10.2025 р.	
4	Реалізація гри на платформі Unreal Engine 5;	25.10.2025 р.	
5	Дослідження та тестування гри «Втеча»;	30.10.2025 р.	
6	Оформлення пояснювальної записки;	05.11.2025 р.	
7	Оформлення графічного матеріалу	10.11.2025 р.	
8	Оформлення допоміжного матеріалу	15.11.2025 р.	

Студент _____ **Сергій НІКІТЧЕНКО**
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ **Галина КИРИЧЕК**
(підпис) (ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

ПЗ: 117 с., 55 рис., 11 табл., 20 джерел.

SURVIVAL ESCAPE; UNREAL ENGINE 5; ДЕРЕВА ПОВЕДІНКИ; КОМП'ЮТЕРНА ГРА; НЕІГРОВІ ПЕРСОНАЖІ; СИСТЕМА ЗАПИТІВ СЕРЕДОВИЩА; ФРАКЦІЙНА ВЗАЄМОДІЯ; ШТУЧНИЙ ІНТЕЛЕКТ.

Мета роботи - реалізація та впровадження комп'ютерної гри на платформі Unreal Engine 5 з використанням командного методу розробки

Об'єкт дослідження – процес проєктування та реалізації геймплейного середовища у жанрі survival escape з модульною архітектурою ігрової логіки.

Предмет дослідження – методи, моделі та програмні засоби побудови архітектури ігрової логіки й адаптивної поведінки інтелектуальних агентів (неігрових персонажів), а також моделі командної розробки у середовищі Unreal Engine 5 з гібридним підходом C++/Blueprint.

Методи дослідження – теоретичний аналіз, об'єктно-орієнтоване проєктування, UML-моделювання, застосування кінцевих автоматів і дерев поведінки, візуальне програмування в Blueprint, використання гнучких методологій Agile/Scrum; технічні засоби – ігровий рушій Unreal Engine 5, мова програмування C++, інтегроване середовище Visual Studio, система керування версіями Git.

Результати – реалізовано ключові модулі гри «Втеча» з гібридною архітектурою C++/Blueprint, впроваджено систему штучного інтелекту на основі дерев поведінки та системи запитів середовища, створено систему фракційної взаємодії; проведено тестування, що підтвердило ефективність застосування методологій Agile/Scrum у процесі розробки.

Галузь використання – навчальні та дослідницькі розробки, інтерактивні симуляції, навчальні та дослідницькі розробки з використанням Unreal Engine 5.

ABSTRACT

Explanatory note to the master's work: 117 pages, 55 figures, 11 tables, 20 sources.

ARTIFICIAL INTELLIGENCE; BEHAVIOR TREES; COMPUTER GAME; ENVIRONMENT QUERY SYSTEM; FACTION INTERACTION; NON-PLAYER CHARACTERS; SURVIVAL ESCAPE GENRE; UNREAL ENGINE 5.

Purpose of the work – implementation and deployment of a computer game on the Unreal Engine 5 platform using a team-based development approach.

Object of the research – the process of designing and implementing a gameplay environment in the survival escape genre with a modular architecture of game logic.

Subject of the research – methods, models and software tools for building the architecture of game logic and adaptive behavior of intelligent agents (non-player characters), as well as models of team development in the Unreal Engine 5 environment using a hybrid C++/Blueprint approach.

Research methods – theoretical analysis, object-oriented design, UML modeling, application of finite state machines and behavior trees, visual scripting in Blueprint, use of Agile/Scrum methodologies; technical tools – Unreal Engine 5 game engine, C++ programming language, Visual Studio integrated development environment, Git version control system.

Results – key modules of the game "Escape" with a hybrid C++/Blueprint architecture have been implemented, an artificial intelligence system based on behavior trees and an environment query system has been introduced, and a faction interaction system has been created; testing has been carried out, confirming the effectiveness of applying Agile/Scrum methodologies in the development process.

Field of application – the computer game development industry, interactive simulations, educational and research solutions using Unreal Engine 5.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД – База даних

ООП – Об’єктно-орієнтоване програмування

ПЗ – Програмне забезпечення

СУВ – Система управління версіями

ШІ – Штучний інтелект

ВТ – Дерево поведінки

DLC – Розширюваний вміст

EQS – Система запитів середовища

FPS – Кадрів на секунду

FSM – Кінцевий автомат

HUD – Елементи інтерфейсу користувача

IDE – Інтегроване середовище розробки

NPC – Неігровий персонаж

UE5 – Рушій Unreal Engine 5

UI – Інтерфейс користувача

UML – Уніфікована мова моделювання

ВСТУП

Актуальність теми полягає в зростанні складності та вимог до сучасних комп’ютерних ігор, зокрема у жанрі survival escape, де поєднуються нелінійний геймплей, фракційна взаємодія, адаптивна поведінка NPC та високі очікування щодо якості користувацького досвіду. В умовах розвитку ігрової індустрії особливого значення набувають архітектурні підходи, що забезпечують модульність, масштабованість, повторне використання компонентів і можливість подальшого розширення ігрових систем, у тому числі за рахунок DLC. Це формує потребу у створенні гнучких архітектурних рішень, орієнтованих на інтеграцію різнорідних підсистем геймплею та ШІ.

Дипломна робота присвячена створенню комп’ютерної гри «Втеча» на базі рушія Unreal Engine 5 із використанням гібридного підходу до реалізації гри, у межах якого Blueprint застосовується для швидкого прототипування та візуальної логіки, а C++ – для реалізації продуктивних, масштабованих і гнучких підсистем. Метою роботи є обґрунтування архітектури ігрової логіки, побудови системи фракцій та поведінкових ШІ-модулів, реалізації ключових геймплейних механік та демонстрація можливостей Unreal Engine 5 для створення сучасних survival escape-ігор. Для досягнення поставленої мети необхідно розв’язати такі основні завдання: проаналізувати жанр survival escape та сучасні підходи до побудови ігрових систем; визначити вимоги до гри «Втеча» та сформувати її концепцію; розробити архітектурну модель гри з використанням UML-діаграм; реалізувати прототип гри з ключовими підсистемами на платформі Unreal Engine 5 із застосуванням Blueprint та C++; провести тестування роботи основних підсистем і проаналізувати отримані результати.

Структурно дипломна робота складається з п’яти основних розділів. У першому розділі виконано аналіз жанру survival escape, розглянуто сучасні інструменти розробки, підходи до побудови архітектури ігрових систем та

моделювання поведінки NPC. У другому розділі сформульовано вимоги до гри «Втеча», описано постановку задачі, ключові сценарії взаємодії гравця з ігровим середовищем і критерії оцінювання якості геймплею. Третій розділ присвячено розробці архітектурної моделі гри: описано модулі AI, фракційної системи, квестів, інвентаря, торгівлі та інтерфейсу користувача, наведено UML-діаграми, що формалізують структуру та взаємодію підсистем. У четвертому розділі розглянуто програмну реалізацію гри «Втеча» на платформі Unreal Engine 5 із використанням Blueprint та C++, включно з організацією ігрових рівнів, інтеграцією підсистем і забезпеченням можливості подальшого розширення. П’ятий розділ містить опис методики тестування, результати експериментальних досліджень роботи AI, фракційної взаємодії, геймплейних сценаріїв та аналіз продуктивності системи.

Результати, отримані в рамках дипломної роботи, можуть бути використані в комерційній розробці комп’ютерних ігор, під час створення навчальних демоверсій для дисциплін, пов’язаних із програмуванням та геймдизайном, а також як базова платформа для подальших досліджень у галузі поведінкового штучного інтелекту та архітектури ігрових систем. Окремі підходи й рішення, реалізовані в дипломній роботі, були апробовані в наукових публікаціях автора, присвячених командній організації розробки комп’ютерних ігор та аналізу сучасних кіберзагроз у цифрових середовищах [1].

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ І ТЕХНОЛОГІЙ

1.1 Характеристика жанру survival escape та аналіз сучасних ігор

Жанр survival escape є піджанром ігор виживання, що зосереджений на подоланні обмеженого простору та ухиленні від загроз за умов мінімальних ресурсів. Основна мета гравця полягає у знаходженні способу втечі зі складної ситуації, використовуючи механіки прихованості, дослідження середовища, стратегічного мислення та взаємодії з елементами оточення. На відміну від відкритих survival-ігор, де пріоритетом є довготривале існування, survival escape робить акцент на інтенсивності, напрузі та прогресивному розкритті локації [2]. На рисунку 1.1 схематично подано основні ознаки жанру survival escape, згруповані за просторовими, ігровими та поведінковими характеристиками.

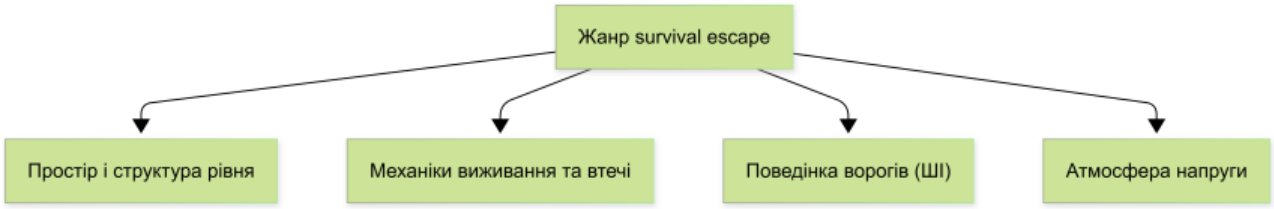


Рисунок 1.1 – Узагальнена схема ключових ознак жанру survival escap

До ключових ознак жанру належать:

– обмежений простір, у якому гравець поступово відкриває нові ділянки;

– значний акцент на уникненні прямого контакту з противниками;

– динамічна реакція ворогів на шум, сліди чи видимість гравця;

– пошук ключових предметів, інструментів або механізмів для відкриття шляхів;

– сюжетні тригери, що активуються залежно від дій користувача;

– підвищений рівень психологічної напруги, який формує відчуття

переслідування [3];

– нелінійність проходження, що забезпечує можливість альтернативних сценаріїв.

Аналіз сучасних представників жанру свідчить, що survival escape значною мірою залежить від поведінкових моделей штучного інтелекту та інтерактивності середовища [4]. У грі Outlast противники реагують на візуальні та звукові сигнали, патрулюють територію та здатні змінювати маршрути залежно від поведінки гравця. Гра Amnesia: The Dark Descent зробив акцент на механіці прихованості, де будь-який контакт із ворогом створює загрозу негайної поразки, а ШІ оцінює рівень шуму та освітлення. У серії The Escapists ключовою частиною ігрового процесу є пошук шляхів втечі через взаємодію з NPC, збір ресурсів та дотримання режиму, що ускладнює виявлення порушення. Ігри Alien: Isolation та Remothered демонструють складні ШІ-моделі, у яких противники використовують адаптивні стратегії переслідування та навчання на основі дій гравця. Приклади сучасних ігор, що належать до жанру survival escape, зведені у таблиці 1.1.

Сучасні survival escape-ігри зосереджуються на високому рівні непередбачуваності поведінки ворогів, що підсилює відчуття загрози. Застосування складних дерев поведінки, систем пошуку цілей та середовищних тригерів дозволяє створювати сценарії, в яких кожне рішення користувача впливає на подальший хід подій. У таких іграх гравець постійно перебуває в умовах невизначеності: маршрут патрулювання ворогів, їхня чутливість до шуму, реакція на попередні дії користувача та стан ігрового світу формують унікальні комбінації ситуацій навіть у межах одного рівня. Поширення процедурної генерації, фракційних систем і багаторівневих моделей загрози додатково посилює відчуття «живого» середовища, що реагує на гравця, а не просто програє заздалегідь запрограмовані скрипти. Окремої уваги заслуговує тенденція до поєднання stealth-механік, елементів психологічного хорору й ресурсного менеджменту, коли вибір між прямою конфронтацією, втечею чи тимчасовим переховуванням має довгострокові наслідки для стратегії

виживання. Завдяки цьому жанр зберігає актуальність і активно розвивається у напрямку більшої інтерактивності, нелінійності та глибшого занурення в атмосферу гри [5].

Таблиця 1.1 – Приклади ігор жанру survival escape

Назва гри	**Платформи**	**Основні механіки**
Alien: Isolation	PC, PlayStation, Xbox	Стелс, переховування, обмежені ресурси, дослідження закритого рівня
Outlast	PC, PlayStation, Xbox, Switch	Виживання без зброї, хованки в укриттях, втеча від переслідувачів, лінійні коридори
Amnesia: The Dark Descent	PC, консолі (пізніші порти)	Психологічний хорор, обмежені ресурси, світло/темрява як геймплейний ресурс, уникання прямого контакту
The Forest	PC, PlayStation	Виживання у відкритому світі, будівництво укриттів, крафт, дослідження, групи ворогів
Hello Neighbor	PC, консолі	Стелс, проникнення в будинок, циклічні спроби втечі/проникнення, експерименти з маршрутом
Escape the Backrooms (або аналогічні інді-ігри)	PC	Кооперативне виживання, дослідження лабіринтів, обмежена видимість, втеча від різних типів загроз

1.2 Постановка проблеми та визначення завдань дипломної роботи

Жанр survival escape передбачає динамічну взаємодію гравця з ворогами, середовищем та сюжетними подіями, що вимагає складної, адаптивної та оптимізованої архітектури ігрових систем. Попри значну кількість сучасних survival-ігри, більшість з них характеризуються рядом обмежень, пов’язаних із передбачуваністю поведінки NPC, лінійною структурою сценарію, недостатньою взаємодією з фракціями або статичною логікою прогресу. Це знижує рівень варіативності та реіграбельності, що є ключовими показниками якості ігор цього жанру. Узагальнений аналіз переваг та недоліків основних

підходів до реалізації ШІ в таких роботах наведено у таблиці 1.2. Додатково слід відзначити, що багато існуючих рішень зосереджуються лише на окремих аспектах, ігноруючи цілісну інтеграцію ШІ, наративу та фракційних механік, що відкриває простір для подальших досліджень і розробки більш комплексних моделей взаємодії.

Таблиця 1.2 – Переваги та недоліки підходів до ШІ

Підхід	**Переваги**	**Недоліки**
Behavior Trees (BT)	Чітка ієрархічна структура прийняття рішень; зручна візуальна модель; легко розширювати та модифікувати; добре поєднується з Blackboard та AI Perception в Unreal Engine 5; підтримується «з коробки» у UE5	Може ставати складним і заплутаним при великій кількості вузлів; важливо правильно проєктувати пріоритети й умови, інакше з'являється нестійка поведінка; відносно важче описувати довготривалі стани, ніж у деяких інших моделях
Environment Query System (EQS)	Автоматизований пошук найкращих позицій у просторі; зручний опис запитів високого рівня; інтеграція з NavMesh; можливість враховувати багато факторів (видимість, відстань, укриття тощо); добре працює разом з Behavior Trees	Потребує додаткових налаштувань і профілювання, щоб не перевантажувати систему; складні запити можуть бути ресурсомісткими; логіка EQS погано читається без досвіду; важко налагоджувати «на око»
Finite State Machines (FSM)	Проста й інтуїтивна модель; легко описати базові стани (idle, patrol, chase, attack); зручно для невеликих або простих NPC; добре підходить для чітко обмежених сценаріїв	Погано масштабується при збільшенні кількості станів і переходів; діаграма швидко перетворюється на «спагеті»; важко повторно використовувати та розширювати складну FSM; логіка станів часто жорстко зашита в код

Основна проблема, яку покликана вирішити дипломна робота, полягає у створенні інтерактивної survival escape-системи з нелінійною структурою, адаптивним штучним інтелектом і механізмами, що реагують на рішення гравця. Виникає необхідність у побудові модуля поведінки NPC, здатного оцінювати стан середовища, ухвалювати рішення за різних сценарних умов та взаємодіяти з фракційною логікою. Додатковою проблемою є реалізація системи, що забезпечує варіативність проходження через квести, сценарні гілки та динамічні події, які впливають на баланс сил у грі; порівняння застосування різних підходів до ШІ в сучасних іграх та у грі «Втеча» зведено у таблиці 1.3.

Таблиця 1.3 – Застосування підходів до ШІ в іграх та у грі «Втеча»

Підхід	**Типові сценарії використання**	**Чому підходить / не підходить для гри «Втеча»**
Behavior Trees (BT)	Поведінка NPC-охоронців, патрулювання, переслідування гравця; реакція на шум, зір, тригери рівня; перемикання між режимами «спокій / підозра / тривога»; координація простих групових дій	Дуже добре підходить як основний інструмент для логіки NPC у «Втечі», дозволяє будувати складну, але керовану поведінку для різних типів персонажів, зберігаючи наочність і можливість швидкої зміни в Blueprint
Environment Query System (EQS)	Вибір точки для пошуку гравця після втрати контакту; пошук укриттів; вибір позицій для блокування маршрутів; розміщення NPC у вигідних тактичних позиціях; оцінка небезпеки зон	Ідеально доповнює Behavior Trees як інструмент просторового прийняття рішень. Підходить для реалізації пошуку, засідок, оптимальних маршрутів та зон патрулювання, але вимагає обмеження частоти запитів і продуманої оптимізації
Finite State Machines (FSM)	Простий ШІ для допоміжних NPC; тригерні об’єкти з малою кількістю станів (увімкнено / вимкнено / зламано); базові вороги з обмеженим набором дій; скриптовані події	Може використовуватися локально для окремих підсистем або простих NPC, але як основний підхід до ШІ у «Втечі» не підходить через слабку масштабованість. Логічно застосовувати FSM для простих станів компонентів, тоді як загальна поведінка будується на BT + EQS

Загальнену схему постановки проблеми та основних завдань дипломної роботи подано на рисунку 1.2.

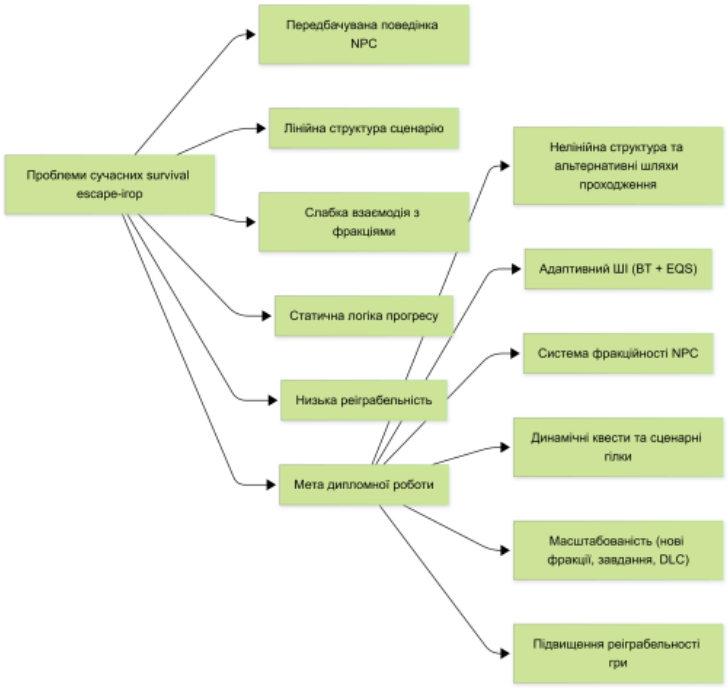


Рисунок 1.2 – Схема постановки проблеми та завдань дипломної роботи

У межах дипломної роботи визначено такі завдання:

– побудова моделі геймплейної логіки, що забезпечує нелінійність та альтернативні шляхи проходження;

– реалізація архітектури штучного інтелекту на основі поведінкових дерев і запитів середовища;

– створення системи фракційності, де ставлення NPC змінюється відповідно до рішень гравця;

– впровадження механізмів взаємодії з предметами, зонами середовища та квестовими тригерами;

– забезпечення можливості масштабування через додавання нових фракцій, завдань, сценарних ліній або DLC;

– підвищення реіграбельності через адаптивну реакцію системи на стиль гри користувача.

Таким чином поставлена проблема визначає потребу у створенні комплексної ігрової системи з модульною архітектурою, яка поєднує механіки втечі, дослідження, прихованості та соціальної взаємодії з динамічним ШІ та розгалуженою сюжетною моделлю [6].

1.3 Аналіз підходів до реалізації складної ШІ-логіки в іграх

Системи штучного інтелекту в сучасних відеоіграх відіграють ключову роль у формуванні інтерактивного середовища, поведінки противників та загальної динаміки геймплею. Для жанру survival escape, що базується на напрузі, непередбачуваності та адаптивній реакції ворогів на дії гравця, особливо важливо застосовувати моделі прийняття рішень, здатні підтримувати варіативність сценаріїв і високий рівень реіграбельності.

Серед найпоширеніших підходів до побудови ШІ у сучасній ігровій індустрії виокремлюють поведінкові дерева (Behavior Trees, BT), системи

запитів до середовища (Environment Query System, EQS) та автомати кінцевих станів (Finite State Machines, FSM). Кожен із цих підходів забезпечує власний механізм контролю логіки NPC, має специфічні переваги й обмеження та по-різному впливає на архітектуру гри, складність підтримки й подальшого масштабування системи. Узагальнену схему взаємодії основних підходів до реалізації ШІ-логіки в survival escape-іграх наведено на рисунку 1.3.

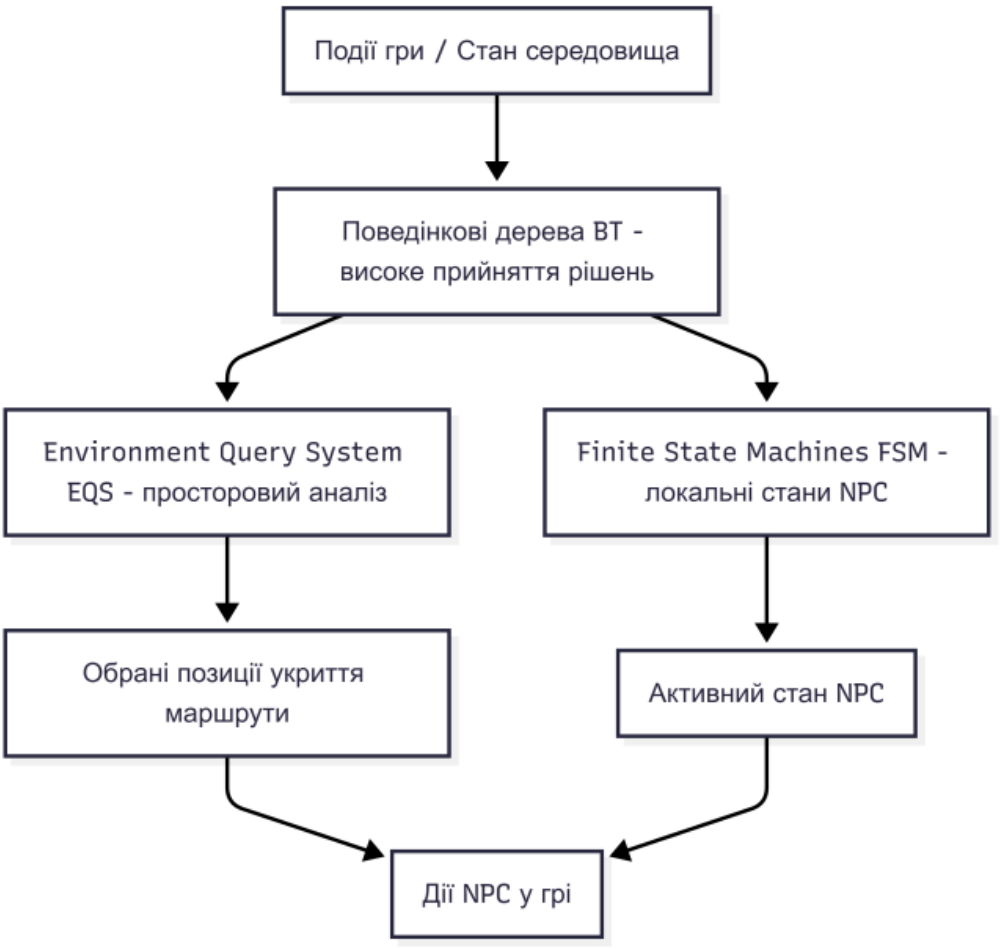


Рисунок 1.3 – Узагальнена схема архітектури ШІ survival escape-ігри із застосуванням Behavior Trees, EQS та FSM

Поведінкові дерева забезпечують ієрархічну структуру прийняття рішень, у межах якої NPC послідовно аналізує умови та виконує дії залежно від контексту. Структура дерева складається з вузлів перевірок та вузлів дій, що дає змогу гнучко комбінувати прості поведінкові шаблони у складні сценарії. Завдяки цьому BT активно використовуються в сучасних комерційних іграх. У

Alien: Isolation ШІ xenomorph адаптується до поведінки гравця, аналізуючи маршрути й реагуючи на зміну умов; у The Last of Us рекурсивні поведінкові дерева визначають реакції ворогів на звук, укриття, втрату союзників або появу загроз; у Hitman подібні структури керують логікою виявлення підозрілої поведінки, пошуку гравця та повернення до патрулювання. Приклад фрагмента дерева поведінки ворожого NPC наведено на рисунку 1.4.

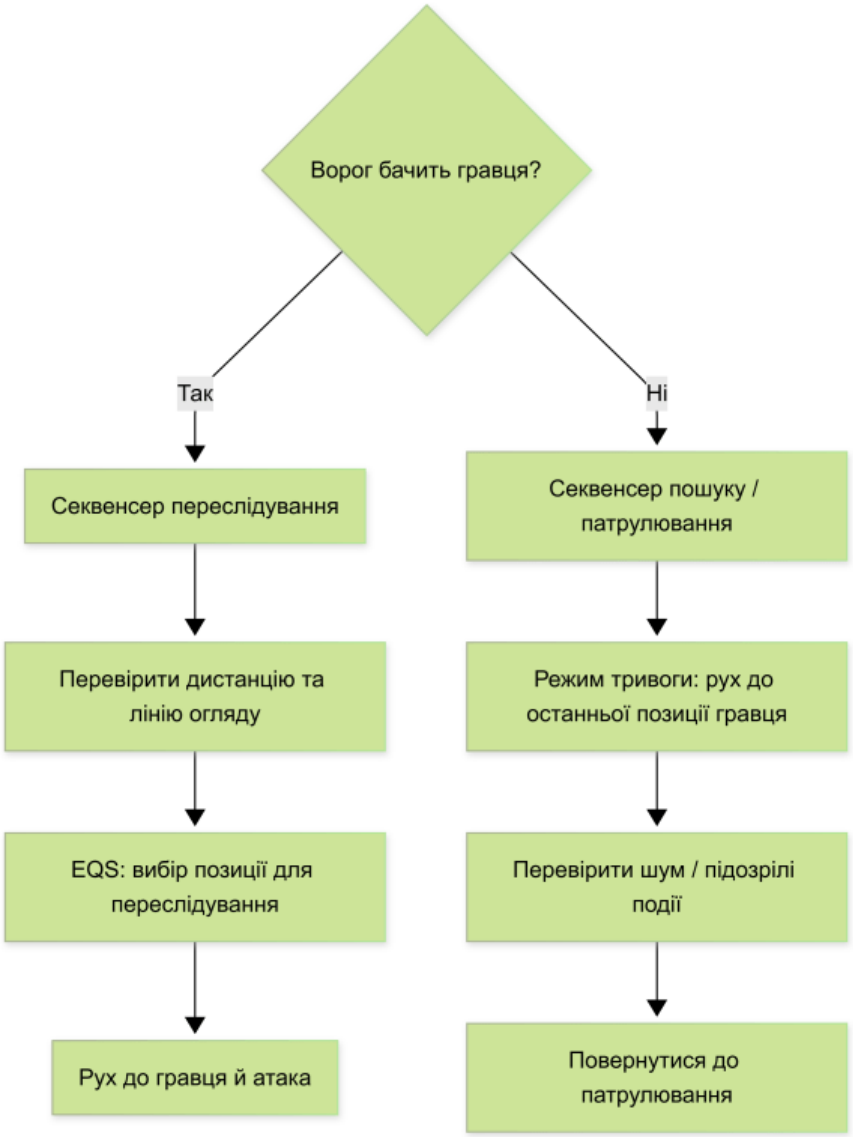


Рисунок 1.4 – Приклад фрагмента дерева поведінки ворожого NPC у survival escape-грі

Серед ключових переваг поведінкових дерев варто виділити чітку структурованість логіки, можливість поетапного масштабування без

радикальної перебудови, підтримку складних рішень за рахунок комбінації селекторів і секвенсерів, а також тісну інтеграцію з модульними системами рушіїв, зокрема Unreal Engine 5. Це робить ВТ одним із базових інструментів для побудови адаптивного ШІ в survival escape-іграх, де вороги повинні реагувати на дії гравця, змінювати маршрути, створювати сценарії напруги та уникати повторюваних шаблонів поведінки.

Environment Query System (EQS) є спеціалізованим інструментом рушія Unreal Engine, який доповнює поведінкові дерева механізмом просторового аналізу середовища. На відміну від класичних алгоритмів пошуку шляху, EQS функціонує як система запитів: NPC формує множину потенційних позицій, оцінює їх за заданими критеріями та обирає оптимальну. Типові сценарії застосування EQS охоплюють пошук укриттів під час бою, визначення найкращої точки для переслідування гравця, побудову маршрутів патрулювання, а також виявлення зон, з яких ШІ може перехопити гравця або зібрати необхідну інформацію.

За допомогою генераторів і фільтрів EQS дає змогу враховувати відстань до гравця, наявність прямої або непрямої лінії видимості, рівень загрози в окремих областях, а також недоступні чи небезпечні ділянки сцени. У результаті рішення NPC стають більш природними: противник у survival escape-грі може самостійно обирати позиції для засідки, обхідні маршрути або точки відступу, адаптуючись до стилю гри користувача. Це суттєво підвищує непередбачуваність ігрових ситуацій та посилює відчуття напруги. Приклад візуалізації результатів EQS-запиту для вибору оптимального укриття наведено на рисунку 1.5. Додатково отримані дані можуть використовуватися для налагодження та профілювання ШІ, порівняння різних конфігурацій рівнів і демонстрації роботи системи під час презентації або захисту дипломної роботи перед експертною комісією та ширшою аудиторією.

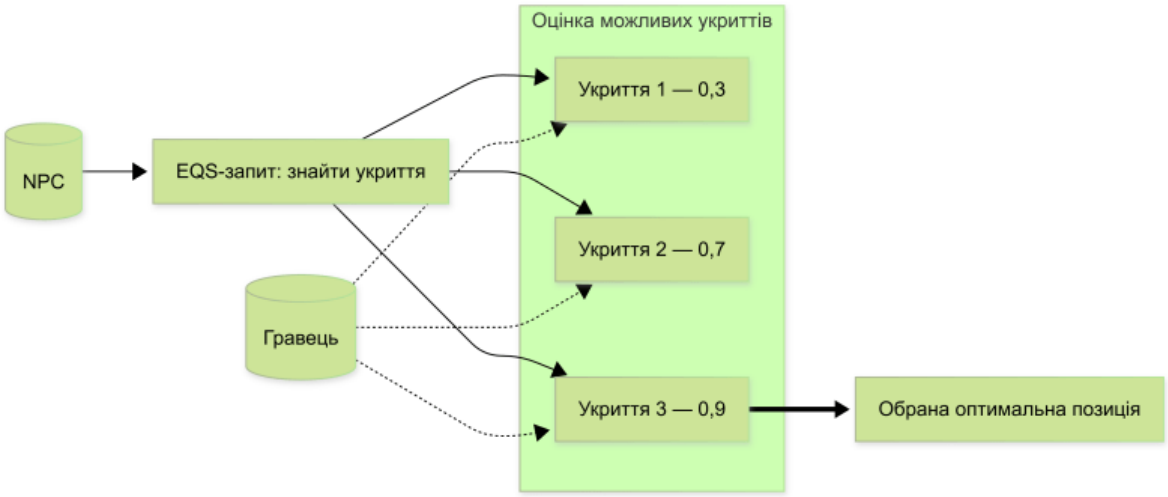


Рисунок 1.5 – Візуалізація результатів EQS-запиту для вибору оптимального укриття ворогом

Автомати кінцевих станів (Finite State Machines, FSM) належать до класичних і найпростіших засобів побудови ШІ-логіки. У межах FSM NPC перебуває в одному з фіксованих станів, таких як патрулювання, переслідування, пошук, відступ, атака, а переходи між ними відбуваються за попередньо визначених умов. Перевагами такого підходу є простота реалізації, висока передбачуваність поведінки та невеликі вимоги до обчислювальних ресурсів, що робить FSM доцільними для другорядних NPC, службових скриптів або допоміжних технічних механік. Типову структуру автомата кінцевих станів для ворожого NPC наведено на рисунку 1.6.

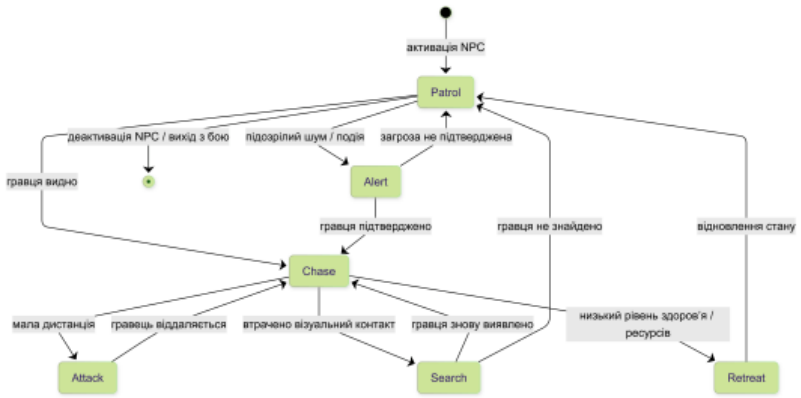


Рисунок 1.6 – Приклад автомата кінцевих станів для ворожого NPC з базовими поведінковими режимами

Водночас у контексті складних ігор із нелінійним геймплеєм автомати кінцевих станів демонструють низку суттєвих обмежень. Зі зростанням кількості станів різко ускладнюється структура переходів, модель погано масштабується, важко реалізувати комбінування кількох мотивацій одночасно, а поведінка NPC часто стає повторюваною та механічною. FSM погано пристосовані до опису багаторівневої, контекстно залежної логіки, яка є критично важливою для survival escape-ігри.

Таким чином, аналіз підходів до реалізації ШІ-логіки в survival escape-іграх показує, що найбільш ефективними є комбіновані архітектури, у яких поведінкові дерева задають високорівневу структуру прийняття рішень, EQS забезпечує просторово обґрунтований вибір цілей і позицій, а автомати кінцевих станів використовуються для локальних станів або простих підсистем. Поєднання цих підходів дає змогу досягти балансу між продуктивністю, передбачуваністю та адаптивністю, що є необхідним для створення напруженого, варіативного та реіграбельного геймплею в межах жанру survival escape.

1.4 Аналіз ігрових рушіїв

Ігровий рушій визначає архітектуру, продуктивність та технічні можливості майбутньої гри. Вибір рушія суттєво впливає на якість графіки, складність реалізації штучного інтелекту, інтерактивність середовища та масштабованість системи. Для гри жанру survival escape особливо важливими є підтримка складних ШІ-моделей, гнучка система подій, наявність засобів профілювання продуктивності, високий рівень візуалізації та можливість швидкої розробки ітеративних прототипів із мінімальними витратами часу. Розглянемо три популярні рушії: Unity, Unreal Engine 5 та Godot, що активно застосовуються у сучасній індустрії [7].

Порівняння зазначених рушіїв доцільно здійснювати за такими критеріями, як підтримка 3D-графіки та сучасних рендерингових технологій, інструменти для побудови ШІ-поведінки, наявність вбудованих систем анімації й роботи з фізикою, зручність інтеграції з системами контролю версій, а також активність спільноти розробників та доступність навчальних матеріалів. Це дозволяє обґрунтовано обрати рушій, що найкраще відповідає вимогам survival escape-ігри та обмеженням командної розробки. Узагальнене порівняння рушіїв Unity, Unreal Engine 5 та Godot за основними характеристиками для survival escape-ігри наведено на рисунку 1.7. Додатково враховується ліцензійна модель, вимоги до апаратних ресурсів, підтримка мультиплатформенності та можливість подальшого масштабування гри, що особливо важливо для довготривалої підтримки та розширення гри в різних сценаріях використання та навчальних кейсах.

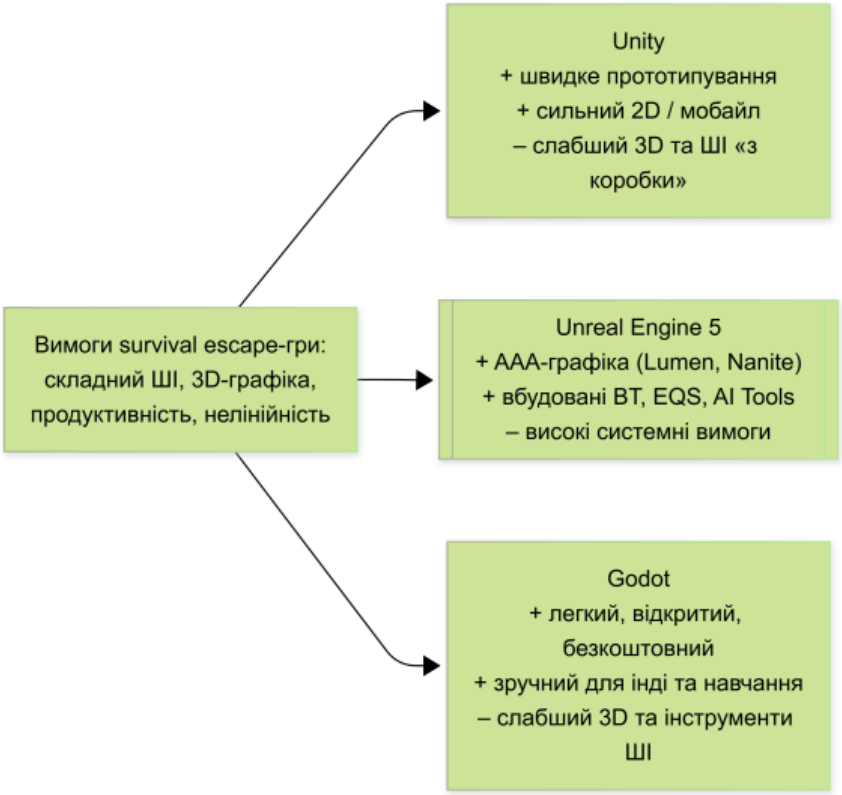


Рисунок 1.7 – Порівняльна схема характеристик рушіїв Unity, Unreal Engine 5 та Godot для survival escape-ігри

1.4.1 Unity

Unity є одним із найпоширеніших рушіїв у розробці ігор різного масштабу. Його основними сильними сторонами є невисокі порогові вимоги, гнучкість та велика екосистема інструментів. Unity забезпечує можливість створення 2D і 3D ігор, підтримує кросплатформеність і має широкий набір плагінів, які спрощують впровадження нових механік.

Основні переваги Unity:

– зручна інтеграція зі сторонніми бібліотеками завдяки C#;

– велика кількість шаблонів та інструментів для швидкого прототипування;

– широка база навчальних матеріалів та підтримка спільноти;

– ефективність у розробці мобільних та невеликих 3D-ігор.

Недоліки Unity у контексті survival escape-гра:

– обмежена продуктивність у складних 3D-сценах;

– графічні можливості без додаткових шейдерів значно поступаються рушіям високого класу;

– складніше створювати адаптивні ШІ-системи без сторонніх інструментів;

– менш інтегрована робота з Behavior Trees та EQS, які доводиться реалізовувати вручну.

Unity добре підходить для інді та мобільних ігор, але потребує значних доопрацювань при створенні ігор із високою динамікою ШІ, складною графікою та нелінійними сценаріями [8]. У типовому робочому процесі розробнику доводиться додатково інтегрувати сторонні фреймворки для поведінки NPC, оптимізувати рендеринг під складні сцени та вручну підтримувати консистентність логіки гілок сюжету. Це збільшує час налаштування, ускладнює підтримку гри на пізніх етапах і робить Unity менш привабливим варіантом для великомасштабних survival escape-рішень порівняно з рушіями, де зазначені інструменти вже частково інтегровані на рівні базової архітектури.

1.4.2 Unreal Engine 5

Unreal Engine 5 є одним із найпотужніших рушіїв для створення сучасних 3D-ігор, включаючи AAA-рівень. Його сильні сторони зосереджені на графічних технологіях, високій продуктивності та інтегрованих засобах роботи зі складними ШІ-системами. UE5 використовує C++, а також має власну систему візуального програмування Blueprint, яка дозволяє швидко створювати прототипи та будувати ігрову логіку [9].

Основні переваги Unreal Engine 5:

– Lumen забезпечує реалістичне глобальне освітлення та відображення;

– Nanite дозволяє працювати з моделями високої полігональності без втрати FPS;

– інтегровані інструменти ШІ: Behavior Tree, Blackboard, EQS;

– підтримка DataTables та Gameplay Tags для побудови систем фракцій та сценаріїв;

– міцна архітектура для розгалуженої логіки та масштабування.

Для survival escape гри UE5 є оптимальним варіантом завдяки:

– здатності створювати динамічні інтерактивні сцени з високою деталізацією;

– готовій системі поведінкових дерев та запитів середовища без сторонніх плагінів;

– підтримці змішаного підходу (Blueprint + C++), що зменшує час розробки;

– можливості реалізувати нелінійну логіку, фракції, квести та складні сценарії.

Недоліки рушія:

– високі системні вимоги;

– крута крива навчання для новачків;

– великі розміри ігор і потреба в оптимізації.

У контексті дипломної роботи Unreal Engine 5 повністю відповідає

вимогам жанру survival escape та забезпечує необхідний набір інструментів для реалізації складного штучного інтелекту й нелінійної структури проходження. Використання вбудованих AI-модулів (зокрема Behavior Trees та EQS) разом із можливістю комбінувати C++ і Blueprint дозволяє гнучко налаштовувати логіку NPC, розширювати її під вимоги фракційної взаємодії та підтримувати сценарні розгалуження без критичних обмежень щодо масштабування і продуктивності.

1.4.3 Godot

Godot є відкритим та легким рушієм, орієнтованим переважно на інді-розробку. Він має власну мову GDScript, що нагадує Python, і надає високий рівень доступності для початківців. Godot підтримує 2D і 3D-ігри, однак його можливості у сфері тривимірної графіки та складних ШІ значно поступаються іншим рушіям [10].

Переваги Godot:

– відкритий вихідний код та безкоштовне використання;

– легкий і швидкий навіть на слабких пристроях;

– зручність для невеликих ігор та навчальних цілей;

– спрощений інтерфейс редактора.

Недоліки у контексті survival escape:

– обмежена продуктивність у великих 3D-сценах;

– недостатньо потужна система ШІ порівняно з UE5;

– відсутність вбудованих Behavior Trees та EQS, які потрібно реалізовувати вручну;

– недостатня кількість ресурсів, інструментів та прикладів для складних AAA-ігор.

Godot є чудовим рушієм для простих інді-ігор або навчальних робіт, оскільки він забезпечує легкий поріг входу, зрозумілий робочий процес і достатній набір базових інструментів для швидкого створення прототипів. Проте для створення масштабної survival escape-гри з динамічним ШІ, великою кількістю взаємодіючих NPC і нелінійним сценарієм його можливостей часто виявляється недостатньо: можуть виникати обмеження щодо готових

інструментів для складної AI-логіки, профілювання продуктивності та підтримки високорівневих систем у великих проєктах, що ускладнює розробку та подальше масштабування [11].

1.5 Обґрунтування актуальності створення гри «Втеча»

Сучасна індустрія відеоігор демонструє стійкий інтерес до ігор, що поєднують елементи виживання, нелінійного сюжету та інтелектуальної поведінки противників. Більшість існуючих ігор цього типу зосереджені або на хорор-компоненті, або на вузько спрямованій механіці втечі, що обмежує глибину взаємодії з NPC, варіативність сценаріїв та можливість адаптивної реакції світу на дії гравця. Це створює обґрунтовану нішу для розробки гри, у якій поєднано фракційну систему, нелінійний сюжет, поведінкову ШІ-модель та можливість подальшого розширення [12].

Актуальність створення гри «Втеча» зумовлена кількома групами чинників: технологічними, методичними та індустріальними. З технологічної точки зору, використання Unreal Engine 5 дає змогу дослідити й застосувати сучасні можливості рушія, зокрема Behavior Trees, EQS, DataTables, систему тегів та комбіновану модель Blueprint + C++. Узагальнені характеристики відомих ігрових рушіїв, серед яких розглядається UE5, наведено у таблиці 1.4, а порівняльний аналіз цих рушіїв за ключовими параметрами (підтримка ШІ-систем, інструменти для роботи з даними, можливості масштабування) подано у таблиці 1.5. Це дозволяє не лише обґрунтувати вибір Unreal Engine 5 як базової платформи, а й показати його переваги у контексті побудови модульної архітектури, що підтримує розширення через нові сценарії, фракції та DLC. Методичний аспект пов’язаний із тим, що дипломна робота слугує прикладом комплексного застосування об’єктно-орієнтованого підходу, UML-моделювання [13], проєктування ігрових систем та командної розробки у

реалістичному сценарії, наближеному до умов комерційного геймдеву [14].

Таблиця 1.4 – Основні характеристики відомих рушіїв

Рушій	**Сильні сторони**	**Обмеження**
Unity	Підтримка 2D та 3D, зручний для мобільних ігор, велика екосистема плагінів	Менш реалістична графіка за замовчуванням, потребує додаткового доопрацювання шейдерів для high-end survival
Unreal Engine 5	Високоякісна графіка (Lumen, Nanite), Behavior Trees, C++ інтеграція, Blueprint, оптимізована обробка ШІ	Вищі апаратні вимоги, складніший поріг входу
Godot Engine	Відкритий вихідний код, легкий, зручний для інді-ігор	Обмежена підтримка складних ШІ-систем та великомасштабних 3D-ігор
CryEngine	Дуже сильна графічна складова, реалістична фізика, орієнтований на шутери	Важко кастомізується, має меншу спільноту та складніший пайплайн розробки

З індустріальної точки зору, попит на ігри з нелінійним проходженням, варіативними фіналами та адаптивним штучним інтелектом продовжує зростати. Гравці очікують від ігор не лише високої якості графіки, а й осмисленої взаємодії зі світом, де дії мають наслідки, а поведінка NPC не зводиться до простих скриптів. «Втеча» орієнтована на задоволення цих очікувань за рахунок фракційної системи, репутаційної моделі, торгівлі та багаторівневої ШІ-логіки. Додатково дослідження гри «Втеча» має навчальне значення: її результати можуть бути використані як основа для подальших досліджень у галузі інтерактивних симуляцій, адаптивних систем прийняття рішень та побудови ігрових прототипів у навчальному процесі, а також для розробки практичних кейсів, що демонструють застосування сучасних технологій геймдизайну та ШІ в освітньому середовищі [15].

Таким чином, створення гри «Втеча» є актуальним як з позиції розвитку геймдев-індустрії, так і з точки зору дослідження сучасних підходів до проєктування і реалізації складних інтерактивних систем. Дипломна робота поєднує науково-практичний інтерес, можливість апробації актуальних

технологій та перспективу подальшого розширення і вдосконалення, а також може слугувати базою для створення навчальних матеріалів, лабораторних робіт і демонстраційних прототипів у галузі комп'ютерних наук та інженерії програмного забезпечення, що підвищує його цінність для академічної спільноти та практичної підготовки фахівців.

Таблиця 1.5 – Порівняльний аналіз за ключовими параметрами

Критерій	**Unity**	**Unreal Engine 5**	**Godot**
Графічна деталізація	Середня	Висока (Lumen, Nanite)	Низька / середня
Робота з ШІ	FSM	Behavior Trees + EQS	Потрібна кастомізація
Змішана логіка	Visual Scripting слабший	Blueprint + C++	GDScript
Масштабність гри	Середня	Висока	Низька
Підтримка фракційної логіки та нелінійного сюжету	Реалізується вручну	Підтримується через Behavior Tree + DataTables	Обмежено
Придатність для survival escape	Потребує значного допрацювання	Оптимальний варіант	Обмежено
Спільнота та ресурси	Дуже велика	Дуже велика	Середня

1.6 Висновки до розділу

У даному розділі проаналізовано жанр survival escape, особливості його геймплейної моделі та вимоги до штучного інтелекту, необхідного для формування напруженого й варіативного ігрового процесу. Визначено ключові чинники реіграбельності: нелінійність розвитку подій, адаптивна поведінка NPC, інтерактивність середовища та залежність сценарних змін від дій гравця.

Сформульовано проблему обмеженості традиційних підходів, що часто призводять до передбачуваної поведінки супротивників, надмірної лінійності проходження та недостатньої інтеграції соціальної динаміки у вигляді

фракційних і репутаційних механік. У цьому контексті визначено завдання дипломної роботи, спрямовані на створення гри з модульною архітектурою, розвиненою ШІ-логікою та нелінійною структурою проходження.

Проведено аналіз основних технічних підходів до побудови ШІ в іграх: кінцевих автоматів, дерев поведінки та систем запитів до середовища (EQS). Показано, що Behavior Trees у поєднанні з EQS забезпечують вищий рівень адаптивності та контекстної реакції порівняно з класичними FSM, які доцільно застосовувати переважно для локальних, простіших сценаріїв.

Додатково виконано порівняльний аналіз рушіїв Unity, Unreal Engine 5 та Godot. Обґрунтовано вибір Unreal Engine 5 як найбільш придатної платформи для реалізації survival escape-гри завдяки потужним графічним можливостям, наявності вбудованих інструментів ШІ, підтримці комбінованого підходу Blueprint + C++ та розвиненій інфраструктурі для побудови складних ігрових систем.

Отримані результати аналізу підтверджують актуальність створення гри «Втеча» як програмного засобу, що відповідає сучасним тенденціям індустрії та дає змогу комплексно дослідити підходи до реалізації нелінійної, фракційно орієнтованої, ШІ-насиченої ігрової системи. Сформоване теоретичне підґрунтя використовується у наступних розділах для проєктування архітектури гри, визначення її структурних компонентів і подальшої програмної реалізації.

2 ТЕХНОЛОГІЇ, ЗАСОБИ ТА ІНСТРУМЕНТИ РЕАЛІЗАЦІЇ ГРИ «ВТЕЧА»

Реалізація гри «Втеча» потребує застосування технологій, які здатні забезпечити одночасно високу продуктивність, гнучкість архітектури та можливість швидкого прототипування ігрових механік. Обраний рушій Unreal Engine 5 підтримує змішаний підхід до розробки, поєднуючи класичне

програмування мовою C++ із візуальною системою Blueprint. Такий підхід є доцільним для survival escape-гри, де поєднуються складні ШІ-системи, фракційна логіка, інтерактивне середовище та багаторівневий користувацький інтерфейс. У цьому розділі розглядаються основні технології та інструменти, що використовуються для реалізації гри, з акцентом на виборі формату побудови логіки гри та взаємодії між її модулями, а також на оцінюванні їх придатності для довгострокового супроводу, масштабування функціональності та інтеграції з інструментами тестування і оптимізації продуктивності.

2.1 Порівняння Blueprint та C++

Unreal Engine 5 підтримує два основні підходи до створення ігрової логіки: візуальне програмування за допомогою Blueprint та текстове програмування на C++. Обидва інструменти інтегровані в єдине середовище і можуть використовуватись паралельно, що формує гібридну архітектуру. Вибір між Blueprint та C++ впливає на швидкість розробки, можливості оптимізації, гнучкість змін та складність підтримки програмної системи.

Blueprint є візуальною скриптовою системою, яка дозволяє будувати логіку через граф вузлів. Вона є зручною для: швидкого прототипування механік; створення UI-елементів та екранних віджетів; організації простих квестових сценаріїв і тригерів; опису взаємодії з окремими об’єктами оточення. Перевага Blueprint полягає в тому, що зміни можна вносити без повної перекомпіляції всієї гри, а результат одразу перевіряти у редакторі. Це робить систему особливо корисною для етапів активних експериментів із геймплеєм та балансом.

C++ у Unreal Engine використовується для реалізації низькорівневої та ресурсомісткої логіки. Він є доцільним для: побудови ядра геймплею; реалізації складних ШІ-алгоритмів і фракційних систем; управління даними, кешування

та оптимізації; реалізації нестандартних механік, які виходять за межі можливостей Blueprint. На відміну від Blueprint, C++ забезпечує вищу продуктивність, більший контроль над пам’яттю та можливість створення власних компонентів, які згодом можуть використовуватися у Blueprint як готові модулі.

Додатково C++ є зручнішим для інтеграції з системами контролю версій, написання модульних тестів і підтримки довготривалого життєвого циклу гри. Саме на C++ доцільно реалізовувати підсистеми, що інтенсивно працюють із ресурсами: обробку великої кількості NPC, складні запити до фракційних та геймплейних даних, кастомні менеджери збережень, системи профілювання та логування. Такий підхід дає змогу поєднати гнучкість візуального сценарного налаштування в Blueprint із надійністю й масштабованістю «движка» логіки на C++.

Порівнюючи ці підходи у контексті гри «Втеча», доцільно виділити їхні основні характеристики; узагальнене порівняння наведено у таблиці 2.1 а схематичний розподіл обов’язків між Blueprint та C++ у структурі ігрової логіки показано на рисунку 2.1:

– швидкість прототипування та зміни логіки у Blueprint є значно вищою, ніж у C++;

– продуктивність та масштабованість рішень на C++ вища, особливо для ШІ, фракцій та складних систем;

– Blueprint краще підходить для візуально орієнтованих завдань і сценарної логіки;

– C++ є основою для побудови ключових модулів і забезпечує узгодженість архітектури.

Таким чином, у дипломній роботі, присвяченій розробці гри «Втеча», обґрунтовано обрано змішаний підхід. Blueprint використовується для: реалізації UI; побудови квестових гілок; тригерів взаємодії; сценарних подій. C++ застосовується для: реалізації ШІ-системи на основі Behavior Trees та EQS; фракційної логіки та репутації; управління даними предметів, NPC та

параметрів світу; оптимізації виконання ресурсомістких обчислень. Такий розподіл дозволяє поєднати переваги обох інструментів, зберігаючи баланс між гнучкістю розробки та продуктивністю системи, що додатково спрощує подальший супровід коду, полегшує залучення нових розробників до розробки гри та масштабування функціональності без кардинальної перебудови архітектури системи.

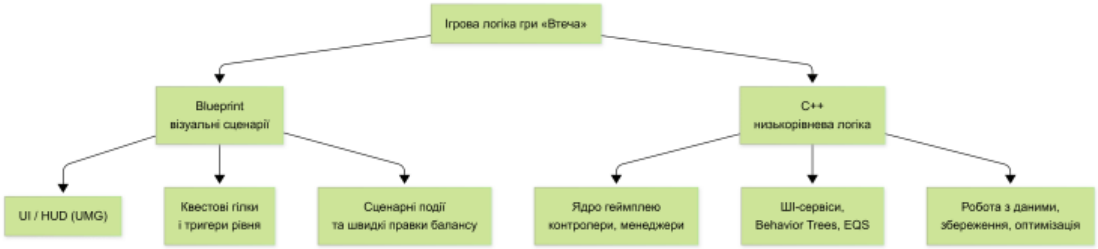


Рисунок 2.1 – Розподіл відповідальностей між Blueprint та C++ в ігровій логіці гри «Втеча» в Unreal Engine 5

Таблиця 2.1 – Порівняння Blueprint та C++ у грі «Втеча»

Критерій	**Blueprint**	**C++**
Швидкодія	Виконання повільніше, ніж нативний код; достатньо для більшості геймплейних задач, але небажано для «важких» обчислень чи масового ШІ	Висока продуктивність, нативний код, оптимальний для ресурсомістких частин системи
Гнучкість змін під час розробки	Дуже зручно швидко змінювати логіку без перекомпіляції; добре підходить для прототипування та дрібних правок	Будь-які зміни потребують перекомпіляції; внесення правок повільніше, особливо для нетехнічних учасників команди
Зручність відлагодження	Візуальний дебаг, можливість крокового виконання, наочне відстеження даних; але великі графи важко читати	Потужні засоби налагодження на рівні коду (breakpoint, профілювання, логування), але вхідний поріг вищий
Придатність для ШІ	Зручно описувати високорівневу логіку поведінки, працювати з Behavior Trees, Blackboard, подіями і тригерами	Оптимально реалізовувати базові сервіси ШІ, контролери, обробку даних і складні алгоритми
Придатність для UI та геймплейних івентів	Ідеально підходить для UMG, анімацій, тригерів, реакцій на події; легко пов’язувати дані з віджетами	Логіка даних і моделі можуть бути реалізовані в C++, але безпосередньо будувати UI на C++ складніше й повільніше в розробці
Розподіл ролей у команді	Доступний для дизайнерів, левел-дизайнерів та менш досвідчених програмістівї	Орієнтований на технічних розробників; вимагає впевнених знань C++ та структури UE5

2.2 Архітектурні підходи до побудови геймплейних систем

Побудова геймплейних систем у сучасних іграх вимагає архітектурних підходів, які забезпечують модульність, розширюваність, повторне використання логіки та можливість незалежного розвитку окремих підсистем. Для жанру survival escape це особливо важливо, оскільки ШІ, квестові механіки, фракційна взаємодія, інвентар, торгівля та система прогресу мають працювати узгоджено, але при цьому залишатися відносно ізольованими одна від одної з точки зору реалізації. Схематичне узагальнення основних підходів до організації геймплейної архітектури у грі «Втеча» наведено на рисунку 2.2.

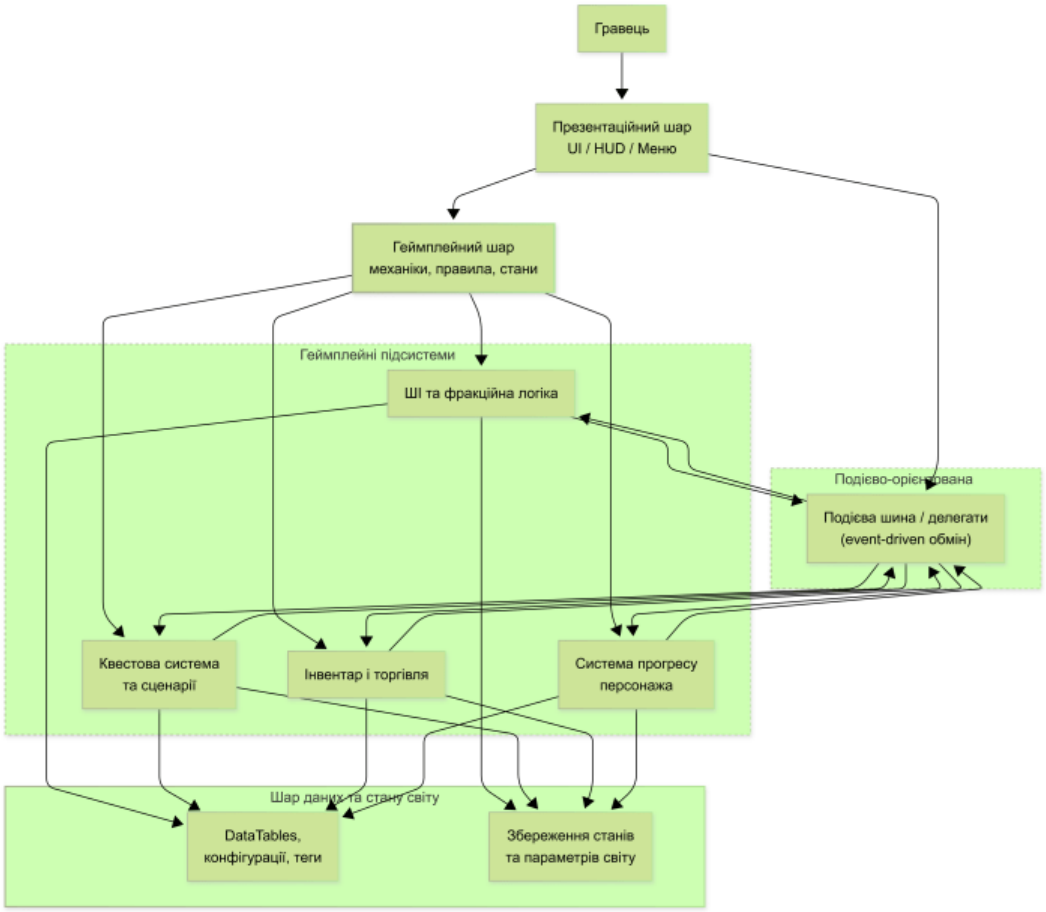


Рисунок 2.2 – Узагальнена архітектура геймплейних систем гри «Втеча» з поєднанням компонентного, подієво-орієнтованого та шарового підходів.

Серед основних архітектурних підходів, що застосовуються під час побудови геймплейних систем, можна виділити:

– об’єктно-орієнтовану модель із використанням спадкування;

– компонентно-орієнтований підхід (Component-Based Architecture);

– модель сутність–компонент–система (Entity-Component-System, ECS);

– подієво-орієнтовану архітектуру (Event-Driven);

– шарову (layered) організацію логіки гри.

Об’єктно-орієнтована модель традиційно застосовується у більшості ігрових рушіїв і передбачає використання класів, спадкування та поліморфізму. Вона є зручною для опису базових ігрових об’єктів (персонажів, предметів, тригерів), однак при надмірно глибокій ієрархії може призводити до жорсткого зв’язування логіки, що ускладнює подальший розвиток ігрової системи. У контексті складних survival escape-систем, де NPC, фракції та елементи оточення мають різні набори поведінкових характеристик, лише класичний ООП-підхід є недостатньо гнучким.

Компонентно-орієнтована архітектура передбачає, що ігровий об’єкт складається з набору незалежних компонентів: рух, здоров’я, інвентар, сенсори, бойова логіка тощо. Такий підхід активно використовується в Unreal Engine через систему Actor та ActorComponent. Він дає змогу: виділяти функціональність у окремі модулі; повторно використовувати компоненти в різних акторах; замінювати або розширювати поведінку без зміни базового класу. У грі «Втеча» це дозволяє, наприклад, будувати різні типи NPC на основі однієї сутності, змінюючи лише набір компонентів (фракційна поведінка, торгові можливості, агресивність) [16].

Модель Entity-Component-System (ECS) орієнтована на розділення даних та логіки. Сутності виступають контейнерами ідентифікаторів, компоненти зберігають дані, а системи обробляють групи компонентів. ECS добре масштабується та забезпечує високу продуктивність, однак її повна інтеграція в готовий рушій потребує додаткового проєктування. У межах Unreal Engine 5

повноцінна ECS-модель може бути реалізована частково, наприклад через DataTables, Gameplay Tags та окремі системи для обробки ШІ-поведінки чи фракційної логіки.

Подієво-орієнтована архітектура базується на ідеї обміну повідомленнями між підсистемами через події. Це дозволяє зменшити зв’язність модулів: система квестів може реагувати на події «гравець помічений охоронцем» чи «змінено репутацію у фракції», не маючи жорсткої залежності від конкретної реалізації ШІ або UI. У Unreal Engine 5 такий підхід реалізується через делегати, події в Blueprint та Broadcaster/Listener-патерни.

Шарова архітектура використовується для логічного розділення гри на рівні: презентаційний (UI, HUD); геймплейний (механіки, правила, стани); дані (налаштування, таблиці, ресурси). Такий підхід забезпечує чіткі межі відповідальності між підсистемами та зменшує кількість прямих залежностей між інтерфейсом, ігровою логікою та сховищами даних. Завдяки цьому зміни у правилах гри або ШІ-логіці можуть виконуватися без переробки UI, а оновлення конфігурацій і контенту не потребує втручання у код механік. У підсумку структура ізолює зміну бізнес-логіки від візуальних елементів і зберігання даних, що підвищує підтримуваність, спрощує тестування та полегшує масштабування ігрової системи.

У грі «Втеча» доцільно поєднати компонентно-орієнтований підхід, подієво-орієнтовану модель та шарову організацію. Базові сутності (гравець, NPC, об’єкти оточення) реалізуються як актори з набором компонентів, що відповідають за рух, здоров’я, ШІ, фракційну логіку, торгівлю та взаємодію. Взаємодія між системами (ШІ, квести, UI, економіка) організовується через події, що зменшує кількість прямих залежностей між модулями. Дані про фракції, предмети, параметри NPC та квестові стани зберігаються у структурованому вигляді (DataTables, конфігураційні об’єкти), що спрощує масштабування та подальше розширення гри. Такий комплексний підхід забезпечує баланс між гнучкістю, прозорістю архітектури та продуктивністю геймплейних систем.

2.3 Підхід до моделювання ШІ у UE5

Моделювання штучного інтелекту в Unreal Engine 5 ґрунтується на поєднанні кількох вбудованих підсистем: Behavior Tree, Blackboard, ШІ Perception, Navigation System та Environment Query System (EQS). Така комбінація дає змогу створювати адаптивну, контекстно залежну поведінку NPC, що є критично важливим для survival escape-гри, де гравець перебуває під постійним тиском з боку противників.

У грі «Втеча» основою ШІ є Behavior Tree, який визначає загальну логіку прийняття рішень NPC. Кожен противник описується через набір задач: патрулювання; переслідування гравця; пошук після втрати візуального контакту; повернення до базової точки; взаємодія з тригерами рівня. Behavior Tree взаємодіє з Blackboard, де зберігаються змінні стану: поточна ціль; остання відома позиція гравця; рівень тривоги; активна фракційна належність. Це дозволяє NPC реагувати на зміну умов без жорсткого кодування сценаріїв.

AI Perception використовується для моделювання органів чуттів NPC. Канали зору та слуху дають можливість реалізувати базові стани: противник бачить гравця; противник чує шум; противник втратив гравця з поля зору. Система навігації (NavMesh) забезпечує побудову маршрутів у межах рівня, що дозволяє ШІ обходити перепони, використовувати коридори, кімнати та зони укриття.

EQS застосовується для вибору позицій, у яких NPC повинен діяти. Наприклад, охоронець може: обрати найкращу точку для огляду; наблизитись до ймовірної позиції гравця; обрати маршрут, що максимізує зону контролю; знайти укриття у разі небезпеки. Завдяки EQS рішення NPC набувають просторового контексту, що робить поведінку менш шаблонною та більш природною.

У грі «Втеча» застосовується гібридний підхід: Behavior Tree керує високорівневими рішеннями, а локальні дії реалізуються через невеликі FSM

(стани анімації, короткі реакції на події, локальні квестові сценарії). Параметри ШІ зберігаються у DataTables, що дає змогу змінювати агресивність, радіус сприйняття, тип патрулювання та фракційну належність без модифікації коду [17]. Підхід забезпечує гнучкість і масштабованість системи.

2.4 Підхід до побудови системи фракційності, торгівлі та економіки

Система фракційності у грі «Втеча» покликана створити динамічні соціальні взаємодії між гравцем та NPC. Кожна фракція має власні цілі, ставлення до інших угруповань та специфічну поведінку стосовно гравця. Це дозволяє формувати альтернативні стилі проходження: співпраця з одними фракціями; конфлікт з іншими; нейтральний шлях із мінімальною взаємодією. Узагальнену схему взаємозв'язку фракційності, торгівлі та економіки в грі «Втеча» наведено на рисунку 2.4.

Рисунок 2.3 – Структурна схема системи фракційності, торгівлі та економіки в грі «Втеча»

Фракційна модель базується на таких елементах: набір фракцій із базовими параметрами (назва, ідеологія, зона контролю, рівень агресії); матриця відносин фракція–фракція; репутація гравця всередині кожної фракції; події, що змінюють репутацію. Дані зберігаються у DataTables, а для відстеження стану використовуються структуровані об’єкти та Gameplay Tags. Поведінка NPC (ворожість, готовність до торгівлі, видача квестів) залежить від репутаційних значень.

Система торгівлі є надбудовою над фракційністю. Кожен торговець має набір товарів; цінові модифікатори, що залежать від фракції та репутації; специфічні умови доступу (наприклад, продаж рідкісних предметів лише союзникам). Логіка торгівлі реалізується у комбінації C++ (для розрахунків, перевірки інвентаря, валідації угод) та Blueprint (для побудови UI-вікон, відображення товарів, вибору опцій гравцем). Економіка є локалізованою, тобто не прагне до повної симуляції ринку, але формує відчуття системності та наслідковості дій гравця.

Економічна модель включає: внутрішню валюту; бартерні операції; рідкісні ресурси, прив’язані до конкретних фракцій; витратні предмети, що впливають на геймплей (ліки, інструменти для зламу, ключі доступу). Усі ці елементи пов’язані з фракційною логікою: виконання завдань підвищує репутацію; висока репутація зменшує ціни; зрада фракції погіршує умови торгівлі або повністю блокує доступ до торговців.

Таким чином, система фракційності, торгівлі та економіки інтегрується з ШІ та квестовими механіками, формуючи нелінійний простір рішень, де дії гравця мають відчутні наслідки як на рівні сюжетного прогресу, так і на рівні доступних ресурсів.

2.5 Моделі побудови сценаріїв та квестів (linear, FSM, branching narrative)

Побудова сценарної структури та квестової системи є ключовим аспектом survival escape-гри, оскільки саме від організації завдань залежить відчуття цілісності, напруги та варіативності проходження. У практиці геймдизайну застосовуються три базові моделі побудови квестів: лінійна (linear); модель на основі автоматів кінцевих станів (FSM); розгалужена модель (branching narrative). Вибір конкретної моделі або їхньої комбінації безпосередньо впливає на рівень контролю над сюжетом, складність реалізації логіки та обсяг зусиль, необхідних для тестування [18]. Узагальнену схему використання цих моделей у грі «Втеча» наведено на рисунку 2.5.

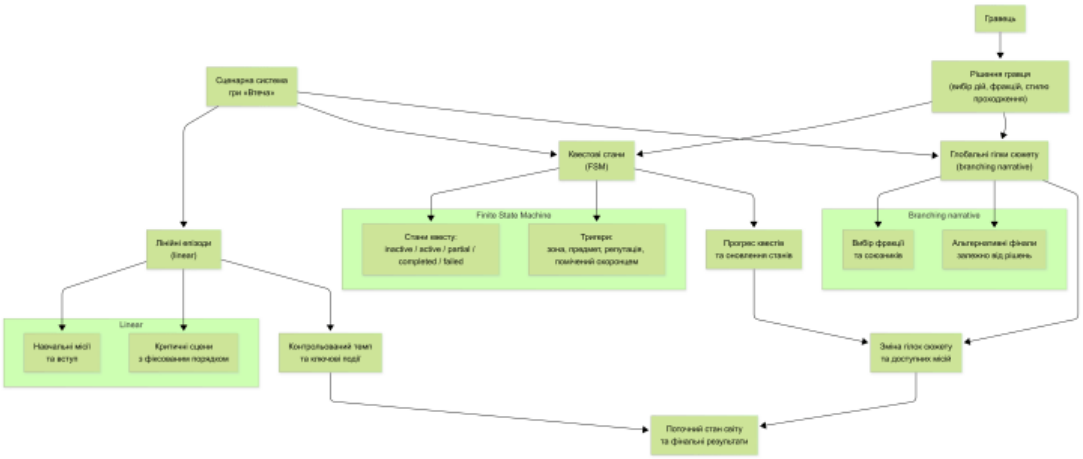


Рисунок 2.4 – Схема комбінованого використання лінійної моделі, FSM та branching narrative в квестовій системі гри «Втеча»

Лінійна модель передбачає фіксовану послідовність завдань, де кожен етап активується лише після завершення попереднього. Вона є простою для реалізації, легко тестується та гарантує контрольований темп оповіді. Однак у контексті гри «Втеча» чисто лінійний підхід обмежував би варіативність та повторюваність проходження, зменшуючи мотивацію до повторного гри та експериментів із різними стилями поведінки.

Модель FSM застосовується на рівні внутрішньої логіки квестів. Кожне завдання має стани: неактивний; активний; виконаний частково; завершений; провалений. Переходи між цими станами відбуваються за умовними тригерами: гравець увійшов у зону; гравець підібрав ключовий предмет; гравець був помічений охоронцем; змінилась репутація у фракції. FSM є зручною для опису локальних сценаріїв, але не підходить як єдина глобальна модель сюжету, оскільки погано масштабується для великої кількості взаємопов'язаних гілок.

Branching narrative забезпечує наявність кількох сценарних гілок, що залежать від дій гравця. У грі «Втеча» цей підхід використовується для формування варіантів: втеча за допомогою конкретної фракції; самостійна втеча без союзників; спроба контролювати баланс між кількома фракціями з різними наслідками. Кожне значуще рішення гравця (підтримка однієї зі сторін, відмова від квесту, агресивна поведінка) змінює доступні шляхи та фінали, формуючи відчуття персоналізованого проходження та підвищуючи емоційне залучення.

На практиці у грі «Втеча» використовується комбінована схема: глобальний сюжет побудований як branching narrative із ключовими розгалуженнями та фіналами; окремі квести та їх внутрішні етапи реалізуються через FSM; лінійні фрагменти застосовуються для критичних сцен, де необхідно забезпечити фіксований розвиток подій (наприклад, вступна місія, перший контакт з основною фракцією, навчальні епізоди). Такий підхід дозволяє поєднати контроль над темпом оповіді з варіативністю вибору та підвищеною реіграбельністю, а також створює прозору структуру квестів, зручну для подальшого розширення, налагодження й командної роботи кількох розробників одночасно. Додатково подібна організація спрощує відстеження наслідків рішень гравця: гілки сюжету, стани FSM та скриптовані лінійні епізоди можуть логуватися, аналізуватися та коригуватися окремо, що полегшує балансування складності, впровадження нових гілок і поступове нарощування контенту без руйнування вже існуючої структури.

2.6 Обґрунтування вибору технологічного стеку

Технологічний стек гри «Втеча» формується виходячи з вимог до жанру, складності ШІ та потреби в модульній архітектурі. Базовою платформою обрано Unreal Engine 5, який забезпечує: високий рівень графічної деталізації; вбудовані інструменти для побудови Behavior Trees, Blackboard та EQS; підтримку змішаного підходу Blueprint + C++; розвинену систему роботи з даними (DataTables, Gameplay Tags).

Мова C++ використовується для реалізації: ядра геймплею; ШІ-логіки; фракційної системи; механізмів економіки; управління збереженнями та конфігурацією. Blueprint застосовується для: побудови UI та HUD; реалізації тригерів рівня; скриптових сцен та кат-сцен; швидкої адаптації геймплейних рішень без перекомпіляції гри. Такий розподіл дозволяє зберегти баланс між продуктивністю та гнучкістю.

Для моделювання даних використовуються: структури (USTRUCT) для опису параметрів NPC, предметів, фракцій; DataTables як централізоване сховище конфігурацій; Gameplay Tags для маркування станів, типів об’єктів, належності до фракцій і сценарних умов. Це спрощує підтримку та розширення системи: додавання нової фракції або типу NPC зводиться до внесення змін у таблиці без модифікації основної логіки.

Додатково до стеку належать: СУВ (Git) для відстеження змін коду й контенту; інтегроване середовище розробки Visual Studio для роботи з C++; редактор Unreal Engine як основний інструмент побудови рівнів, тестування ШІ та налагодження геймплею. Для документування та координації командної роботи застосовуються системи ведення задач і внутрішні Wiki-сховища; збірка проєкту може бути інтегрована з інструментами безперервної інтеграції, які автоматизують компіляцію, запуск тестів і базовий аналіз якості коду перед включенням змін до основної гілки репозиторію.

Узагальнюючи, обраний технологічний стек: забезпечує сумісність із

вимогами жанру survival escape; підтримує створення складної ШІ-логіки без сторонніх рішень; сприяє побудові модульної архітектури; дає можливість подальшого масштабування гри через додавання нових рівнів, фракцій, квестів та DLC без кардинальної перебудови базових систем. Також він забезпечує узгоджений цикл розробки: від проєктування структур даних і коду до візуальної збірки рівнів, профілювання продуктивності й автоматизованого тестування ключових сценаріїв. Наявність єдиного технологічного стеку полегшує командну роботу, спрощує онбординг нових учасників і дозволяє підтримувати єдині підходи до стилю коду, організації контенту та випуску оновлень гри.

2.7 Висновки до розділу

У цьому розділі було розглянуто основні технології, засоби та підходи, що використовуються для розробки комп’ютерної гри «Втеча» на платформі Unreal Engine 5. Порівняння Blueprint та C++ показало доцільність застосування змішаного підходу: візуальне програмування доцільно використовувати для швидкого прототипування, побудови інтерфейсів, квестових тригерів та окремих сценарних подій, тоді як C++ обрано для реалізації продуктивного геймплейного ядра, складної ШІ-логіки, фракційної системи, економічних механізмів та роботи з даними. Такий розподіл обов’язків між інструментами дозволяє поєднати гнучкість розробки із забезпеченням необхідного рівня оптимізації.

Були проаналізовані архітектурні підходи до побудови геймплейних систем, зокрема об’єктно-орієнтована модель, компонентно-орієнтований підхід, подієво-орієнтована архітектура та часткові елементи ECS. На основі цього зроблено висновок про доцільність використання комбінованої моделі, у якій базові сутності реалізуються як актори з компонентами, взаємодія між

підсистемами організовується через події, а дані структуровано в окремих таблицях і конфігураціях. Такий підхід забезпечує модульність, зменшує зв’язаність між модулями та спрощує подальше масштабування гри.

Розглянутий підхід до моделювання ШІ в UE5 показав, що поєднання Behavior Trees, Blackboard, ШІ Perception, NavMesh та EQS дозволяє побудувати адаптивну систему поведінки противників, здатну реагувати на контекст середовища, дії гравця та сценарні умови. Додаткове використання локальних FSM-структур для внутрішніх станів окремих сценаріїв забезпечує необхідну керованість і передбачуваність на мікрорівні без втрати гнучкості на загальному рівні.

Було обґрунтовано підхід до побудови системи фракційності, торгівлі та економіки, яка інтегрується з ШІ і квестовими механіками. Фракційна модель, заснована на репутації та матриці відносин між угрупованнями, надає змогу впливати на доступ до ресурсів, завдань і торгових можливостей, формуючи варіативні стилі проходження та різні наслідки для гравця. Економіка розглядається як інструмент підтримки геймплею, а не як самодостатня симуляція, що відповідає жанровим вимогам survival escape.

Окрему увагу приділено моделям побудови сценаріїв і квестів. Показано, що поєднання лінійних фрагментів, FSM для внутрішньої логіки завдань та розгалуженої моделі (branching narrative) для глобального сюжету є оптимальним рішенням для досягнення балансу між керованістю оповіді та високою реіграбельністю. Така комбінована схема дозволяє враховувати наслідки рішень гравця, підтримувати кілька альтернативних шляхів втечі та формувати різні фінали.

Узагальнюючи, вибір технологічного стеку, архітектурних підходів та моделей ШІ, фракційної взаємодії, економіки й сценарної структури є обґрунтованим і взаємопов’язаним. Обрана комбінація засобів Unreal Engine 5, C++, Blueprint, Behavior Trees, EQS, DataTables та Gameplay Tags забезпечує необхідний рівень гнучкості, продуктивності та масштабованості для реалізації гри «Втеча» як комплексної survival escape-гри з нелінійним проходженням та

адаптивною поведінкою NPC. Отримані результати створюють методологічну й технологічну основу для подальшого проєктування та програмної реалізації систем гри в наступних розділах.

3 ПРОЄКТУВАННЯ КОМП'ЮТЕРНОЇ ГРИ

3.1 Мета, об'єкт, предмет і завдання

Метою дипломної роботи є проєктування та реалізація комп'ютерної гри жанру survival escape «Втеча» на базі рушія Unreal Engine 5 з використанням командного методу розробки, що передбачає розподіл ролей у команді, організацію спільної роботи над грою та застосування сучасних засобів колективної розробки програмного забезпечення. Гра має продемонструвати можливість побудови комплексної геймплейної системи з адаптивним штучним інтелектом, фракційною моделлю взаємодії та нелінійною сценарною структурою, а також показати ефективність командного підходу під час створення ігрових продуктів.

Об'єктом дослідження є процес проєктування, командної розробки та тестування інтерактивного геймплейного середовища у жанрі survival escape на платформі Unreal Engine 5.

Предметом дослідження є методи, моделі та програмні засоби побудови архітектури ігрової логіки, адаптивної поведінки інтелектуальних агентів (NPC), фракційних взаємодій, системи торгівлі та квестових сценаріїв, а також організація командного процесу розробки, включно з розподілом ролей, використанням СУВ та плануванням задач.

Для досягнення поставленої мети у дипломній роботі необхідно розв'язати такі завдання:

– проаналізувати жанр survival escape, сучасні підходи до побудови ШІ та ігрових систем, а також обґрунтувати актуальність створення гри

«Втеча»;

– обрати та обґрунтувати технологічний стек, архітектурні підходи й інструменти для реалізації геймплейних систем на базі Unreal Engine 5;

– розробити структуру гри, визначити основні сутності, їх взаємодію, фракційну модель, сценарні гілки та систему квестів;

– розробити архітектуру та реалізувати систему штучного інтелекту на основі Behavior Trees, Blackboard, AI Perception та EQS з урахуванням вимог жанру survival escape;

– розробити структурута реалізувати систему фракційності, торгівлі й економічних взаємодій, інтегровану з ШІ і квестовою логікою;

– організувати процес командної розробки гри із визначенням ролей (наприклад, технічний лід, розробник геймплейної логіки, розробник інтерфейсу, дизайнер рівнів), розподілом задач та застосуванням СУВ;

– описати обрані підходи до взаємодії в команді (правила внесення змін, узгодження архітектурних рішень, проведення рев'ю коду та контенту);

– створити ігрові рівні, інтерфейс користувача та інтерактивні елементи середовища, що підтримують обрану модель геймплею;

– провести тестування всіх систем

3.2 Організація командної розробки гри «Втеча»

Реалізація гри «Втеча» виконується як командна робота із застосуванням командного методу організації праці, розподілом ролей і відповідальностей, а також спільним плануванням і контролем виконання завдань. Такий підхід наближений до практики комерційної розробки ігор і дає змогу дослідити не лише технічні аспекти створення ігрової системи, а й командну взаємодію; її структуру подано на рисунку 3.1, а основні ролі наведено в таблиці 3.1. У межах розробки виділяються такі основні ролі:

– технічний лід, відповідальний за загальну архітектуру, вибір технологій і ключових підходів до реалізації, підтримку цілісності кодової бази, інтеграцію модулів та узгодження технічних рішень між учасниками команди;

– розробник геймплейної логіки, що зосереджується на механіках взаємодії гравця з оточенням, квестами та об’єктами; розробник ШІ, який працює над Behavior Trees, EQS, налаштуванням AI Perception та параметрами NPC;

– дизайнер рівнів, відповідальний за структуру локацій, тригери, зони ризику й укриття; розробник інтерфейсу користувача, відповідальний за HUD, меню, вікна інвентаря та торгівлі. Залежно від розміру команди ці ролі можуть поєднуватися однією особою, проте розподіл обов’язків зберігається.

Таблиця 3.1 – Ролі та відповідальність учасників команди розробки гри «Втеча»

Роль у команді	**Короткий опис ролі**	**Основні обов’язки**	**Основні сфери відповідальності**
Розробник логіки гри (геймплей / ШІ)	Відповідає за реалізацію геймплейної логіки, роботи ШІ та інтеграцію основних систем гри	Реалізація механік руху й взаємодії гравця; налаштування ШІ (Behavior Trees, EQS, AI Controller); імплементація квестової логіки, фракційної системи, інвентаря; інтеграція систем з UI	Модуль ШІ; квестова система; фракційна модель; інвентар; системи тригерів та ігрових подій
Художник / UI-розробник / 3D-моделер	Відповідає за візуальну складову гри, створення 3D-моделей та інтерфейсу користувача	Створення та доопрацювання 3D-моделей об’єктів і персонажів; впровадження макетів та реалізація екрану інвентаря, HUD, діалогових вікон і меню; підбір стилістики, шрифтів та іконок	3D-моделі об’єктів і персонажів; інтерфейс користувача (HUD, інвентар, квести, меню); візуальна цілісність стилю гри
Левел-дизайнер / 3D-моделер оточення	Відповідає за побудову карт, структуру рівнів та розміщення ігрових елементів у просторі	Проєктування плану рівнів; побудова оточення (кімнати, коридори, зони охорони, точки огляду); розміщення зон патрулювання, точок спавну NPC та предметів; налаштування навігаційної сітки (NavMesh) та тригерів	Структура рівнів; ігрові зони (небезпечні / безпечні / фракційні); маршрути патрулювання; розташування предметів та інтерактивних об’єктів; підтримка коректної роботи ШІ в оточенні

Організація роботи будується на використанні СУВ Git та спільного репозиторію, у якому зберігаються вихідні коди, налаштування гри та частина контенту. Для узгодження змін застосовуються гілки розробки:

– основна гілка з стабільною версією;
– окремі гілки для реалізації нових функцій;
– гілки для експериментальних рішень.

Перед злиттям змін у основну гілку виконуються перевірка та базове тестування, що знижує ризик появи конфліктів або критичних помилок.

Планування завдань здійснюється у вигляді розбиття розробки гри на ітерації (спринти), кожна з яких має конкретну ціль:

– реалізація мінімального геймплею;
– інтеграція ШІ; впровадження фракційної системи;
– додавання системи торгівлі тощо.

Кожне завдання описується з точки зору очікуваного результату та критеріїв готовності. Для комунікації та узгодження рішень використовуються короткі зустрічі (або їхні онлайн-аналоги), на яких обговорюються поточний стан розробки гри, проблеми та наступні кроки.

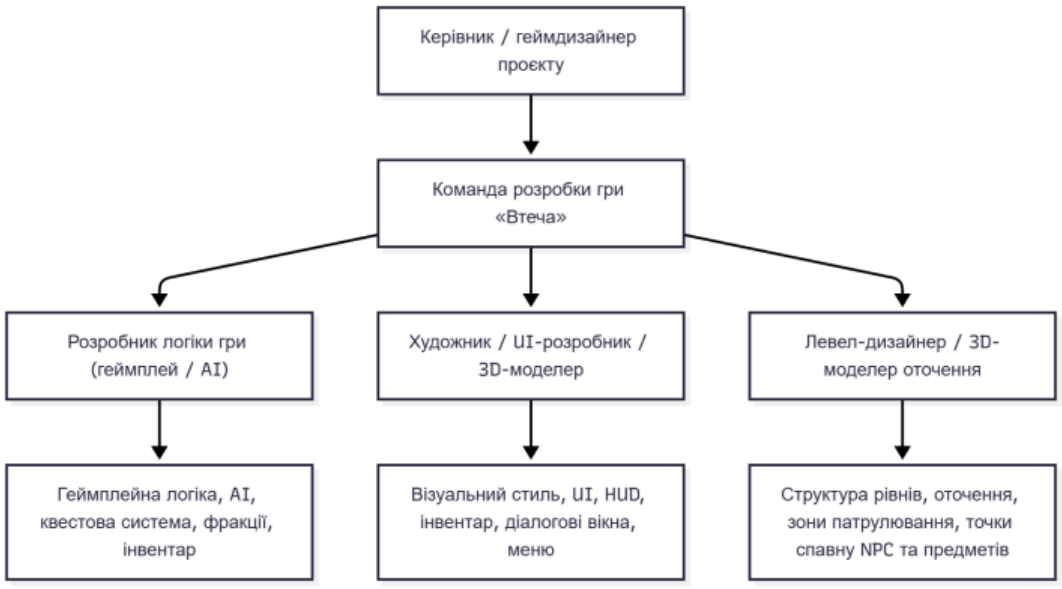


Рисунок 3.1 – Схема командної структури розробки гри «Втеча»

3.3 Функціональні вимоги до гри

Функціональні вимоги визначають, яку саме поведінку має забезпечувати система з точки зору гравця та внутрішньої логіки гри. Для комп'ютерної гри «Втеча» основні функціональні вимоги формуються навколо механік виживання, втечі, взаємодії з NPC, фракціями, предметами та середовищем [19].

До ключових функціональних вимог належать:

– можливість керування персонажем гравця у режимі реального часу, включно з переміщенням, взаємодією з об'єктами, використанням предметів та виконанням дій прихованості;

– наявність системи штучного інтелекту, яка забезпечує поведінку NPC з урахуванням зору, слуху, маршруту патрулювання, переслідування, пошуку гравця після втрати контакту та реакції на події;

– підтримка фракційної моделі, у межах якої NPC належать до різних угруповань, а ставлення фракцій до гравця та одна до одної залежить від вчинків гравця;

– реалізація системи квестів та сценарних подій, що запускаються на основі дій гравця, рівня репутації, перебування у певних зонах чи взаємодії з конкретними NPC;

– наявність інвентаря, який дозволяє зберігати, використовувати, комбінувати й віддавати предмети, що впливають на можливість втечі, доступ до зон, рівень виживання;

– впровадження системи торгівлі з NPC, де ціни, доступність товарів та умови угод залежать від фракційної належності та репутації;

– реалізація користувацького інтерфейсу, що відображає стан персонажа, активні цілі, ключові параметри (здоров'я, ресурси, репутацію), а також дає змогу зручно взаємодіяти з інвентарем і квестами;

– забезпечення можливості досягнення кількох варіантів фіналу

залежно від обраних гравцем стратегій, фракційних союзів і виконаних завдань.

Функціональні вимоги визначають обсяг реалізації, який має бути досягнутий у межах дипломної роботи, і слугують основою для проєктування архітектури та перевірки коректності роботи системи під час тестування.

3.4 Нефункціональні вимоги (продуктивність, оптимізація, масштабованість)

Нефункціональні вимоги характеризують якість роботи системи та її здатність стабільно функціонувати в заданих умовах. Для гри «Втеча» до основних нефункціональних вимог належать вимоги до продуктивності, оптимізації, масштабованості, надійності та зручності використання.

Вимоги до продуктивності включають:

– стабільний показник частоти кадрів на цільовій конфігурації обладнання;

– прийнятний час завантаження рівнів;

– оперативну реакцію системи керування та інтерфейсу на дії гравця.

Система повинна забезпечувати коректну роботу ШІ, квестів і фракційної логіки без значних просідань продуктивності навіть у складних сценах.

Вимоги до оптимізації передбачають:

– раціональне використання ресурсів процесора та пам’яті;

– обмеження кількості одночасно активних ШІ-агентів; застосування механізмів LOD для моделей та спрощення обробки об’єктів, що знаходяться поза полем зору гравця;

– оптимізацію запитів EQS та частоти оновлення поведінкових дерев.

Усі обчислювально складні операції мають виконуватись з урахуванням впливу на загальну продуктивність.

Масштабованість вимагає, щоб архітектура гри дозволяла: додавати нові фракції; розширювати набір NPC; вводити додаткові квестові лінії та рівні; змінювати баланс без кардинальної переробки існуючих модулів. Для цього застосовуються модульний підхід, конфігурація через DataTables та використання тегів і параметрів, а не жорстко закодованих значень.

До інших нефункціональних вимог належать: надійність (відсутність критичних помилок, що заважають проходженню); зручність інтерфейсу (інформативність HUD, логічна структура меню, простота навігації по інвентарю та квестах); супроводжуваність коду (дотримання структурованої архітектури, поділ на модулі, єдиний стиль оформлення). Виконання цих вимог забезпечує не лише комфорт гравця, але й можливість подальшої розробки та підтримки ігрової системи.

3.5 Опис об’єктів системи та UML-діаграми

Опис об’єктів системи є основою подальшого проєктування архітектури гри, оскільки він визначає ключові сутності, їхні властивості та взаємозв’язки. У грі «Втеча» виділяються об’єкти, що належать до кількох груп: гравець; неігрові персонажі (NPC); фракції; предмети; зони й елементи оточення; квестові сутності та тригери. Кожна з цих груп має власні параметри та поведінку, але всі вони взаємодіють у межах єдиного геймплейного простору, формуючи цілісну модель світу. Коректне виділення та формалізація об’єктів дає змогу побудувати модульну архітектуру, налаштувати роботу ШІ, системи фракційності та інвентаря, а також забезпечити узгодженість між логікою гри, базою даних параметрів і візуальним представленням.

Гравець є центральною сутністю системи. Він має параметри: позиція у світі; стан здоров’я; набір характеристик, що впливають на сприйняття та взаємодію (швидкість, рівень шуму, помітність); інвентар із предметами;

поточний статус у фракціях (репутація, союзник, ворог, нейтральний). Дії гравця змінюють стан світу, відносини з NPC та доступні сценарні гілки.

NPC представлені як інтелектуальні агенти, що належать до конкретних фракцій, мають власні ролі (охоронець, торговець, лідер фракції, звичайний житель) та параметри поведінки: рівень агресивності; зону патрулювання; радіус зору та слуху; реакцію на репутацію гравця. Вони взаємодіють між собою та з гравцем на основі фракційної логіки та ШІ-моделей.

Фракції є логічними об’єднаннями NPC з певною ідеологією, цілями та зоною впливу. Для кожної фракції задаються: ставлення до інших фракцій; базове ставлення до гравця; перелік доступних квестів; особливі ресурси та товари. Система фракцій впливає на сценарний прогрес і варіативність проходження, відкриваючи або обмежуючи певні шляхи втечі та варіанти взаємодії.

Предмети поділяються на: корисні (ліки, ресурси, інструменти); ключові (необхідні для відкриття зон або прогресу за квестом); торгові (об’єкти, що мають цінність у торгівлі з NPC). Кожен предмет має тип, вартість, рідкість, можливі способи використання та обмеження. Це дозволяє формалізувати інвентарну систему та пов’язані з нею ігрові механіки.

Зони та елементи оточення включають локації з різним рівнем ризику; зони впливу фракцій; місця появи подій та квестів; тригери, що запускають зміни стану світу. Вони визначають просторову структуру гри, маршрути руху NPC та шляхи втечі гравця, а також впливають на складність проходження й доступність ресурсів.

Формалізований опис об’єктів системи доповнюється їх UML-поданням. Для опису статичної структури використовується клас-діаграма (class diagram), яка відображає основні класи гри: гравець; базовий NPC; конкретні типи NPC; фракція; предмет; квест; інтерфейсні компоненти. Між класами задаються відносини успадкування, асоціації, композиції та агрегації. Узагальнена структура основних сутностей гри «Втеча» зображена на рисунку 3.2, що дозволяє наочно показати архітектуру та зв’язки між об’єктами, а також

виявити потенційні місця розширення й повторного використання компонентів.

Для опису динамічної поведінки застосовуються діаграми послідовності (sequence diagrams), які відображають сценарії взаємодії між об’єктами, зокрема: виявлення гравця NPC; реакцію фракції на зміну репутації; процес торгівлі; виконання квесту. Такі діаграми демонструють порядок виклику методів і обмін повідомленнями у часі, що є важливим для аналізу коректності реалізації сценаріїв.

Алгоритмічні аспекти поведінки системи представляються за допомогою діаграм діяльності (activity diagrams), які описують логіку прийняття рішень ШІ на високому рівні, сценарії втечі з певної зони та послідовність дій під час виконання квесту. Вони дозволяють формалізувати гілкування, паралельні процеси та умови переходів, а також виявити потенційні вузькі місця в логіці.

Станові аспекти окремих сутностей моделюються діаграмами станів (state machine diagrams). Для NPC визначаються стани, пов’язані з поведінкою: патрулювання; підозра; переслідування; пошук; повернення до маршруту. Для квестів задаються стани: неактивний; активний; виконаний; провалений. Використання діаграм станів полегшує розробку й тестування поведінки системи, підвищує прозорість архітектури та спрощує командну розробку й супровід програмного засобу [20].

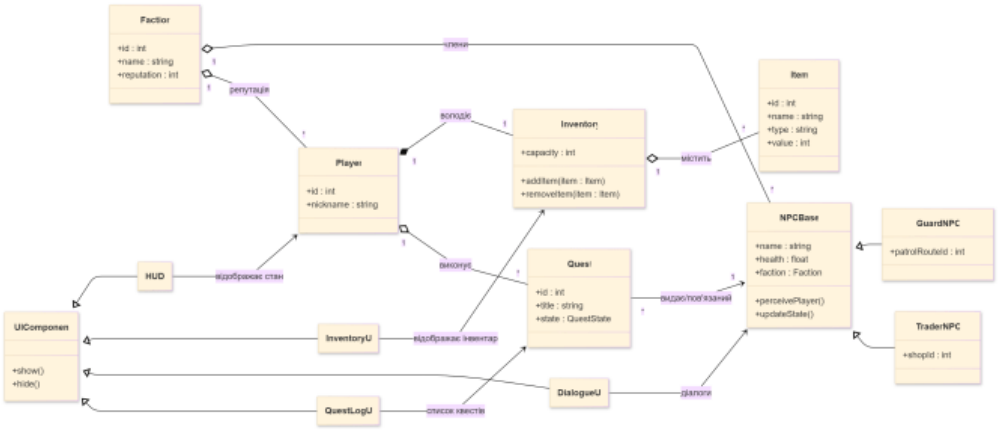


Рисунок 3.2 – UML-діаграма класів основних сутностей

3.6 Архітектура ігрової системи

Архітектура гри «Втеча» будується на модульному підході з поділом на підсистеми: ШІ; фракції; квести; торгівля; інвентар; інтерфейс; система даних. Кожен модуль відповідає за власну область відповідальності, а взаємодія між ними організовується через чітко визначені інтерфейси та події. Узагальнена структура функціональних модулів системи наведена на рисунку 3.3, який демонструє взаємозв'язки між основними підсистемами та їх місце в загальній архітектурі.

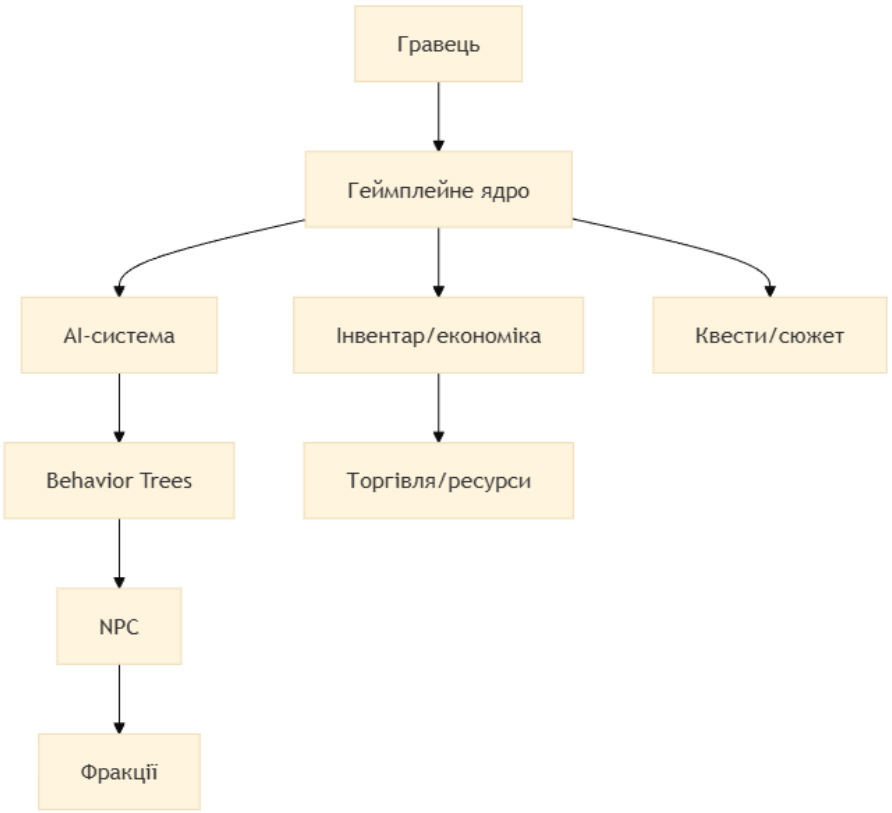


Рисунок 3.3 – Функціональні модулі системи

Базовим елементом архітектури є актори Unreal Engine, до яких підключаються компоненти, що реалізують конкретну поведінку. Наприклад, персонаж гравця має компоненти руху, здоров'я, інвентаря, взаємодії з

об’єктами; NPC містить компоненти ШІ, фракційної належності, сенсорів. Дані про параметри об’єктів зберігаються у DataTables, а логіка прийняття рішень реалізується через Behavior Trees та додаткові структури, що забезпечує гнучку конфігурацію поведінки без зміни базового коду.

На логічному рівні архітектура поділяється на три шари: шар представлення (UI, HUD); геймплейний шар (механіки, ШІ, квести, фракції, торгівля, інвентар); шар даних (таблиці, конфігурації, збереження). Взаємодія між шарами відбувається у напрямку від даних до геймплею та від геймплею до представлення. Такий підхід дозволяє змінювати спосіб відображення інформації без втручання у бізнес-логіку, забезпечує чітке розділення відповідальностей між компонентами та спрощує подальшу підтримку, розширення й тестування окремих частин системи.

Модуль ШІ відповідає за реалізацію поведінки NPC. Він включає Behavior Trees, Blackboard із змінними стану, AI Perception для моделювання чуттів, EQS для просторових рішень і навігаційну систему. На основі цих компонентів ШІ-противники здатні патрулювати, виявляти гравця, переслідувати його, здійснювати пошук після втрати контакту, реагувати на події та зміну стану світу. Узагальнена схема станів і переходів ШІ-противника наведена на діаграмі станів, зображеній на рисунку 3.4.

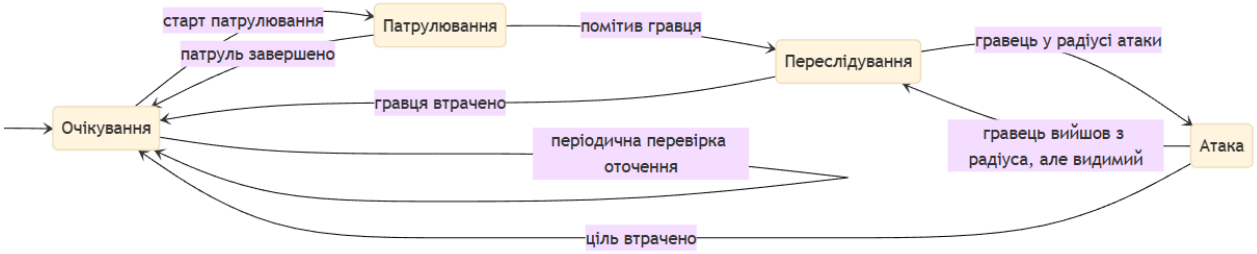


Рисунок 3.4 – Діаграма станів ШІ-противника

Конфігурація ШІ задається через параметри у DataTables: радіуси сприйняття; типи реакцій; сценарні особливості окремих NPC. Це дає змогу

створювати різні профілі поведінки (агресивні охоронці, нейтральні торговці, ворожі патрулі) без модифікації базової архітектури, а також швидко коригувати баланс складності.

Модуль фракцій реалізує логіку відносин між угрупованнями та між фракціями і гравцем. У ньому зберігаються список фракцій, матриця відносин фракція–фракція, репутаційні показники гравця та правила зміни репутації. Модуль виступає джерелом інформації для ШІ й квестової системи, визначаючи, чи буде NPC агресивним, чи відкриється доступ до торгівлі, які квести стануть доступними в поточному стані гри. Уся фракційна інформація зберігається у структурованому вигляді, що дозволяє легко додавати нові фракції, змінювати баланс взаємин і розширювати сценарні гілки без перебудови всієї системи.

Модуль квестів відповідає за створення, зберігання та обробку завдань. Кожен квест має ідентифікатор, опис, поточний стан, умови активації, вимоги до виконання та наслідки. Модуль реагує на події, що надходять з інших підсистем: дії гравця, зміну репутації, досягнення певних зон, взаємодію з NPC. Внутрішній життєвий цикл квесту реалізується у вигляді кінцевого автомата (FSM), який визначає переходи між станами «неактивний», «активний», «виконаний», «провалений». Зовнішня логіка побудована на подієвій моделі, що забезпечує гнучку реакцію на зміни у світі гри та дозволяє комбінувати квестові гілки з фракційними й економічними механіками.

Модуль торгівлі підтримує обмін предметами між гравцем і NPC. Він включає логіку формування списку товарів, розрахунок цін з урахуванням фракційної належності та репутації, перевірку наявності ресурсів у гравця, застосування результатів угоди до інвентаря. Цінові параметри та асортимент можуть змінюватися залежно від поточного стану світу, виконаних квестів та рівня загальної небезпеки, що підсилює взаємодію між економічною моделлю й сценарною логікою. Модуль тісно інтегрується з фракційною системою та модулем інвентаря: саме він відповідає за коректне оновлення даних про ресурси, зміну доступності товарів і фіксацію наслідків транзакцій.

Торгові операції здійснюються через інтерфейс користувача, але всі розрахунки виконуються на рівні логіки модуля торгівлі. Такий підхід забезпечує узгодженість даних, можливість подальшого розширення економічних механік (знижки, унікальні пропозиції, динамічні ціни) і спрощує налагодження й тестування поведінки торгових NPC у різних ігрових сценаріях.

Модуль UI відповідає за відображення інформації гравцю та організацію взаємодії з системою. До нього належать HUD, меню паузи, вікно інвентаря, вікно квестів, торгові інтерфейси, діалогові вікна. Модуль отримує дані з геймплейних підсистем і відображає їх у зручній формі, забезпечуючи гравцеві доступ до стану персонажа, активних завдань, ресурсів і можливих дій.

Використовується підхід, за якого інтерфейс не містить логіки прийняття рішень, а лише реагує на зміни стану та передає команди геймплейним системам. Це спрощує підтримку та модифікацію інтерфейсу, дозволяє змінювати візуальне оформлення й структурну організацію екранів без ризику порушити роботу базових механік, а також полегшує адаптацію гри до різних роздільних здатностей і сценаріїв використання.

3.7 Проєктування бази даних для зберігання параметрів (DataTables)

Для зберігання конфігураційних даних у грі «Втеча» використовуються DataTables Unreal Engine. Вони дозволяють описувати параметри у табличному вигляді, що зручно для редагування й масштабування. Основні таблиці включають: опис NPC; опис предметів; опис фракцій; конфігурацію квестів; параметри ШІ-поведінки. Узагальнена структура основних таблиць конфігураційних даних наведена на рисунку 3.5.

У таблиці NPC зберігаються: тип персонажа; фракційна належність; базові характеристики; посилання на профіль ШІ. Таблиця предметів містить: ідентифікатор; назву; тип; рідкість; вартість; ефект при використанні. Таблиця

фракцій включає: назву; опис; ставлення до інших фракцій; базове ставлення до гравця. Квестова таблиця описує: умови активації; вимоги до виконання; нагороди; зміни репутації.

Застосування DataTables дозволяє розділити код і дані: логіка модулів працює з абстрактними параметрами, тоді як конкретні значення можуть змінюватися без перекомпіляції гри. Це особливо важливо при балансуванні гри, додаванні нового контенту та адаптації системи під інші сценарії.

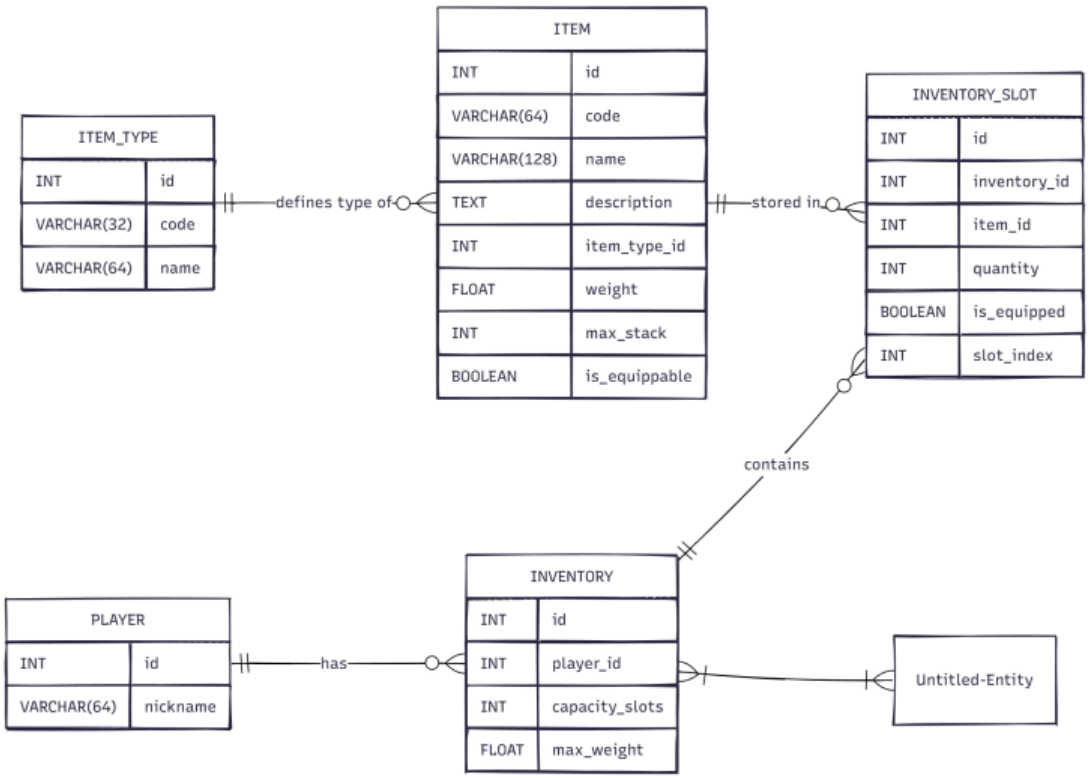


Рисунок 3.5 – Діаграма станів ШІ-противника

3.8 Проєктування мережевої реплікації

Мережева реплікація у грі «Втеча» розглядається як перспектива розширення гри до багатокористувацького режиму. На етапі проєктування передбачається можливість розподілу частини логіки між сервером і клієнтами,

а також визначаються об’єкти, які потенційно потребуватимуть синхронізації. До таких об’єктів належать: позиція та стан гравця; ключові дії (взаємодія з предметами, активація тригерів); стани NPC; результати квестів; зміни у фракційній системі. Архітектура гри будується з урахуванням того, що основна логіка може виконуватися на стороні сервера, а клієнти отримуватимуть оновлені стани через механізми реплікації Unreal Engine. Навіть якщо у базовій версії гри «Втеча» багатокористувацький режим не реалізується повністю, закладення принципів реплікованої архітектури на етапі проєктування спрощує подальшу еволюцію гри у напрямку кооперативного чи змагального режиму без радикальної переробки коду. Це дає змогу заздалегідь визначити критичні зони навантаження, продумати політику синхронізації даних та мінімізувати ризики розсинхрону ігрових сесій у мережевому середовищі з затримкою.

3.9 Висновки до розділу

У цьому розділі було виконано проєктування комп’ютерної гри «Втеча» з урахуванням як технічних, так і організаційних аспектів. Визначено мету, об’єкт, предмет і завдання дипломної роботи, а також сформульовано підхід до організації командної розробки із розподілом ролей, використанням СУВ та ітеративним плануванням робіт.

Сформовано функціональні та нефункціональні вимоги до гри, що задають рамки для реалізації геймплейних систем, ШІ, фракційної моделі, економіки та інтерфейсу. Описано основні об’єкти системи, включно з гравцем, NPC, фракціями, предметами та зонами, а також запропоновано використання UML-діаграм для формалізації структури та поведінки системи.

Запропонована архітектура гри базується на модульному та шаровому підходах, що дозволяє розділити відповідальність між підсистемами ШІ, фракцій, квестів, торгівлі та UI. Проєктування бази даних на основі DataTables

забезпечує гнучке управління параметрами та спрощує подальше розширення контенту. Попередній аналіз можливостей мережевої реплікації створює основу для потенційного переходу до багатокористувацького режиму.

Таким чином, розділ 3 формує цілісну модель майбутньої гри «Втеча»

4 РЕАЛІЗАЦІЯ ГРИ «ВТЕЧА»

Реалізація комп’ютерної гри «Втеча» ґрунтується на попередньо спроєктованій архітектурі та включає впровадження модулів штучного інтелекту, фракційної системи, квестової логіки, інтерфейсу користувача, інвентаря, інтерактивного середовища та засобів оптимізації. На даному етапі теоретичні моделі перетворюються на конкретні реалізації в Unreal Engine 5 із використанням C++ та Blueprint, що дозволяє створити цілісний ігровий продукт, готовий до тестування та подальшого дослідження. Особлива увага приділяється інтеграції окремих підсистем між собою, налаштуванню обміну даними й подій, а також забезпеченню відтворюваності результатів експериментів, що є необхідною умовою для подальшого аналізу ефективності запропонованих технічних рішень.

4.1 Структура програмного коду гри в Unreal Engine 5

Структура програмного коду гри «Втеча» в Unreal Engine 5 побудована за модульним принципом із чітким розподілом відповідальностей між C++ класами, компонентами та Blueprint-ресурсами. Ядро геймплею, системи штучного інтелекту, фракційні та квестові підсистеми реалізовано на рівні C++, тоді як інтерфейс користувача, значна частина тригерів рівня та окремі варіанти

поведінки NPC конфігуруються у вигляді Blueprint-структур. Такий підхід поєднує продуктивність нативного коду з гнучкістю візуального налаштування і спрощує подальше розширення гри та командну роботу.

Базу геймплейних об’єктів становить набір C++ класів персонажів і контролерів. Клас AEscapeCharacter відповідає за керування гравцем: рух, обробку вводу, взаємодію з інвентарем, перевірку можливості взаємодії з об’єктами середовища та реакцію на події ШІ й квестової системи. Для неігрових персонажів використовується ієрархія класів на основі ACharacter, зокрема базовий клас ABaseNPCCharacter і спеціалізовані похідні, які відрізняються набором параметрів, анімацій та фракційної належності. Логіка прийняття рішень NPC зосереджена в контролерах типу AGuardAIController та інших класах, похідних від AIController, що інтегруються з Behavior Tree, Blackboard та системою сприйняття.

Функціональність, спільна для різних типів акторів, винесена у компоненти. Компонент UHealthComponent інкапсулює логіку очок здоров’я, отримання шкоди та подій смерті; UInventoryComponent відповідає за зберігання предметів, перевірку наявності ресурсів та взаємодію з системою торгівлі; UInteractionComponent реалізує перевірку досяжності об’єктів, формування підказок і виклик дій взаємодії. За обробку фракційної належності та репутації персонажів відповідають спеціалізовані компоненти, які зберігають ідентифікатор фракції та кешують значення репутації для швидкого доступу з боку ШІ та квестових підсистем.

На рівні загальної логіки гри використовується набір керуючих класів рушія. Клас AEscapeGameModeBase ініціалізує основні підсистеми, визначає типи контролера гравця, HUD і базових класів персонажів, а також керує переходами між рівнями. Клас AEscapePlayerController обробляє ввід користувача, перемикання режимів керування та взаємодію з інтерфейсом. Клас AEscapeHUD підключає ключові елементи UI та здійснює зв’язок між геймплейними даними і екранними відображеннями. Для глобальних сервісів застосовуються підсистеми на основі UGameInstanceSubsystem та

UWorldSubsystem, зокрема UFactionManagerSubsystem для роботи з фракціями, UQuestManagerSubsystem для керування квестами та USaveManagerSubsystem для збереження й відновлення стану гри.

Значна частина конфігураційних даних організована за принципом data-driven підходу. Для опису предметів використовується структура FItemData, що зберігається в DataTable ItemsTable; для фракцій – структура FFactionData в таблиці FactionsTable; для квестів – структура FQuestData у відповідній таблиці завдань. C++ класи працюють із цими таблицями через уніфіковані інтерфейси, що забезпечує відокремлення логіки від контенту та дає змогу змінювати баланс, описи й параметри без модифікації коду.

Blueprint-ресурси використовуються як надбудова над C++ класами. Для основних C++ акторів створюються Blueprint-нащадки, у яких налаштовуються візуальні компоненти, анімації, параметри матеріалів, звукові ефекти та конкретні значення змінних. Behavior Trees і EQS-запити зберігаються як окремі Blueprint-ресурси, пов’язані з контролерами ШІ, тоді як UI реалізується через Widget Blueprint-и, що отримують дані від відповідних C++ підсистем. Узаємодія між нативним кодом і Blueprint-логікою здійснюється через події, делегати та виклики функцій, що дозволяє гнучко налаштовувати сценарії без порушення цілісності архітектури.

Узагальнену структуру основних класів, компонентів і підсистем гри «Втеча» в Unreal Engine 5 подано на рисунку 4.1, де відображено ключові зв’язки між гравцем, NPC, менеджерами фракцій і квестів, системою інвентаря, інтерфейсом користувача та сервісами збереження. Схема демонструє розподіл відповідальностей між підсистемами, основні точки взаємодії та напрямки обміну даними, зокрема через подієві механізми і спільні структури стану. Така організація забезпечує узгоджену роботу всіх компонентів у межах єдиної архітектури та спрощує подальше розширення функціональності без порушення базової логіки гри.

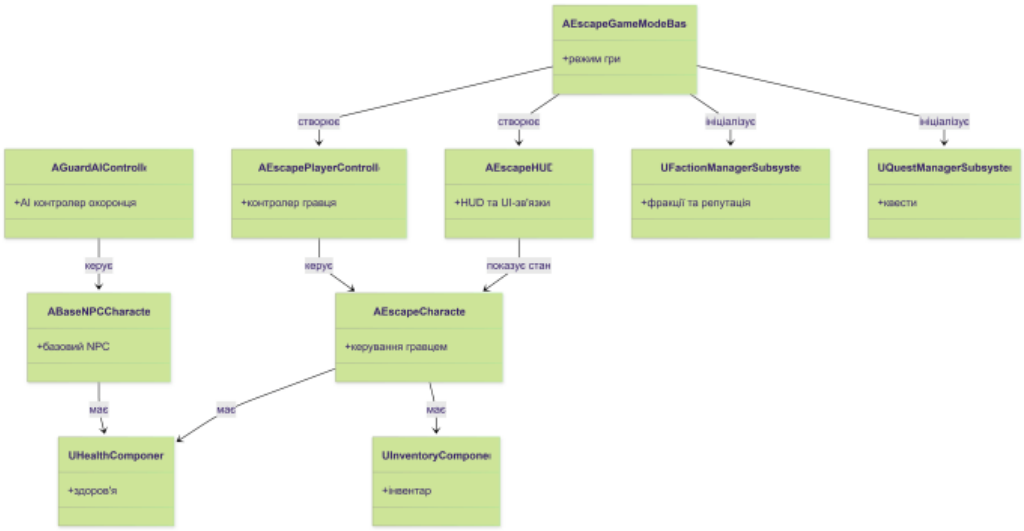


Рисунок 4.1 – Узагальнена структура основних класів гри «Втеча» в Unreal Engine 5

4.2 Реалізація ШІ-системи на основі Behavior Trees та EQS

ШІ-система у грі «Втеча» реалізована на базі стандартних засобів Unreal Engine 5: Behavior Trees, Blackboard, AIController, AI Perception та Environment Query System (EQS). Для кожного типу NPC створюється окремий клас контролера, похідний від AIController, у якому задається прив’язка до відповідного Behavior Tree, відповідної Blackboard-схеми та профілю сприйняття. На етапі ініціалізації контролера виконується налаштування сенсорної системи (зір, слух, реакція на поштовхи/удари), визначаються радіуси виявлення, кути огляду, затримки оновлення стану, а також ключові параметри агресивності й обережності, що пізніше використовуються в Behavior Tree для прийняття рішень.

Логіка поведінки організована за ієрархічним принципом: на верхньому рівні Behavior Tree описує основні стани NPC (патрулювання, переслідування, пошук гравця, повернення на маршрут, тривога), а на нижчих рівнях – конкретні тактичні дії (зміна маршруту обходу, перевірка джерела шуму, вибір

укриття, виклик підкріплення тощо). Перемикання між гілками дерева здійснюється на основі значень ключів Blackboard (наприклад, наявність цілі, остання відома позиція гравця, рівень загрози, стан тривоги), які оновлюються завдяки подіям AI Perception та результатам запитів EQS. Такий підхід забезпечує керовану, але водночас гнучку поведінку NPC, дозволяючи комбінувати стандартні вузли UE5 із кастомними задачами, написаними на C++. Схема Behavior Tree для NPC-охоронця наведена на рисунку 4.1.

Environment Query System використовується для просторового аналізу ігрового оточення: за допомогою EQS NPC може автоматично підбирати оптимальні точки для патрулювання, перевірки підозрілих зон або пошуку укриття, враховуючи видимість, відстань до гравця, наявність перешкод та інші параметри. У результаті поведінка охоронців не обмежується жорстко заданими маршрутами, а адаптується до поточної ситуації в грі, підвищуючи правдоподібність ігрових сценаріїв. Така структура спрощує повторне використання вузлів поведінки між різними типами противників, уніфікацію шаблонів реакцій для різних ролей (патрульний, альфа-охоронець, сторож точки інтересу) та централізоване керування параметрами чутливості й агресивності через конфігураційні дані, не змінюючи базову архітектуру ШІ.

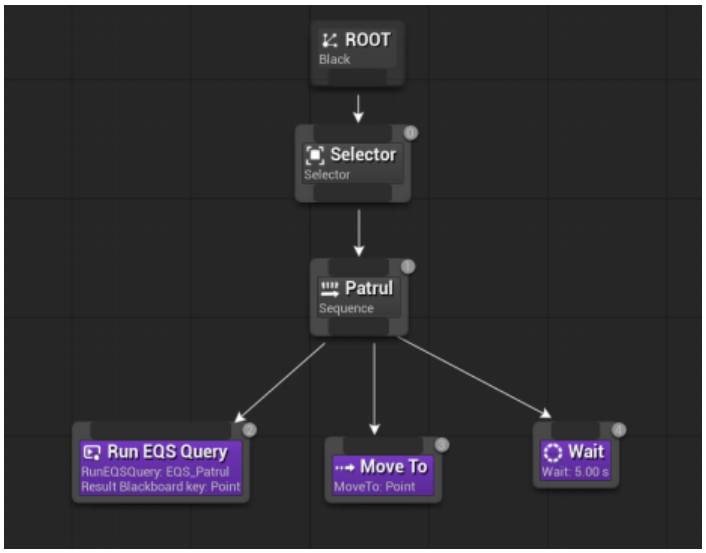


Рисунок 4.2 – Схема Behavior Tree для NPC-охоронця

Основні елементи реалізації:

– створення Behavior Tree для патрулювальних NPC, NPC-охоронців та спеціальних персонажів, які реагують на фракційні події;

– налаштування Blackboard зі змінними: поточна ціль; остання відома позиція гравця; рівень тривоги; активний режим поведінки; стан фракційних відносин;

– впровадження сервісів Behavior Tree, які періодично оновлюють інформацію про гравця, перевіряють видимість, наявність шумів, зміну зон ризику;

– реалізація завдань (tasks), які відповідають за конкретні дії NPC: рух до цілі; початок переслідування; пошук гравця в області; повернення до маршруту; взаємодія з тригерами рівня.

AI Perception використовується для моделювання зору та слуху, формуючи базовий механізм отримання NPC інформації про події в ігровому світі. Для кожного NPC налаштовується сенсор зору з кутом огляду, максимальною дистанцією та параметрами втрати цілі, а також канал слуху з радіусом виявлення шуму і порогами чутливості для різних типів звуків. Це дозволяє розрізняти ситуації, коли гравець потрапляє в поле зору, коли його присутність лише підозрюється через шум, або коли контакт втрачається.

Події сприйняття інтегруються з Behavior Tree через оновлення змінних у Blackboard, що забезпечує реактивну зміну поведінки без ручного керування переходами. Зокрема, оновлення ключів Blackboard (позиція цілі, останній відомий напрямок, рівень загрози) дає змогу NPC своєчасно перемикатися між станами патрулювання, підозри та переслідування, а також ініціювати додаткові перевірки середовища або виклик підкріплення залежно від контексту.

EQS застосовується для просторових рішень. Створюються запити, які генерують множину потенційних позицій: точки для пошуку гравця після втрати контакту; зони, з яких NPC може перекривати основні маршрути гравця; позиції для тимчасових укриттів. Кожній точці призначаються критерії оцінки:

відстань до останньої відомої позиції гравця; наявність прямої видимості; рівень відкритості зони; близькість до зон інтересу. Поведінка системи додатково налаштовується через вагові коефіцієнти критеріїв, що дає змогу створювати різні тактичні профілі NPC (агресивний, обережний, патрульний) та адаптувати їх до конкретних сценаріїв рівня. Behavior Tree викликає EQS-запит у відповідних вузлах, а обрана позиція записується в Blackboard як ціль для переміщення; приклад просторового запиту EQS із відображенням вибраних точок наведено на рисунку 4.2, що дає змогу візуально проаналізувати якість вибору позицій, виявити проблемні зони рівня та скоригувати параметри оцінювання у процесі налагодження й оптимізації геймплею.

Частина ШІ-логіки реалізується у C++ (AIController, базові сервіси, складні обчислення), тоді як Behavior Trees та окремі завдання створюються в Blueprint, що спрощує налагодження та дає змогу швидко змінювати поведінку без перекомпіляції всієї гри, а також забезпечує зручну візуалізацію рішень для аналізу, командної роботи та подальшого доопрацювання сценаріїв. Такий гібридний підхід поєднує високу продуктивність нативного коду з гнучкістю візуального сценарного проєктування, що є особливо важливим для survival escape-ігор, де необхідно часто ітеративно змінювати логіку реакцій NPC на дії гравця.

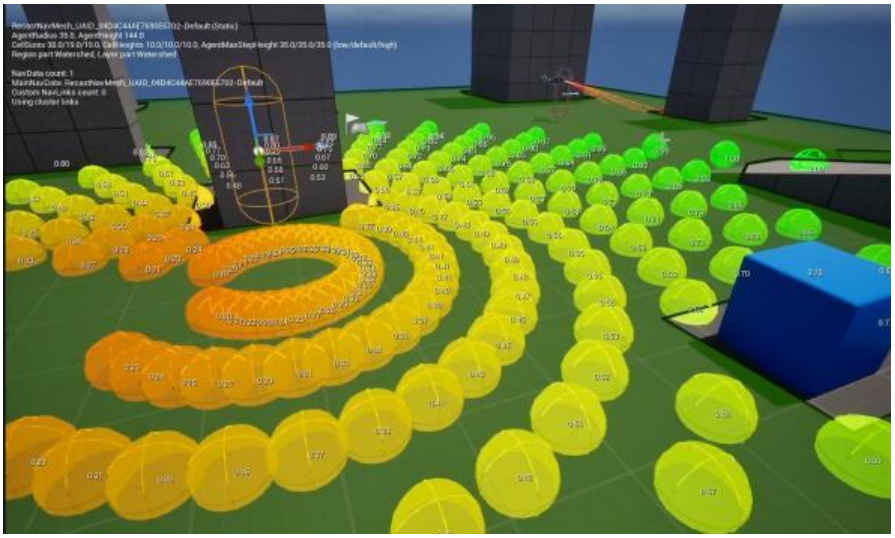


Рисунок 4.3 – Приклад запиту EQS із відображенням вибраних точок

4.3 Реалізація системи фракцій та репутації

Система фракцій реалізована як окремий модуль, що працює зі структурованими даними та надає інтерфейси для інших підсистем. На рівні даних використовується DataTable, у якій описані фракції, їхні базові характеристики та матриця відносин. Для кожної фракції задаються: ідентифікатор; назва; короткий опис; базове ставлення до гравця; параметри агресивності; ключові зони впливу. Графічне відображення поточного рівня репутації гравця у різних фракціях наведено на рисунку 4.3.

Репутація гравця у кожній фракції зберігається у спеціальному об’єкті, що асоційований із профілем гравця. Під час виконання квестів, взаємодії з NPC або участі у конфліктах значення репутації змінюються за заздалегідь визначеними правилами. Наприклад:

– виконання завдання для фракції збільшує репутацію в цій фракції;

– допомога ворогам фракції знижує репутацію;

– агресивні дії проти NPC певного угруповання зменшують репутацію аж до стану відкритої ворожнечі.

Для доступу до фракційних даних створено центральний менеджер фракцій, реалізований у C++. Він надає методи: отримати поточне ставлення фракції до гравця; змінити репутацію; обчислити наслідки події для кількох фракцій одночасно; надати інформацію для ШІ та квестової системи. Додатково менеджер підтримує єдину точку конфігурації порогових значень репутації, що дозволяє швидко змінювати баланс без переписування ігрової логіки, а також збирає статистику взаємодій для подальшого аналізу та тонкого налаштування складності.

NPC отримують інформацію про фракцію та репутацію через компоненти, прив’язані до персонажів. Під час ухвалення рішень Behavior Tree звертається до фракційного модуля: наприклад, вирішити, чи атакувати гравця одразу, попередити, проігнорувати або запропонувати взаємодію. Квестова

система використовує репутацію для розблокування або блокування окремих ліній завдань. Таким чином, фракційна модель безпосередньо впливає на геймплей та варіативність проходження, роблячи реакції світу на дії гравця більш послідовними, передбачуваними з точки зору внутрішньої логіки та водночас менш шаблонними для користувача.

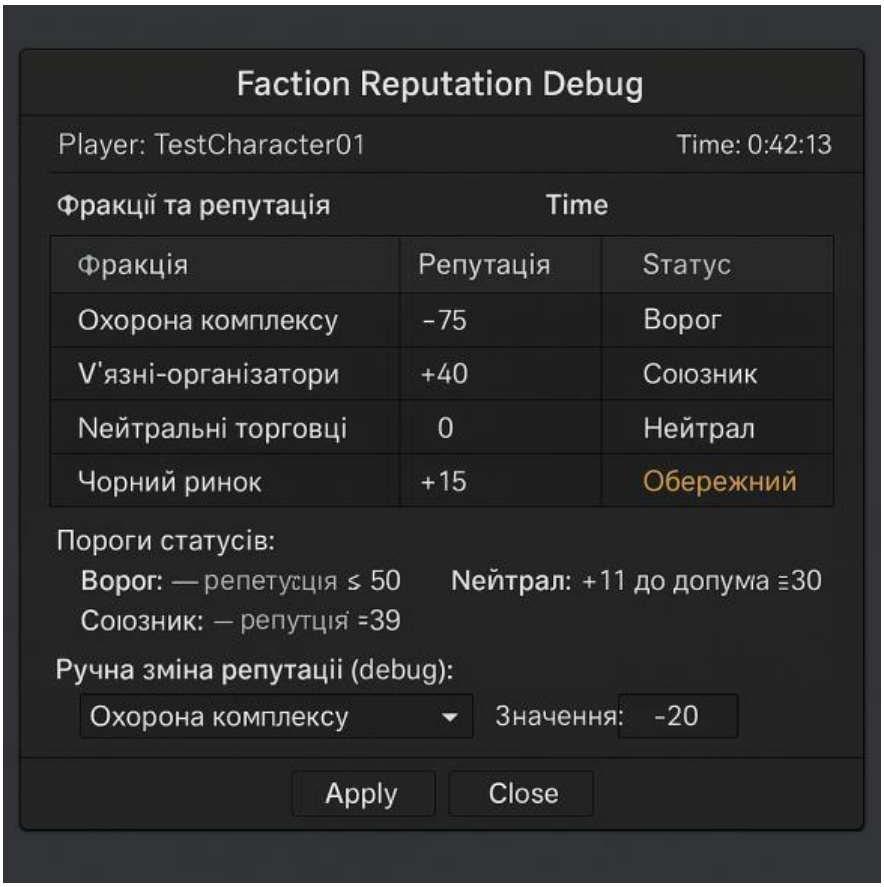


Рисунок 4.4 – Приклад відображення репутації гравця у фракціях

4.4 Реалізація системи квестів та сценарних переходів

Система квестів реалізована як модуль, що керує життєвим циклом завдань, зберігає їхній стан та забезпечує взаємодію з іншими підсистемами. Кожен квест описується структурою даних, у якій вказано: ідентифікатор; назва; опис; фракція, від імені якої видається завдання; умови активації; список

цілей; можливі наслідки; зміни репутації; вплив на сюжет.

Стани квестів реалізовані у вигляді кінцевих автоматів: неактивний; доступний; активний; виконаний; провалений. Переходи між станами ініціюються подіями: гравець поговорив з NPC; увійшов до певної зони; зібрав потрібний предмет; досягнув певного рівня репутації. Для доставки подій використовується подієво-орієнтована модель: модуль квестів підписується на події від ШІ, фракцій, інвентаря та системи тригерів рівня.

Сценарні переходи реалізуються через систему тагів і прапорців стану світу. Ключові події (виконання головного квесту фракції, зрада угруповання, успішна втеча з певної зони) встановлюють сценарні прапорці, які впливають на доступність подальших квестів і фіналів. Наприклад:

– виконання серії квестів для однієї фракції може заблокувати доступ до альтернативної гілки;

– провал критичного завдання переводить гру на інший сценарний шлях;

– успішна співпраця з кількома фракціями відкриває додатковий варіант завершення.

Квестовий модуль інтегрується з UI: активні завдання відображаються у журналі, цілі позначаються на карті або у вигляді підказок, а ключові рішення супроводжуються діалоговими вікнами.

4.5 Реалізація UI

Інтерфейс користувача реалізований за допомогою системи UMG (Unreal Motion Graphics). Основні елементи UI: HUD; головне меню; меню паузи; екран інвентаря; екран квестів; інтерфейс торгівлі; діалогові вікна. Кожен екран створюється як окремий Widget Blueprint, який отримує дані від геймплейних підсистем.

HUD відображає основну інформацію про стан гравця: рівень здоров’я; наявність ключових ресурсів; поточні цілі; індикатори виявлення (наприклад, рівень підозри ворогів). Для цього використовується прив’язка змінних до прогрес-барів, текстових полів та піктограм. Оновлення HUD відбувається через події, що надходять від компонента стану гравця.

Екран інвентаря реалізує відображення списку предметів, їх характеристик та можливих дій (використати; передати; викинути). Логіка взаємодії з предметами реалізована у C++, тоді як відображення й обробка кліків виконується у Blueprint. Подібним чином побудовано екран квестів, де відображаються активні та завершені завдання, їхній опис та статус.

Інтерфейс торгівлі включає два списки: предмети торговця; предмети гравця. Під час вибору товару відображаються ціна, фракційні знижки та умови покупок. Проведення операції викликає функції модуля торгівлі, який оновлює інвентар і валютні ресурси.

Діалогові вікна використовуються для відображення важливих сюжетних подій, вибору реплік у діалогах з NPC та підтвердження рішень, що впливають на репутацію та сценарій. Усі UI-елементи створені таким чином, щоб не дублювати логіку, а лише відображати поточний стан систем і передавати команди до відповідних модулів.

4.6 Реалізація інвентаря, взаємодії з предметами

Система інвентаря реалізована як компонент, що підключається до гравця та, за потреби, до окремих NPC. У компоненті інвентаря зберігається список структур, кожна з яких описує предмет: ідентифікатор; кількість; тип; додаткові параметри. Опис предметів міститься у DataTable, з якої під час завантаження підтягуються назви, описи, іконки, значення характеристик та ефектів; інтерфейс роботи з інвентарем гравця наведений на рисунку 4.4, а приклад

відображення окремого предмета «Вибухівка» зображений на рисунку 4.5.

Взаємодія з предметами відбувається у кілька етапів:

– визначення можливості взаємодії з об’єктом у світі (через трасування променем або зону взаємодії);

– відображення підказки про доступну дію;

– виконання дії (підняти предмет; відчинити двері ключем; активувати механізм);

– оновлення інвентаря та стану світу відповідно до результатів.

Частина предметів є ключовими, тобто впливають на можливість подальшого прогресу: ключі від дверей; картки доступу; унікальні предмети для квестів. Для таких об’єктів у квестовій системі задаються залежності, що враховують наявність предмета в інвентарі та можуть блокувати або розблоковувати окремі етапи завдання, альтернативні шляхи проходження й варіанти фіналу.

Система інвентаря підтримує: стековані предмети; обмеження за кількістю; перевірку доступності ресурсу перед виконанням дії; інтеграцію з торгівлею (списання та надходження предметів, обчислення вартості), а також забезпечує узгодженість стану світу з діями гравця за рахунок централізованого зберігання даних про ресурси.

Рисунок 4.5 – Інтерфейс інвентаря гравця

Рисунок 4.6 – Приклад предмету «Вибухівка»

4.7 Побудова рівнів та інтерактивного середовища

Рівні у грі «Втеча» створені як комбінація внутрішніх приміщень, коридорів, зон охорони та відносно безпечних ділянок. Для кожного рівня визначаються: основні маршрути гравця; зони патрулювання NPC; місця можливих засідок; локації з високою концентрацією ресурсів; зони впливу фракцій. Узагальнена схема просторової структури рівня та розміщення ключових зон наведена на ескізі карти рівня, зображеному на рисунку 4.6.

Інтерактивні елементи середовища включають: двері, що відчиняються ключами або через панелі доступу; камери спостереження, які збільшують рівень тривоги; термінали, що активують події; пастки та барикади. Взаємодія з

ними реалізована через компоненти та Blueprint-логіку, пов’язану з квестами, фракціями та ШІ; приклад вигляду охоронної камери, що впливає на рівень тривоги та поведінку NPC, наведено на рисунку 4.7.

На рівнях розміщуються навігаційні сітки (NavMesh), що дозволяють NPC пересуватися з урахуванням перешкод. Для зон підвищеного ризику та контрольних точок створюються спеціальні маркери, які використовуються Behavior Trees та EQS для побудови маршрутів і вибору позицій спостереження. Додатково такі маркери дозволяють задавати пріоритетні зони чергування, місця для засідок і точки regroup-поведінки, коли NPC втрачає гравця з поля зору, але продовжує його пошук.

Розміщення тригерів забезпечує запуск квестових подій, зміни фракційних станів, активацію кат-сцен та зміну складності (наприклад, посилення патрулів після провалу завдання). Частина тригерів працює у фоновому режимі, реагуючи на приховані параметри сесії (час, шум, рівень тривоги), що робить рівні не лише простором для переміщення, а й активним елементом логіки гри, тісно пов’язаним зі ШІ та наративом.

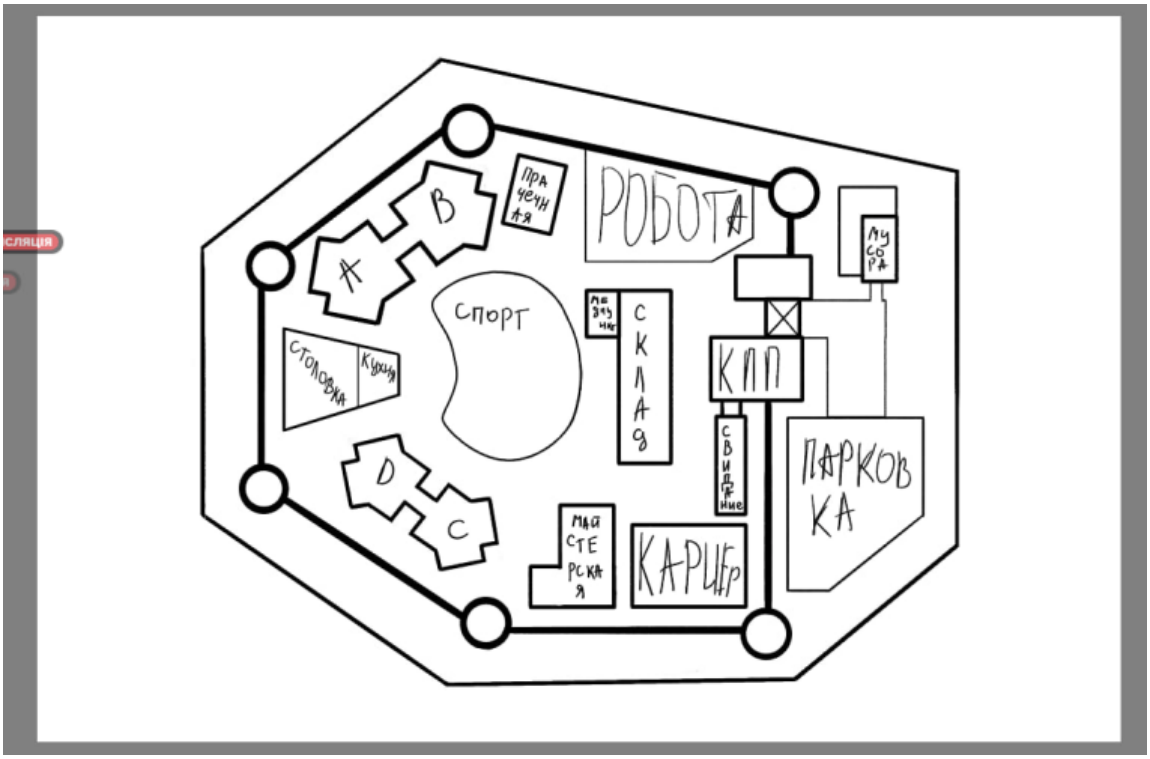


Рисунок 4.6 – Ескіз карти рівня гри «Втеча»

Рисунок 4.7 – Вигляд охоронної камери

4.8 Оптимізація гри

Оптимізація у грі «Втеча» спрямована на забезпечення стабільної продуктивності за збереження складної логіки ШІ та високої деталізації сцен. Для цього застосовано низку підходів на рівні контенту, коду та налаштувань рушія.

На рівні ШІ використано: зниження частоти оновлення поведінкових дерев для NPC, що знаходяться далеко від гравця; деактивацію частини сенсорів у неактуальних зонах; обмеження максимальної кількості одночасно активних агентів, які виконують складні EQS-запити; кешування результатів деяких обчислень.

На рівні графіки застосовано: рівні деталізації (LOD) для моделей; відсікання об’єктів поза полем зору (frustum culling); спрощення матеріалів для другорядних елементів; оптимізацію освітлення через використання поєднання динамічного та попередньо розрахованого освітлення там, де це можливо.

На рівні даних та логіки систем: мінімізацію дублювання інформації у DataTables; використання посилань замість копіювання структур; поділ великих

систем на підмодулі, що активуються лише за потреби; профілювання вузьких місць за допомогою вбудованих інструментів Unreal Engine (Session Frontend, профайлери), що дає змогу виявляти неефективні запити, оптимізувати структуру зберігання параметрів і зменшувати кількість зайвих обчислень під час виконання гри.

Результатом є зменшення пікових навантажень на процесор та відеокарту, скорочення часу реакції системи на дії гравця та забезпечення плавного ігрового процесу в межах заданих апаратних вимог, що особливо важливо для стабільної роботи ігрової системи на конфігураціях середнього рівня та під час тривалих ігрових сесій.

4.9 Висновки до розділу

У розділі 4 було розглянуто практичну реалізацію основних підсистем гри «Втеча» на платформі Unreal Engine 5. Реалізовано ШІ-систему на основі Behavior Trees, Blackboard, AI Perception та EQS, що забезпечує адаптивну поведінку NPC, здатність реагувати на дії гравця, зміни середовища та фракційний контекст. Впроваджено фракційну модель та систему репутації, які впливають на доступні квести, торгівлю та сценарні гілки.

Система квестів та сценарних переходів забезпечує підтримку розгалуженого сюжету із залежністю шляху проходження від рішень гравця. Реалізовано інтерфейс користувача, інвентар, взаємодію з предметами, торгові інтерфейси та діалогові вікна, що надають гравцю повний доступ до механік гри. Побудовано рівні та інтерактивне середовище, у якому поєднуються маршрути NPC, зони ризику, фракційні території та квестові тригери.

Особливу увагу приділено оптимізації, що дає змогу підтримувати стабільну продуктивність при наявності складних ШІ- та сценарних систем. Реалізовані рішення підтверджують практичну здійсненність обраних

архітектурних підходів та технологій, створюючи базу для подальшого дослідження роботи системи, аналізу ефективності ШІ та оцінки впливу фракційної та квестової логіки на якість геймплейного досвіду в наступному розділі.

5 ДОСЛІДЖЕННЯ ТА ТЕСТУВАННЯ СИСТЕМИ

5.1 Методика тестування

Дослідження та тестування гри «Втеча» спрямовані на оцінювання коректності роботи геймплейних підсистем, ефективності ШІ, адекватності фракційної взаємодії та стабільності функціонування системи за різних сценаріїв використання. Запропонована методика тестування поєднує функціональне, сценарне, навантажувальне та експериментальне тестування з використанням контрольних сценаріїв, повторюваних експериментів і фіксації результатів у формі логів та узагальнених табличних даних.

Основними етапами методики є: визначення цілей тестування для кожного модуля системи; формування набору тестових сценаріїв для ШІ, фракційної системи, квестів, інвентаря та рівнів; проведення серії запусків гри з фіксацією параметрів (час реакції ШІ, частка коректних реакцій, зміни репутації, стабільність FPS); аналіз отриманих даних і порівняння їх із очікуваними значеннями.

Для тестування ШІ та фракційних механік застосовується підхід контрольованих сценаріїв, коли гравець свідомо повторює однакові дії: наближення до NPC з різних напрямків і на різній відстані; створення шумів за межами поля зору; вхід у зони підвищеної охорони; послідовне виконання або провал квестів певних фракцій. Це дає змогу оцінити стабільність і передбачуваність реакцій системи. Навантажувальне тестування проводиться через штучне збільшення кількості активних NPC і подій, а також вимірювання

продуктивності на заданих апаратних конфігураціях.

Результати досліджень фіксуються у вигляді текстових логів гри, внутрішніх логів ШІ-контролерів, показників продуктивності з використанням профайлера Unreal Engine та табличних даних для подальшого аналізу. Для кожного набору сценаріїв виконується серія повторюваних запусків, що дає змогу усереднити результати та зменшити вплив випадкових коливань продуктивності. Такий підхід дозволяє обґрунтовано оцінити якість реалізації системи та виявити потенційні вузькі місця, а також порівняти різні конфігурації налаштувань ШІ, параметрів рівня й оптимізаційних рішень на основі об’єктивних кількісних показників.

5.2 Тестове середовище та апаратна конфігурація

Реалізація та тестування гри «Втеча» здійснювалися на персональних комп’ютерах, конфігурація яких забезпечує коректну роботу рушія Unreal Engine 5 та дає змогу оцінити продуктивність, стабільність і відгук системи за умов, наближених до типових для сучасних ігор. Базовим середовищем розробки та тестування виступала конфігурація A, що характеризується сучасним багатоядерним процесором, дискретним графічним адаптером середньо-високого класу, достатнім обсягом оперативної пам’яті та високошвидкісним NVMe-накопичувачем. Додатково використовувалася конфігурація B, яка за обчислювальними можливостями є дещо скромнішою. Це дало змогу проаналізувати чутливість гри до зниження ресурсів апаратної платформи та оцінити мінімально комфортні умови для запуску.

Програмне тестове середовище включало сучасні версії 64-розрядних операційних систем сімейства Windows, рушій Unreal Engine 5 гілки 5.4.x, а також інструменти, необхідні для розробки й аналізу продуктивності: інтегроване середовище розробки для роботи з C++-кодом, систему управління

версіями Git, утиліти моніторингу навантаження на CPU, GPU, оперативну пам’ять та підсистему зберігання даних. Узагальнені характеристики апаратного та програмного забезпечення для основних тестових конфігурацій наведено в таблиці 5.2.

Таблиця 5.2 – Тестове середовище та апаратна конфігурація

Конфігурація	**CPU**	**GPU**	**RAM**	**Накопичувач**	**ОС**	**Версія UE5**
A (основна)	AMD Ryzen 5 5600X (6 ядер, 12 потоків)	NVIDIA GeForce RTX 4070 12 ГБ	32 ГБ DDR4-3200	SSD NVMe 1 ТБ	Windows 11 Pro 23H2 64-bit	Unreal Engine 5.4.2
B (додаткова)	AMD Ryzen 5 3600 (6 ядер, 12 потоків)	NVIDIA GeForce RTX 3060 12 ГБ	16 ГБ DDR4-3200	SSD NVMe 512 ГБ	Windows 10 Pro 22H2 64-bit	Unreal Engine 5.4.2

5.3 Оцінювання ефективності ШІ та фракційної взаємодії

Оцінювання ефективності ШІ та фракційної взаємодії у грі «Втеча» зосереджується на трьох основних аспектах: коректності поведінки NPC; швидкості реакції системи на події; впливі рішень гравця на стан світу, ставлення фракцій та доступність геймплейного контенту. Для цього формуються тестові ситуації, у яких поведінка NPC і фракційної системи може бути однозначно оцінена як коректна або некоректна.

Для ШІ формуються сценарії, що охоплюють типові ситуації: наближення гравця до NPC з різних напрямків та на різній відстані; створення шуму за межами поля зору; тимчасова втрата гравця з поля зору та подальший пошук; перебування гравця в зонах підвищеної охорони; зміна фракційної належності гравця або репутації. Для кожного сценарію фіксується, чи виявив NPC гравця, через який час відбулася реакція, які дії були обрані (переслідування, підозра, виклик підкріплення, ігнорування). На основі цього аналізується стабільність

роботи Behavior Trees, налаштувань AI Perception та конфігурації EQS.

Ефективність ШІ характеризується показниками: частка коректних реакцій за низки повторень; середній час реакції на подію; максимальний час реакції; частота неконсистентної поведінки (ігнорування очевидної загрози, надто різка або нелогічна реакція). Для аналізу впливу навантаження на час реакції було проведено експеримент, у межах якого змінювалася кількість одночасно активних NPC на сцені, а час реакції вимірювався за декілька повторень і усереднювався. Вихідні дані для побудови графіка залежності часу реакції ШІ від кількості NPC наведено в таблиці 5.3.

Таблиця 5.3 – Результати вимірювання часу реакції ШІ залежно від кількості NPC

Кількість NPC	**Середній час реакції, с**	**Максимальний час реакції, с**
10	0,22с	0,50
20	0,38	0,55
30	0,42	0,63
40	0,45	0,70

На основі даних таблиці 5.3 побудовано графік на рисунку 5.1, де показано залежність часу реакції ШІ від кількості NPC. Зі збільшенням кількості одночасно активних противників середній час реакції зростає помірно (від 0,35 с для 10 NPC до 0,45 с для 40 NPC), тоді як максимальний час реакції залишається в діапазоні до 0,70 с. У всіх протестованих конфігураціях ШІ зберігає час реакції менше ніж 1 с, що забезпечує достатньо швидке сприйняття дій гравця та підтверджує стабільність реалізованої логіки поведінки навіть за підвищеного навантаження зображено на рисунку 5.1.

Тестування фракційної взаємодії спрямоване на перевірку коректності роботи системи фракцій і репутації, а також на оцінку того, наскільки відчутно для гравця впливають його дії на ставлення NPC і доступність квестів та торгівлі. Для цього формуються сценарії, у яких гравець послідовно: виконує квести однієї фракції; порушує інтереси окремих угруповань; здійснює агресивні дії щодо NPC різних фракцій; уникає участі у конфліктах. У кожному

сценарії відстежуються значення репутації, зміни у доступності діалогів, торгових операцій і квестів, а також те, чи відбувається очікувана зміна ставлення фракцій.

Особливу увагу приділено симетричності та узгодженості змін (чи адекватно реагують ворожі фракції на підтримку їхніх опонентів); коректності роботи порогових значень репутації, що розмежовують стани союзника, нейтрального та ворога; відсутності логічних суперечностей, коли фракція формально ворожа, але продовжує видавати квести або торгувати на звичайних умовах. За результатами тестів коригуються матриця відносин фракція–фракція, величини зміни репутації за різні дії та умови відкриття/блокування контенту. Це забезпечує передбачуваність ігрової логіки, підвищує довіру гравця до системи та зменшує ризик виникнення аномалій поведінки, що руйнують ігрове занурення.

Отримані результати тестування фракційної взаємодії дозволяють оцінити, наскільки фракційна система впливає на реіграбельність гри: чи змінюються сценарні гілки залежно від вибору гравця; чи відкриваються нові варіанти втечі; чи відчувається вагомість прийнятих рішень щодо підтримки або протистояння конкретним угрупованням. У випадку виявлення сценаріїв, де реакція світу є недостатньою або суперечливою, вносяться зміни до правил модифікації репутації, логіки квестів і доступності торгових пропозицій, що посилює роль фракційної моделі в загальній структурі геймплею. Додатково аналізуються крайні випадки, коли гравець навмисно обирає радикальну стратегію (повний нейтралітет або максимальну ворожість до всіх угруповань): це дає змогу перевірити, чи зберігається ігрова мотивація, чи не виникають «мертві зони» прогресу та чи залишається доступним хоча б один осмислений шлях завершення гри з урахуванням попередніх рішень.

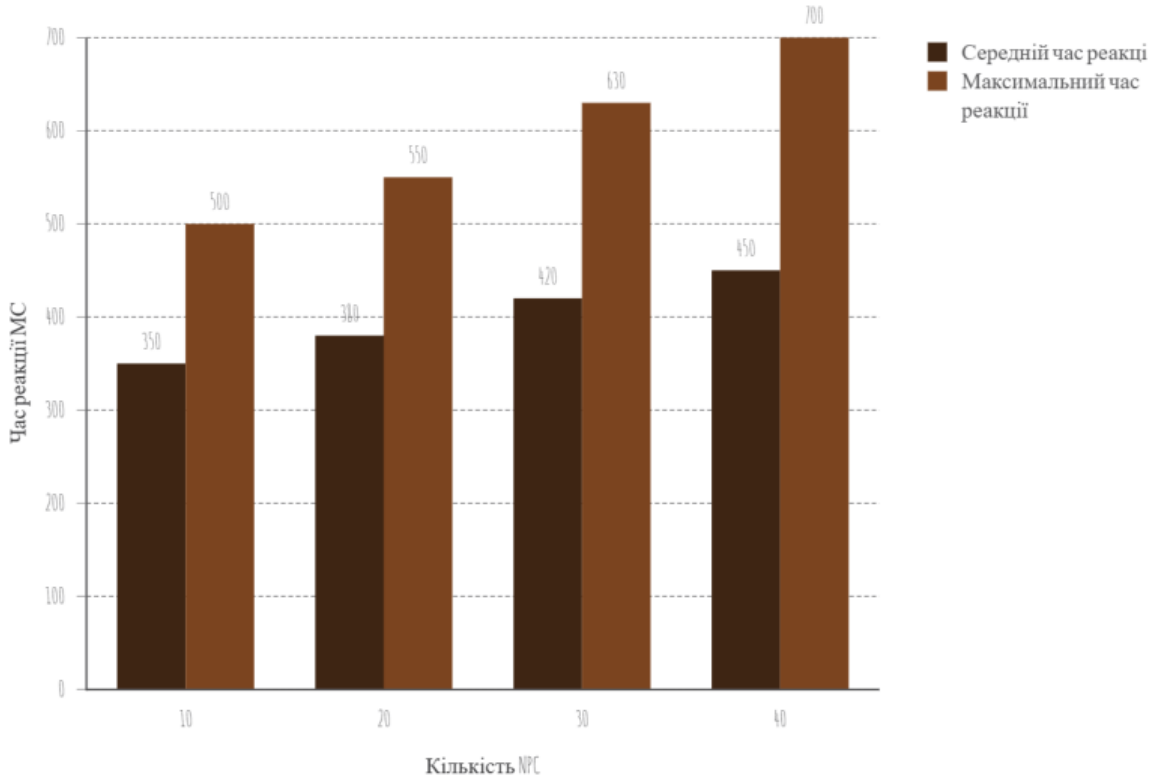


Рисунок 5.1 – Графік залежності часу реакції ШІ від кількості NPC

5.4 Навантажувальне тестування UE5-системи

Навантажувальне тестування має на меті визначити межі стабільної роботи гри «Втеча» при збільшеній кількості активних об’єктів, NPC, подій та взаємодій, а також оцінити вплив інтенсивної роботи ШІ та квестової логіки на продуктивність системи. Для цього створюються тестові конфігурації рівнів, у яких: збільшується кількість NPC, задіяних у патрулюванні та переслідуванні; активується одночасно кілька квестів із різними станами; генерується підвищена кількість подій, пов’язаних із фракційними змінами, тригерами рівня, торгівлею та діалогами.

У процесі навантажувальних тестів фіксуються: середній та мінімальний FPS; час відгуку інтерфейсу; затримки у реакціях ШІ; поява мікрофризів або значних просідань продуктивності. Для аналізу використовуються профайлери

Unreal Engine, що дозволяють визначити, які підсистеми створюють основне навантаження: рендеринг; скриптова логіка; ШІ; фізика; завантаження ресурсів. На основі отриманих результатів робляться висновки щодо необхідності додаткової оптимізації: зменшення частоти оновлення окремих систем; обмеження чисельності активних NPC; перегляду рівня деталізації контенту або спрощення окремих механік у найбільш навантажених сценах.

5.4.1 Залежність FPS від кількості NPC

Для оцінювання впливу кількості одночасно активних NPC на продуктивність системи було проведено окрему серію навантажувальних тестів. У рамках експерименту розглядалися такі конфігурації сцени за кількістю ворожих NPC: 5, 15, 25, 35 та 50 агентів, які одночасно патрулюють, реагують на гравця та генерують події. Вимірювався середній FPS за серію повторюваних запусків. Узагальнені результати наведено в таблиці 5.4.

Таблиця 5.4 – Залежність середнього FPS від кількості NPC під час навантажувального тестування

Кількість NPC	**Середній FPS**
5	130
15	110
25	80
35	55
50	42

На основі даних таблиці 5.4 побудовано графік на рисунку 5.2, де показано залежність середнього FPS від кількості NPC. Із зростанням кількості одночасно активних NPC середній FPS поступово зменшується: від приблизно 130 кадрів за секунду для 5 NPC до близько 40–45 кадрів за секунду для 50 NPC. Це демонструє очікуване падіння продуктивності за рахунок зростання обчислювального навантаження на підсистеми ШІ, фізики та рендерингу. Водночас навіть за 35–50 NPC система зберігає прийнятний рівень продуктивності для цільової платформи, що свідчить про достатню ефективність застосованих оптимізацій зображено на рисунку 5.2.

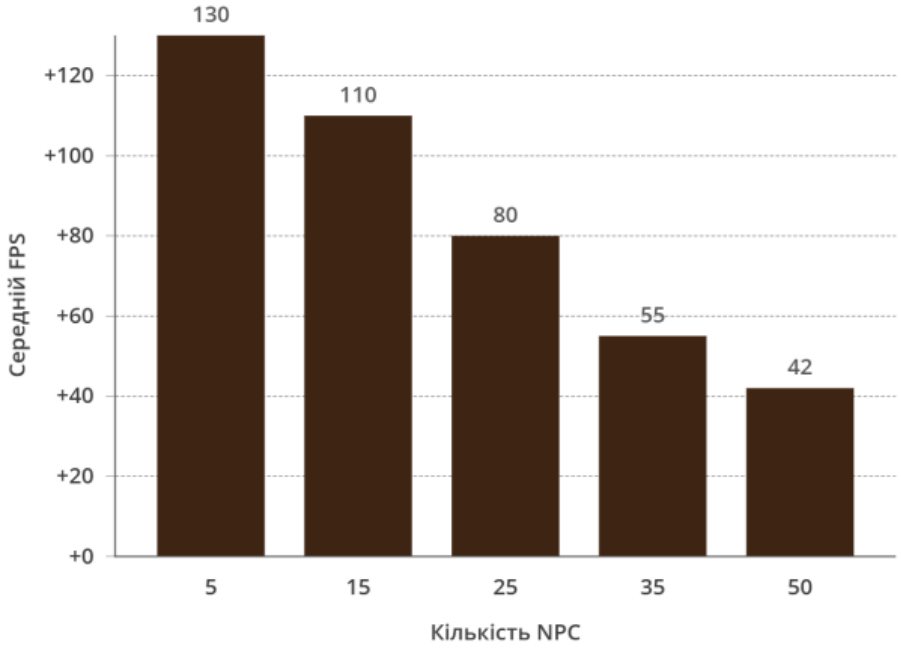


Рисунок 5.2 – Графік залежності середнього FPS від кількості NPC під час навантажувального тестування

5.4.2 Залежність FPS від типу ігрової сцени

Для проведення навантажувального тестування було розглянуто чотири типові ігрові сцени, що відрізняються рівнем складності та кількістю активних підсистем рушія: сцена з мінімальним навантаженням (головне меню); сцена з базовим коридором і статичними об’єктами; сцена з активними NPC, тригерами та квестовою логікою; сцена з максимальним навантаженням, де одночасно працюють NPC, ефекти, елементи UI та паралельні події. Для кожної сцени вимірювався середній FPS у фіксованій точці камери за однакових налаштувань графіки, на одній апаратній конфігурації та протягом однакового проміжку часу, що дає змогу коректно порівняти вплив ускладнення сцени на продуктивність системи. Результати наведено в таблиці 5.5. На основі цих даних побудовано графік залежності FPS від складності сцени, що використовується для аналізу та обґрунтування архітектурних рішень. Додатково фіксувався розкид значень FPS, що дозволило оцінити стабільність кадрів і виявити потенційні вузькі місця у структурі сцен та ресурсів.

Таблиця 5.5 – Залежність середнього FPS від типу ігрової сцени

Сцена	**Опис**	**Середній FPS**
Сцена 1	Головне меню / мінімальне навантаження	155
Сцена 2	Базовий коридор з освітленням та статичними об’єктами	145
Сцена 3	Ігрова зона з активними NPC, тригерами та квестовою логікою	125
Сцена 4	Максимальне навантаження: кілька NPC, активні ефекти, елементи UI та паралельні події	110

На основі таблиці 5.5 побудовано графік на рисунку 5.3, що відображає залежність середнього FPS від типу сцени. Із ускладненням сцени та збільшенням кількості об’єктів, активних NPC, елементів інтерфейсу й подієвих обробників середній FPS очікувано знижується з приблизно 155 до 110 кадрів за секунду. При цьому навіть для найбільш навантаженої сцени продуктивність залишається в межах, комфортних для геймплею, що підтверджує коректність розподілу навантаження між підсистемами рушія та адекватність прийнятих рішень щодо оптимізації зображено на рисунку 5.3.

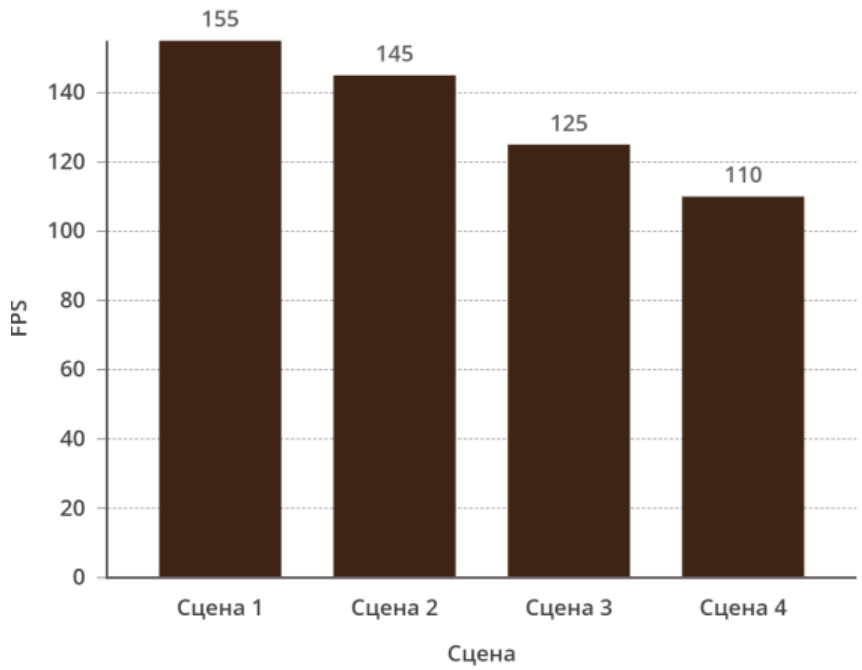


Рисунок 5.3 – Графік залежності середнього FPS від типу ігрової сцени

5.5 Аналіз результатів експериментів

Як показують графіки на рисунках 5.1–5.3, які відображають зміну основних показників у залежності від кількості NPC та складності сцени, та відповідні їм таблиці 5.3–5.5, що містять детальні числові результати для кожної протестованої конфігурації, при збільшенні кількості NPC і ускладненні сцен система зберігає прийнятний рівень продуктивності, а час реакції ШІ залишається меншим за 1 с у всіх протестованих конфігураціях. Це свідчить про те, що обрані архітектурні рішення та параметри налаштування підсистем ШІ є достатньо ефективними для цільових сценаріїв використання.

Аналіз результатів оцінювання ШІ показує, що система забезпечує контекстно залежну поведінку NPC, реагуючи на дії гравця та зміни середовища в межах прийнятних часових затримок. Стабільність часу реакції за зростання кількості NPC вказує на те, що реалізація Behavior Trees, AI Perception та EQS не створює критичних «вузьких місць», а зафіксовані випадки неконсистентної поведінки можуть бути локально виправлені через корекцію порогових значень і параметрів оновлення запитів.

Результати тестування фракційної взаємодії демонструють, що система репутації та матриця відносин між фракціями коректно відображають наслідки дій гравця: змінюється доступність квестів і торгових операцій, з’являються або блокуються альтернативні сценарні гілки та варіанти втечі. Виявлені під час тестування неузгодженості (наприклад, збереження доступу до квестів при ворожому статусі фракції) можуть бути усунуті через уточнення порогів репутації та умов відкриття контенту, що підвищує логічну цілісність ігрового світу.

Навантажувальне тестування показує, що система готова до роботи в умовах підвищеної складності, з великою кількістю активних NPC і подій. Зниження FPS за зростання навантаження є очікуваним і не виходить за межі комфортних значень для цільової апаратної конфігурації. При цьому результати

експериментів слугують основою для подальшого вдосконалення: за потреби можна додатково обмежити кількість одночасно активних NPC, оптимізувати складні сцени або внести зміни до алгоритмів оновлення ШІ.

5.6 Висновки до розділу

У розділі 5 було розглянуто підхід до дослідження та тестування гри «Втеча», спрямований на комплексну оцінку коректності роботи геймплейних підсистем, ефективності ШІ, фракційної взаємодії та продуктивності системи. Запропонована методика тестування включає виконання контрольованих сценаріїв, повторюваних експериментів і навантажувальних випробувань із подальшим аналізом логів і табличних даних, на основі яких побудовані узагальнюючі графіки.

Оцінювання ефективності ШІ показало, що система здатна забезпечувати контекстно залежну поведінку NPC, реагувати на дії гравця та зміни середовища в межах прийнятних часових затримок. Аналіз реакцій NPC дав змогу виявити й скоригувати порогові значення окремих параметрів, підвищивши стабільність і логічність поведінки без суттєвого погіршення продуктивності.

Тестування фракційної взаємодії підтвердило, що система репутації й фракційних відносин впливає на доступність квестів, торгівлі та сценарних шляхів, створюючи альтернативні варіанти проходження та різні фінали. Це свідчить про успішну інтеграцію фракційної логіки в загальну геймплейну модель і підвищує реіграбельність гри.

Навантажувальне тестування дало змогу оцінити межі продуктивності системи при великій кількості активних NPC і подій. Отримані результати показали, що завдяки застосованим методам оптимізації гра підтримує прийнятний рівень продуктивності на цільовій апаратній конфігурації, а

виявлені потенційні «вузькі місця» можуть бути адресно оптимізовані.

Загалом результати дослідження підтверджують, що обрані архітектурні та технологічні рішення є придатними для створення survival escape-гри з адаптивним ШІ, фракційною моделлю та нелінійним сценарієм. Проведені експерименти та тестування формують основу для висновків дипломної роботи та визначення напрямів подальшого розвитку гри.

ВИСНОВКИ

У дипломній роботі здійснено комплексне дослідження процесу проєктування, командної розробки та реалізації комп’ютерної гри «Втеча» у жанрі survival escape на платформі Unreal Engine 5. На основі аналізу сучасних ігор, ігрових рушіїв та підходів до побудови штучного інтелекту обґрунтовано вибір рушія Unreal Engine 5, гібридного підходу C++/Blueprint та модульної архітектури геймплейних систем, орієнтованої на нелінійний геймплей, фракційну взаємодію та адаптивну поведінку NPC.

У ході виконання роботи отримано такі основні результати: проаналізовано жанр survival escape, виявлено його характерні риси та вимоги до ШІ, інтерактивності середовища й реіграбельності; обґрунтовано вибір Unreal Engine 5 порівняно з альтернативними рушіями та визначено доцільність використання змішаного підходу C++/Blueprint для поєднання продуктивності й гнучкості; спроєктовано архітектуру гри «Втеча» з виділенням модулів штучного інтелекту, фракційної системи, квестів, інвентаря, торгівлі, інтерфейсу користувача та підсистеми роботи з даними на основі DataTables; реалізовано ШІ-систему на базі Behavior Trees, Blackboard, AI Perception та EQS, що забезпечує контекстно-залежну й адаптивну поведінку NPC; розроблено систему фракційної взаємодії та репутації, яка змінює доступність квестів, торгівельних і діалогових сценаріїв; реалізовано квестову підсистему, інвентар, HUD, інтерфейс журналу завдань, торгові та діалогові вікна, а також рівні з зонами патрулювання, тригерами подій та фракційними територіями; побудовано методику тестування гри, проведено функціональні, сценарні й навантажувальні експерименти, за результатами яких оцінено коректність роботи геймплейних підсистем, ефективність ШІ та межі стабільної продуктивності за різних конфігурацій навантаження.

Окремо підтверджено ефективність командного методу розробки для створення складних ігрових систем. У процесі роботи над грою «Втеча»

продемонстровано практичне застосування розподілу ролей, системи керування версіями Git, ітеративного планування завдань та узгодження змін у єдиному репозиторії, що забезпечило керованість еволюції програмного засобу, прозорість історії модифікацій і можливість масштабування функціоналу.

Практичне значення отриманих результатів полягає у створенні програмного засобу, який може слугувати основою для подальшого розвитку як самостійного ігрового продукту жанру survival escape, так і навчальної платформи для вивчення архітектури ігрових систем, поведінкового штучного інтелекту та командних підходів до розробки в Unreal Engine 5. Запропоновані архітектурні рішення, підходи до організації геймплейних модулів, побудови ШІ на основі Behavior Trees та EQS, реалізації фракційної логіки та роботи з даними через DataTables можуть бути повторно використані та адаптовані у комерційних і навчальних роботах, пов’язаних із розробкою інтерактивних симуляцій та ігор.

Поставлена у дипломній роботі мета досягнута: спроєктовано та реалізовано комп’ютерну гру «Втеча» у жанрі survival escape на платформі Unreal Engine 5 з використанням командного методу розробки, модульної архітектури, адаптивної ШІ-системи, фракційної моделі, квестової структури та інтерактивного середовища, а також проведено дослідження й тестування її основних підсистем. Подальший розвиток роботи може бути спрямований на розширення фракційної та економічної моделей, впровадження мережевих кооперативних режимів, поглиблення поведінки NPC із застосуванням додаткових аналітичних або машинних підходів, а також на перенесення гри на інші платформи та інтеграцію з розширеними інструментами аналітики гравецьких сесій.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Нікітченко С. О., Скрупський С. Ю. Кіберзагрози в месенджерах: аналіз вразливостей та методи їх експлуатації, Тиждень науки – 2025. Факультет комп'ютерних наук і технологій: тези доповідей наук.-практ. конф. (Запоріжжя, 14–18 квіт. 2025 р.). Запоріжжя: НУ «Запорізька політехніка», 2025. С. 71–73. URL: https://eir.zp.edu.ua/handle/123456789/23287

2. Richard N. The art of survival design: a blueprint to survival game design beyond the genre, Tampere University of Applied Sciences (2024) 53. URL: https://urn.fi/URN:NBN:fi:amk-2024120231812.

3. Zhu Y., Kim D., Alsheimer A., A review of game design techniques for managing suspense, in: International Conference on ArtsIT, Interactivity and Game Creation, Springer, 2022, pp. 174–186. URL: https://link.springer.com/chapter/10.1007/978-3-031-28993-4_13.

4. Marak K. Independent horror games between 2010 and 2020: Selected characteristic features and discernible trends, Images. The International Journal of European Film, Performing Arts and Audiovisual Communication 29 (2021) 175–190. URL: https://www.ceeol.com/search/article-detail?id=1018187.

5. Liepa K. K. Psyche as a Playable Construct in Video Games, Master's thesis, The University of Bergen, 2023. URL: https://hdl.handle.net/11250/3071930.

6. Rahimi S., Walker J. T., Lin-Lipsmeyer L., Shin J., Toward defining and assessing creativity in sandbox games, Creativity Research Journal 36 (2024) 194–212. doi:10.1080/10400419.2022.2156477.

7. Konttinen A., Core features of 3d platformers, Jamk, 2024. 32. URL: https://urn.fi/URN:NBN:fi:amk-2024122238039.

8. Unity Documentation, Unity Technologies, 2025. URL: https://docs.unity.com.

9. Linietsky J., Manzur A., Redot Engine 4.3 documentation in English, Redot community, 2024. URL: https://docs.redotengine.org/en/stable/.

10. Hadi A. Designing platformer game using gamemaker studio,

International Journal of Art, Design, and Metaverse 2. 2024. 23–30. URL: https://topazart.info/e-journals/index.php/ijam/article/view/69/78.

11. Linietsky J., Manzur A., Godot docs. Godot Engine 4.5 documentation in English, Godot community, 2014. URL: https://docs.godotengine.org/en/stable/about/introduction.html.

12. Fritz B. Experiencing Gameworlds: Understanding Zelda and Mario Through the Lenses of Art History and Phenomenology, Lund University, 2025. URL: https://lup.lub.lu.se/search/publication/59a879f4-2fd4-4df4-a59c-9ad17bc36b56.

13. De Lope R. P., Medina-Medina N., Urbieta M., Lliteras A. B., Garcна A. M. A novel uml-based methodology for modeling adventure-based educational games, Entertainment Computing 38 (2021) 100429. doi:10.1016/j.entcom.2021.100429.

14. Poberailo O. The genre specificity of interactive cinema, Scientific Journal of Polonia University 69 (2025) 64–74. doi:10.23856/10.23856/6907.

15. Faisal N., Chadhar M., Goriss-Hunter A., Stranieri A. Business simulation games in higher education: A systematic reviewof empirical research, Human Behavior and Emerging Technologies 2022 (2022) 1578791. doi:10.1155/2022/1578791.

16. Unreal Engine, Epic Games, 2025. URL: https://www.unrealengine.com/en-US/features.

17. Unreal Engine 5 Documentation [Електронний ресурс] – URL: https://docs.unrealengine.com (дата звернення: 23.10.2025).

18. Felini D. Beyond today’s video game rating systems: A critical approach to pegi and esrb, and proposed improvements, Games and Culture 10 (2015) 106–122. doi:10.1177/1555412014560192.

19. Octaviani A. W., Irfansyah I. Formal game element analysis of rhythm fighting game, in: ICON ARCCADE 2021: The 2nd International Conference on Art, Craft, Culture and Design (ICONARCCADE 2021), Atlantis Press, 2021, pp. 243–251. doi:10.1155/2022/1578791.

20. Киричек Г. Г., Нікітченко С. О., Тягунова М. Ю. Комп’ютерна гра з використанням командного методу розробки, Міжнародна науково-практична конференція «Стратегічні орієнтири розвитку науки, освіти, технологій та ключових сфер суспільного життя» (Бостон, США, 10 лист. 2025 р.). Бостон, 2025. С. 112–116. URL: https://www.economics.in.ua/2025/11/10.html

ДОДАТОК А
ЛІСТИНГИ ПРОГРАМИ

Лістинг А.1 – Файл EscapeCharacter.h

```
#pragma once
#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "EscapeCharacter.generated.h"
UENUM(BlueprintType)
enum class EFactionRelation : uint8
{
	Neutral,
	Friendly,
	Hostile
};

UCLASS()
class ESCAPE_API AEscapeCharacter : public ACharacter
{
	GENERATED_BODY()

public:
	AEscapeCharacter();
	virtual void Tick(float DeltaTime) override;
	UFUNCTION(BlueprintCallable)
	void ApplyDamage(float Amount);
	UFUNCTION(BlueprintCallable)
	void AddStamina(float Amount);
	UFUNCTION(BlueprintCallable)
	void AddMoney(int32 Amount);
	UFUNCTION(BlueprintCallable)
	bool CanAfford(int32 Cost) const;
	UFUNCTION(BlueprintCallable)
	float GetHealthPercent() const;
```

```
    UFUNCTION(BlueprintCallable)
    EFactionRelation GetFactionRelation() const;
    UFUNCTION(BlueprintCallable)
    void ModifyFactionReputation(int32 Delta);
protected:
    virtual void BeginPlay() override;
    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly)
    float MaxHealth;
    UPROPERTY(VisibleAnywhere, BlueprintReadOnly)
    float CurrentHealth;
    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly)
    float MaxStamina;
    UPROPERTY(VisibleAnywhere, BlueprintReadOnly)
    float CurrentStamina;
    UPROPERTY(EditDefaultsOnly, BlueprintReadOnly)
    float StaminaRecoveryPerSecond;
    UPROPERTY(VisibleAnywhere, BlueprintReadOnly)
    int32 Money;
    UPROPERTY(EditAnywhere, BlueprintReadOnly)
    int32 FactionReputation;
    UPROPERTY(VisibleAnywhere, BlueprintReadOnly)
    EFactionRelation FactionRelation;
    void UpdateStamina(float DeltaTime);
    void UpdateFactionRelation();
};
```

Лістинг А.2 – Файл EscapeCharacter.cpp

```
#include "EscapeCharacter.h"
#include "GameFramework/PlayerController.h"

AEscapeCharacter::AEscapeCharacter()
{
    PrimaryActorTick.bCanEverTick = true;

    MaxHealth = 100.f;
```

```
        CurrentHealth = MaxHealth;

        MaxStamina = 100.f;
        CurrentStamina = MaxStamina;
        StaminaRecoveryPerSecond = 15.f;

        Money = 0;

        FactionReputation = 0;
        FactionRelation = EFactionRelation::Neutral;
    }

    void AEscapeCharacter::BeginPlay()
    {
        Super::BeginPlay();
        UpdateFactionRelation();
    }

    void AEscapeCharacter::Tick(float DeltaTime)
    {
        Super::Tick(DeltaTime);
        UpdateStamina(DeltaTime);
    }

    void AEscapeCharacter::ApplyDamage(float Amount)
    {
        if (Amount <= 0.f || CurrentHealth <= 0.f)
            return;

        CurrentHealth = FMath::Clamp(CurrentHealth - Amount, 0.f,
MaxHealth);

        if (CurrentHealth <= 0.f)
        {
            AController* Controller = GetController();
            if (Controller)
```

```
            {

DisableInput(Cast<APlayerController>(Controller));
            }
        }
    }

    void AEscapeCharacter::AddStamina(float Amount)
    {
        if (Amount <= 0.f)
            return;

        CurrentStamina = FMath::Clamp(CurrentStamina + Amount,
0.f, MaxStamina);
    }

    void AEscapeCharacter::AddMoney(int32 Amount)
    {
        Money += Amount;
    }

    bool AEscapeCharacter::CanAfford(int32 Cost) const
    {
        return Money >= Cost;
    }

    float AEscapeCharacter::GetHealthPercent() const
    {
        if (MaxHealth <= 0.f)
            return 0.f;

        return CurrentHealth / MaxHealth;
    }

    EFactionRelation AEscapeCharacter::GetFactionRelation() const
    {
```

```
        return FactionRelation;
    }

    void AEscapeCharacter::ModifyFactionReputation(int32 Delta)
    {
        FactionReputation = FMath::Clamp(FactionReputation +
Delta, -100, 100);
        UpdateFactionRelation();
    }

    void AEscapeCharacter::UpdateStamina(float DeltaTime)
    {
        CurrentStamina = FMath::Clamp(CurrentStamina +
StaminaRecoveryPerSecond * DeltaTime, 0.f, MaxStamina);
    }

    void AEscapeCharacter::UpdateFactionRelation()
    {
        if (FactionReputation >= 40)
        {
            FactionRelation = EFactionRelation::Friendly;
        }
        else if (FactionReputation <= -40)
        {
            FactionRelation = EFactionRelation::Hostile;
        }
        else
        {
            FactionRelation = EFactionRelation::Neutral;
        }
    }
```

Лістинг А.3 – Файл EnemyAIController.h

```
#pragma once
#include "CoreMinimal.h"
```

```
#include "AIController.h"
#include "EnemyAIController.generated.h"
UENUM(BlueprintType)
enum class EAIState : uint8
{
	Idle,
	Patrol,
	Chase,
	Attack,
	Flee
};
UCLASS()
class ESCAPE_API AEnemyAIController : public AAIController
{
	GENERATED_BODY()
public:
	AEnemyAIController();
	virtual void Tick(float DeltaTime) override;
	UFUNCTION(BlueprintCallable)
	void SetTargetActor(AActor* NewTarget);
	UFUNCTION(BlueprintCallable)
	EAIState GetState() const;
protected:
	virtual void OnPossess(APawn* InPawn) override;
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly)
	EAIState State;
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly)
	AActor* TargetActor;
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly)
	float SightRadius;
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly)
	float AttackDistance;
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly)
	float FleeHealthThreshold;
	void UpdateState();
	void HandleMovement();
```

```
    void HandleAttack();
};
```

Лістинг А.4 – EnemyAIController.cpp

```
#include "EnemyAIController.h"
#include "GameFramework/Character.h"
#include "Kismet/GameplayStatics.h"

AEnemyAIController::AEnemyAIController()
{
    PrimaryActorTick.bCanEverTick = true;

    State = EAIState::Idle;
    TargetActor = nullptr;

    SightRadius = 2000.f;
    AttackDistance = 200.f;
    FleeHealthThreshold = 0.2f;
}

void AEnemyAIController::OnPossess(APawn* InPawn)
{
    Super::OnPossess(InPawn);
}

void AEnemyAIController::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);

    UpdateState();
    HandleMovement();
    HandleAttack();
}

void AEnemyAIController::SetTargetActor(AActor* NewTarget)
```

```
{
	TargetActor = NewTarget;
}

EAIState AEnemyAIController::GetState() const
{
	return State;
}

void AEnemyAIController::UpdateState()
{
	APawn* ControlledPawn = GetPawn();
	if (!ControlledPawn)
		return;

	float HealthRatio = 1.f;

	if (HealthRatio <= FleeHealthThreshold)
	{
		State = EAIState::Flee;
		return;
	}

	if (!TargetActor)
	{
		State = EAIState::Patrol;
		return;
	}

	const float Distance = FVector::Dist(ControlledPawn->GetActorLocation(), TargetActor->GetActorLocation());

	if (Distance <= AttackDistance)
	{
		State = EAIState::Attack;
	}
```

```
        else if (Distance <= SightRadius)
        {
            State = EAIState::Chase;
        }
        else
        {
            State = EAIState::Patrol;
        }
    }

    void AEnemyAIController::HandleMovement()
    {
        APawn* ControlledPawn = GetPawn();
        if (!ControlledPawn)
            return;

        switch (State)
        {
        case EAIState::Patrol:
            break;
        case EAIState::Chase:
            if (TargetActor)
            {
                MoveToActor(TargetActor);
            }
            break;
        case EAIState::Flee:
            if (TargetActor)
            {
                const FVector Direction = (ControlledPawn->GetActorLocation() - TargetActor->GetActorLocation()).GetSafeNormal();
                const FVector Destination = ControlledPawn->GetActorLocation() + Direction * 600.f;
                MoveToLocation(Destination);
            }
```

```
            break;
        default:
            break;
        }
    }

    void AEnemyAIController::HandleAttack()
    {
        if (State != EAIState::Attack || !TargetActor)
            return;

        APawn* ControlledPawn = GetPawn();
        if (!ControlledPawn)
            return;

        const float Distance = FVector::Dist(ControlledPawn->GetActorLocation(), TargetActor->GetActorLocation());
        if (Distance > AttackDistance)
            return;

        const FRotator LookRotation = (TargetActor->GetActorLocation() - ControlledPawn->GetActorLocation()).Rotation();
        ControlledPawn->SetActorRotation(LookRotation);
    }
```

Лістинг А.5 – EscapeGameInstance.h

```
#pragma once

#include "CoreMinimal.h"
#include "Engine/GameInstance.h"
#include "GameFramework/SaveGame.h"
#include "EscapeGameInstance.generated.h"

UCLASS()
```

```
class ESCAPE_API UPlayerSaveGame : public USaveGame
{
     GENERATED_BODY()

public:
     UPROPERTY(EditAnywhere, BlueprintReadWrite)
     FVector PlayerLocation;

     UPROPERTY(EditAnywhere, BlueprintReadWrite)
     FRotator PlayerRotation;

     UPROPERTY(EditAnywhere, BlueprintReadWrite)
     int32 Money;

     UPROPERTY(EditAnywhere, BlueprintReadWrite)
     FString CurrentLevel;

     UPROPERTY(EditAnywhere, BlueprintReadWrite)
     int32 Difficulty;

     UPROPERTY(EditAnywhere, BlueprintReadWrite)
     TMap<FName, int32> StoryFlags;
};

UCLASS()
class ESCAPE_API UEscapeGameInstance : public UGameInstance
{
     GENERATED_BODY()

public:
     UEscapeGameInstance();

     UPROPERTY(EditAnywhere, BlueprintReadWrite)
     int32 DefaultDifficulty;

     UPROPERTY(EditAnywhere, BlueprintReadWrite)
```

```
        FString ActiveSaveSlot;

        UFUNCTION(BlueprintCallable)
        bool SaveCurrentState();

        UFUNCTION(BlueprintCallable)
        bool LoadCurrentState();
};
```

Лістинг А.6 – EscapeGameInstance.cpp

```
#include "EscapeGameInstance.h"
#include "Kismet/GameplayStatics.h"
#include "GameFramework/Character.h"

UEscapeGameInstance::UEscapeGameInstance()
{
        DefaultDifficulty = 1;
        ActiveSaveSlot = TEXT("MainSlot");
}

bool UEscapeGameInstance::SaveCurrentState()
{
        APawn* Pawn = UGameplayStatics::GetPlayerPawn(this, 0);
        if (!Pawn)
                return false;

        UPlayerSaveGame*                SaveObject                =
Cast<UPlayerSaveGame>(UGameplayStatics::CreateSaveGameObject(UPlay
erSaveGame::StaticClass()));
        if (!SaveObject)
                return false;

        SaveObject->PlayerLocation = Pawn->GetActorLocation();
```

```
          SaveObject->PlayerRotation = Pawn->GetActorRotation();
          SaveObject->CurrentLevel                                    =
UGameplayStatics::GetCurrentLevelName(this, true);
          SaveObject->Difficulty = DefaultDifficulty;
          SaveObject->Money = 0;

          return       UGameplayStatics::SaveGameToSlot(SaveObject,
ActiveSaveSlot, 0);
     }

     bool UEscapeGameInstance::LoadCurrentState()
     {
          if  (!UGameplayStatics::DoesSaveGameExist(ActiveSaveSlot,
0))
               return false;

          UPlayerSaveGame*               SaveObject                =
Cast<UPlayerSaveGame>(UGameplayStatics::LoadGameFromSlot(ActiveSav
eSlot, 0));
          if (!SaveObject)
               return false;

          APawn* Pawn = UGameplayStatics::GetPlayerPawn(this, 0);
          if (!Pawn)
               return false;

          Pawn->SetActorLocation(SaveObject->PlayerLocation);
          Pawn->SetActorRotation(SaveObject->PlayerRotation);

          DefaultDifficulty = SaveObject->Difficulty;

          return true;
     }
```

ДОДАТОК Б
БЛЮПРИНТИ

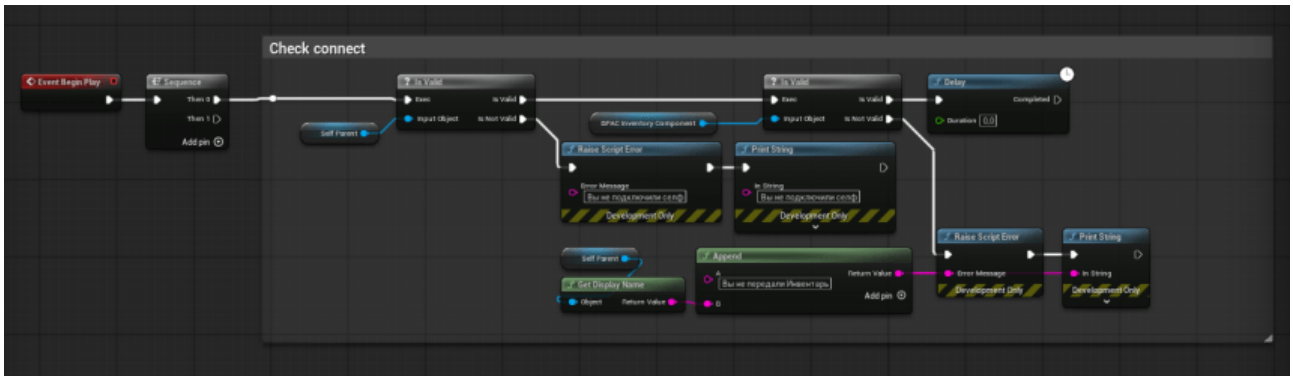


Рисунок Б.1 – BPAC_InteractGetComponent

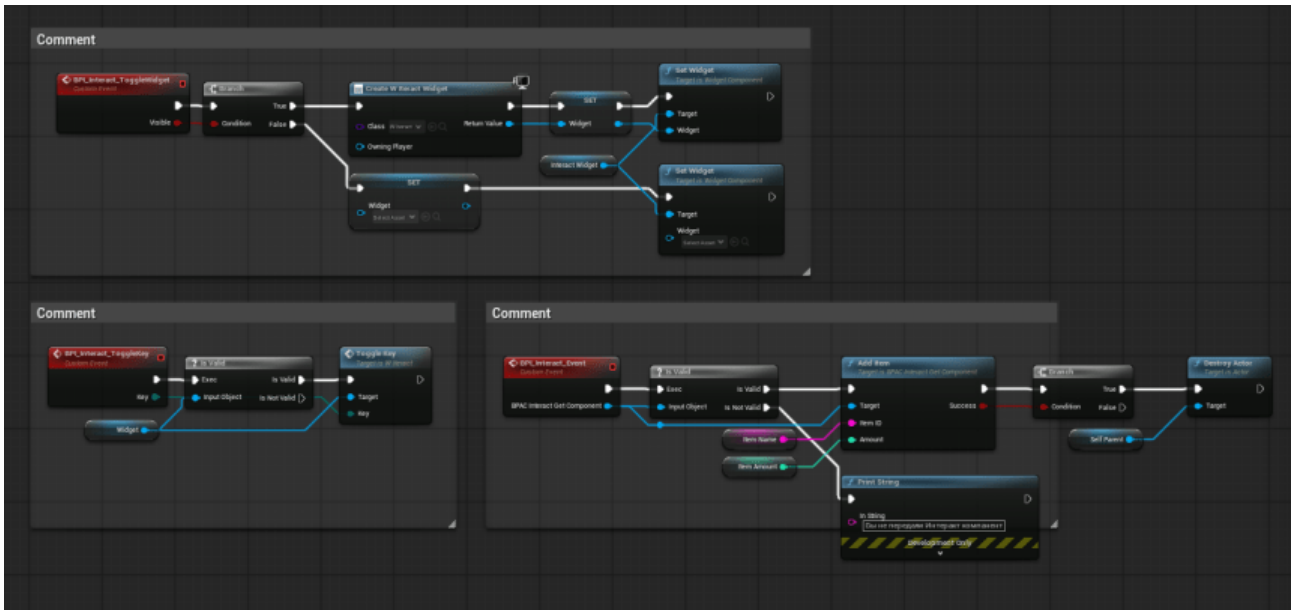


Рисунок Б.2 – BPAC_ItaractableComponent

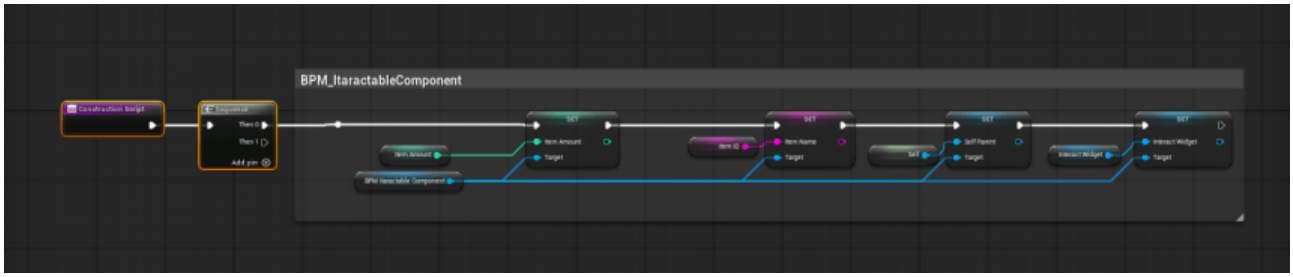


Рисунок Б.3 – BPM_Item

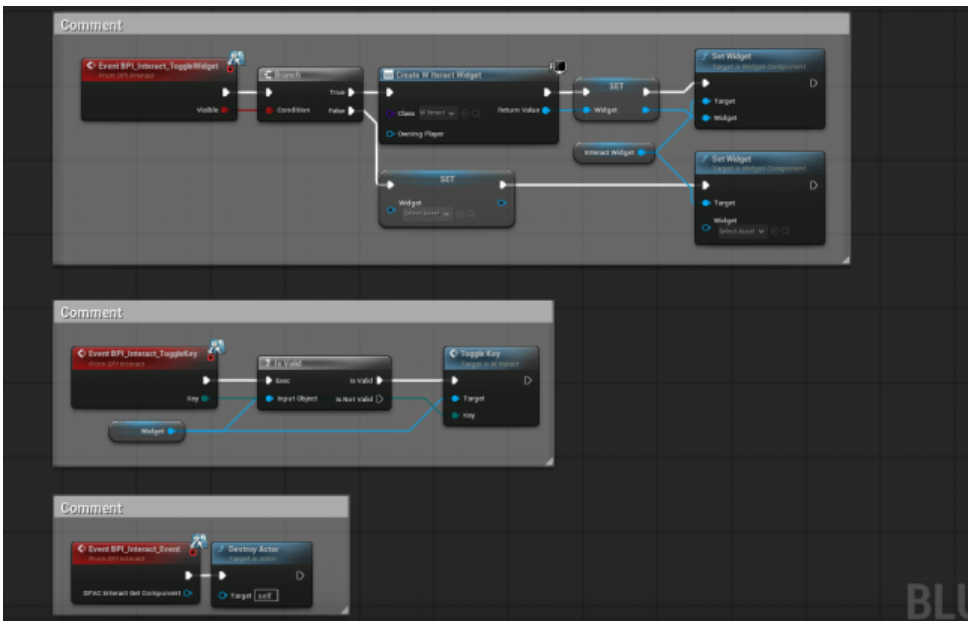


Рисунок Б.4 – BPM_Interactable

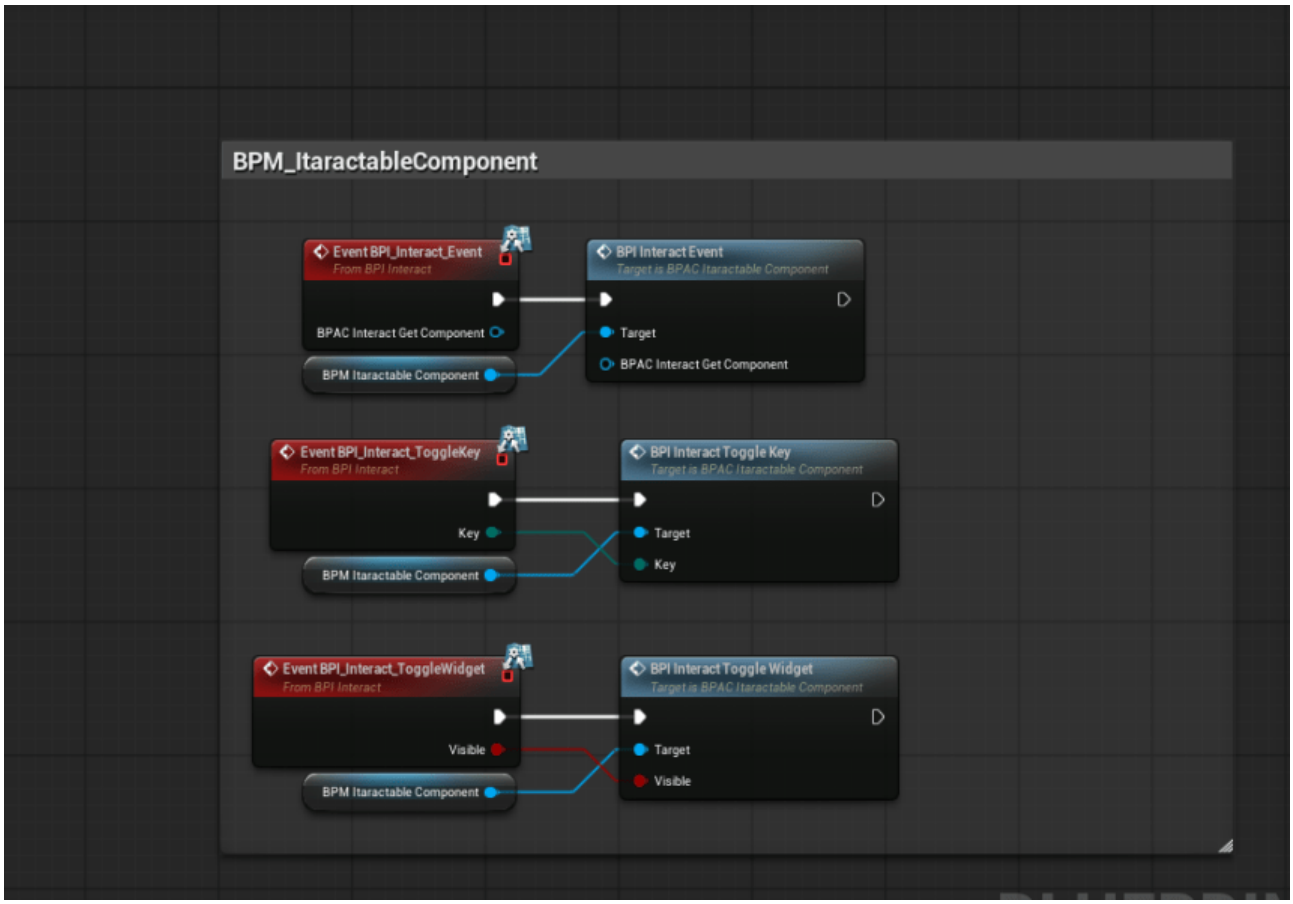


Рисунок Б.5 – BPM_NPC

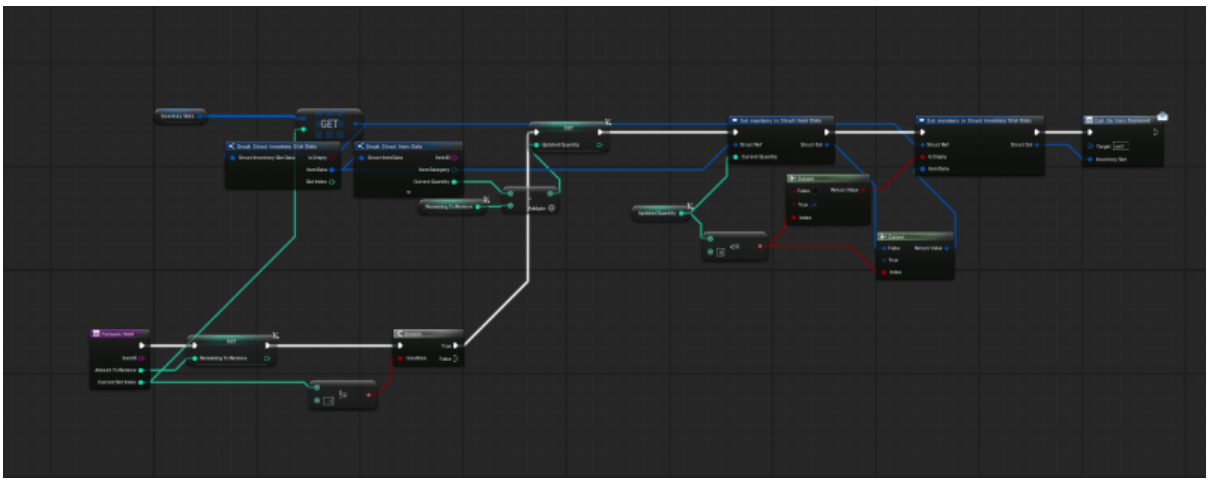


Рисунок Б.6 – BPAC_InventoryComponent

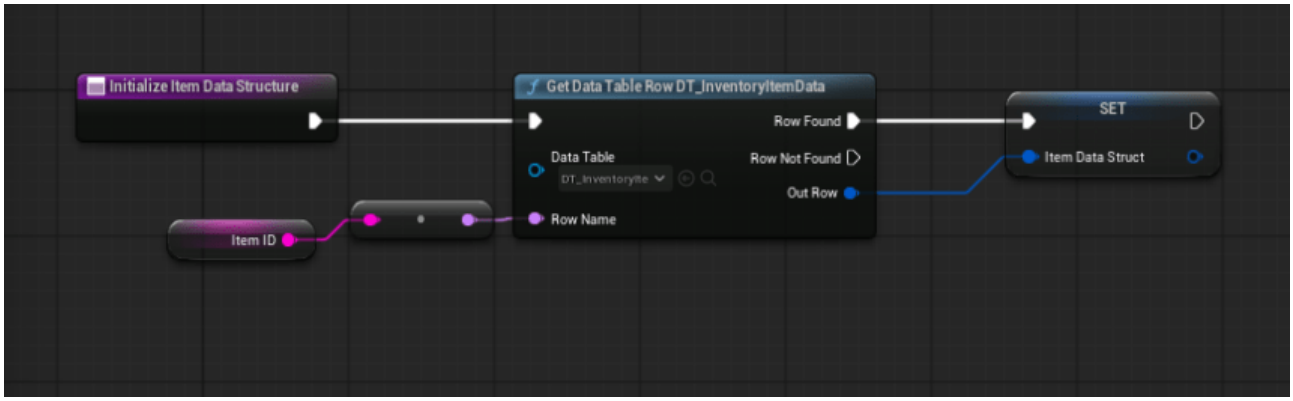


Рисунок Б.6 – Initialize Item Structure

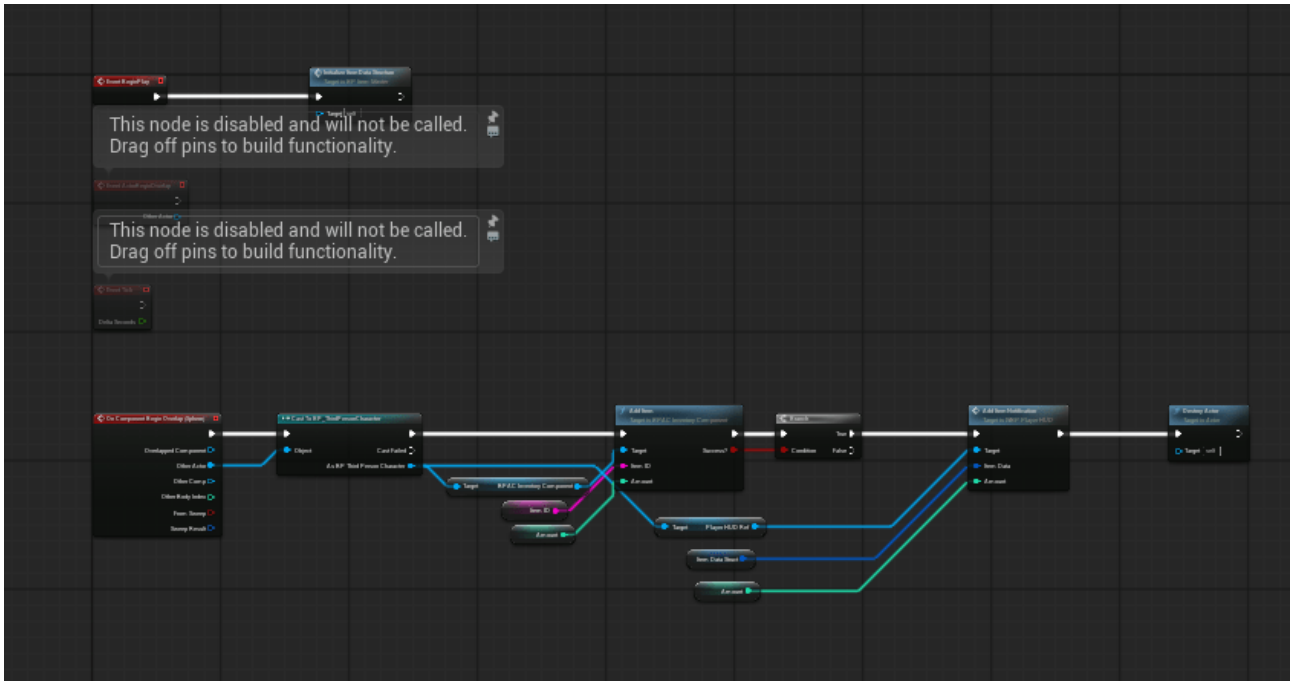


Рисунок Б.6 – BPAC_InventoryComponent

ДОДАТОК В
ІНТЕРФЕЙСИ

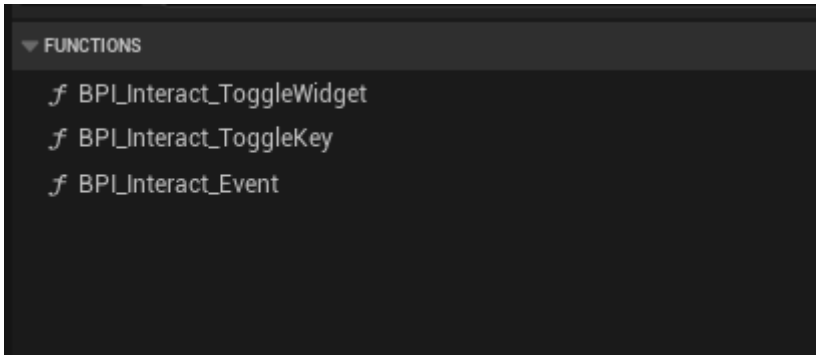


Рисунок В.1 – BPI_Interact

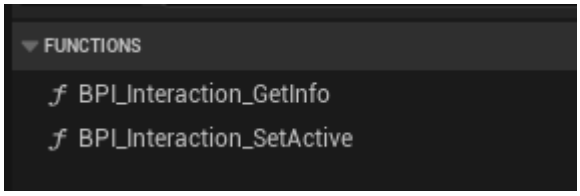


Рисунок В.2 – BPI_Interaction

ДОДАТОК Г
СЛАЙДИ ПРЕЗЕНТАЦІЇ

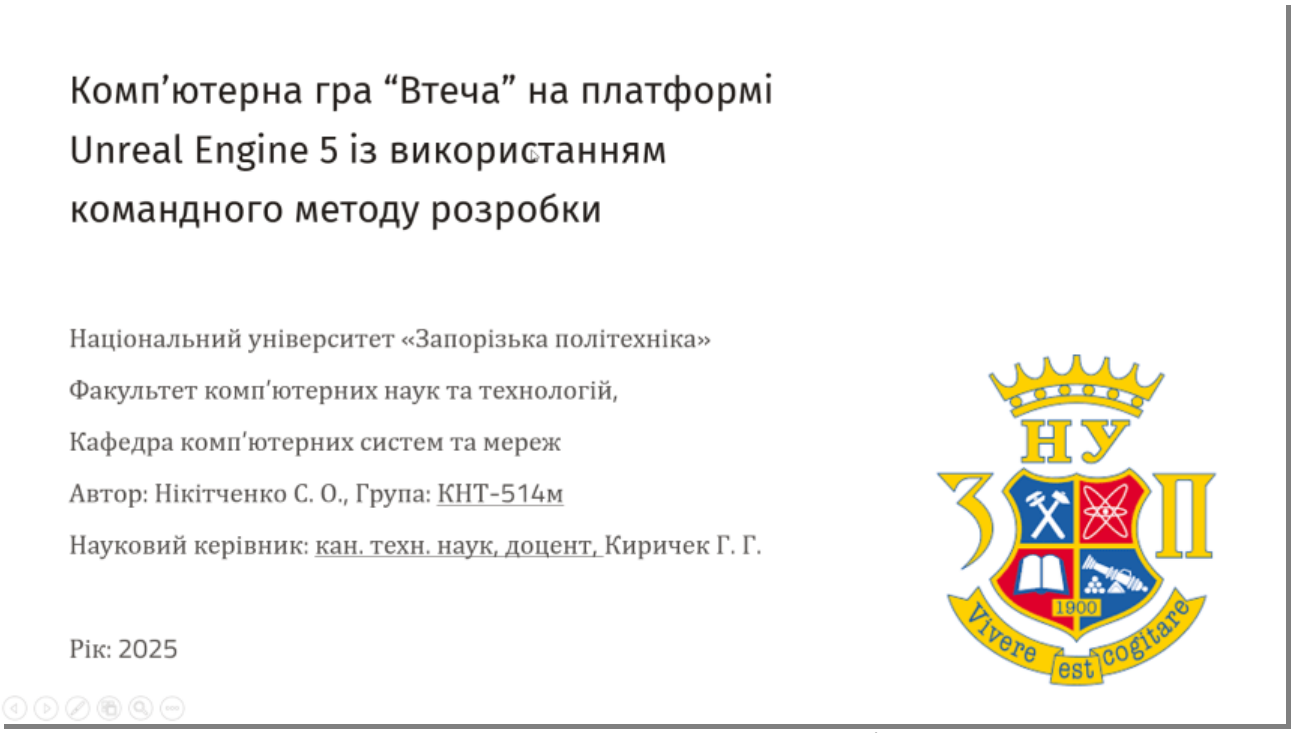


Рисунок Г.1 – Слайд презентації 1

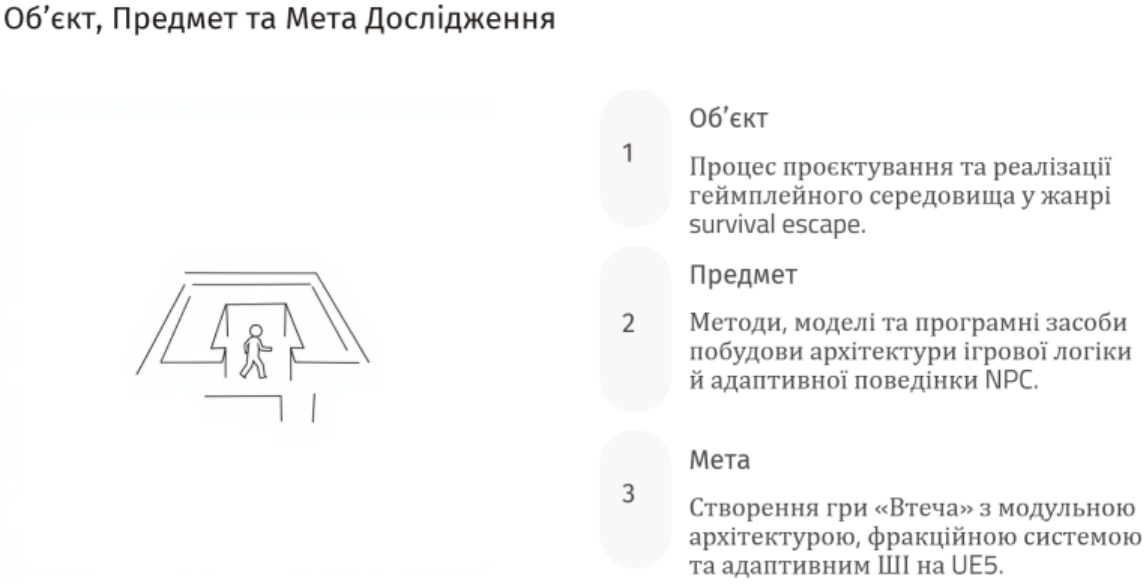


Рисунок Г.2 – Слайд презентації 2

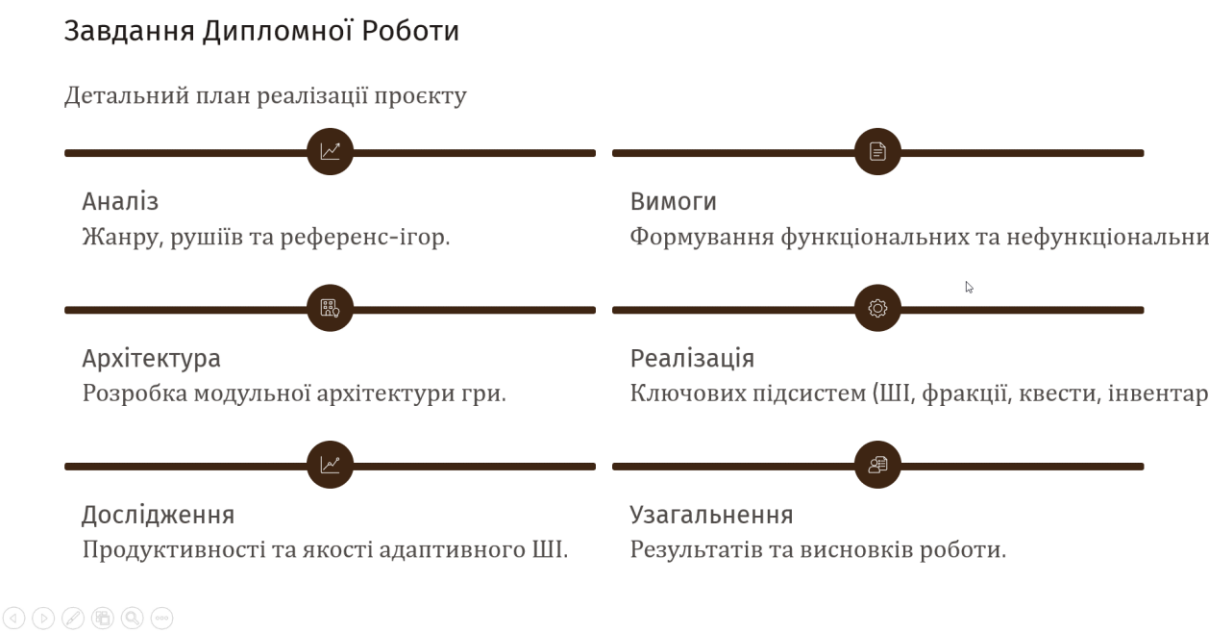


Рисунок Г.3 – Слайд презентації 3

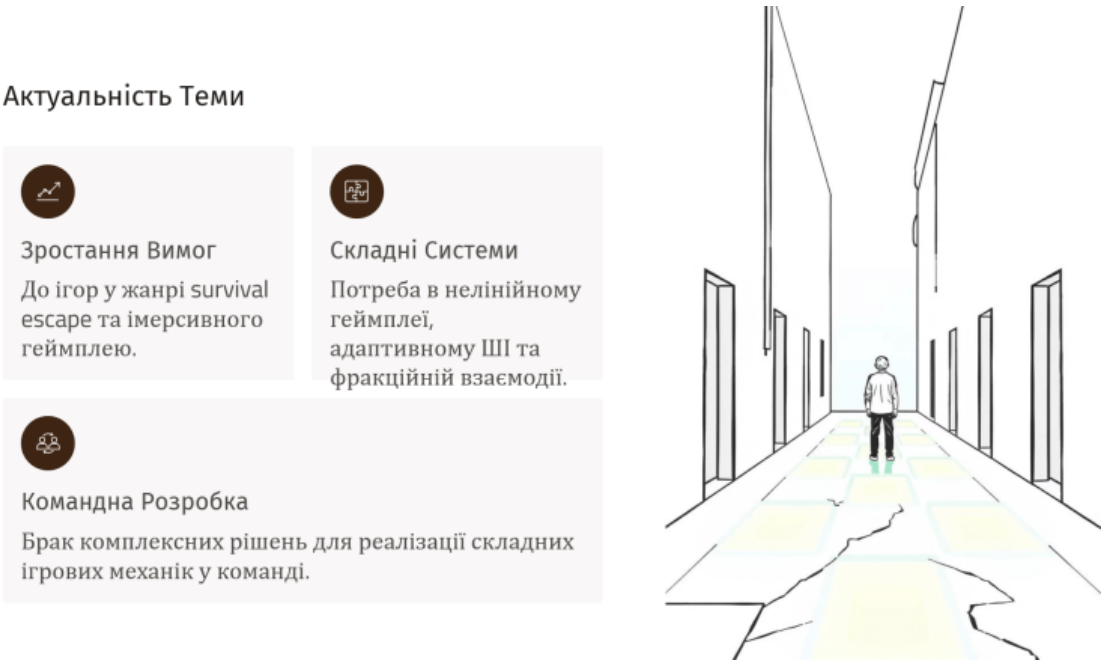
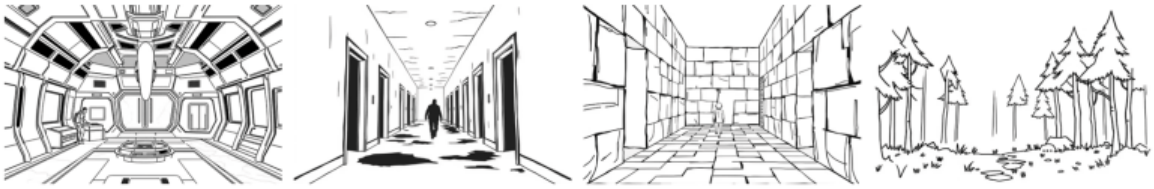


Рисунок Г.4 – Слайд презентації 4

Жанр Survival Escape та Референс-Ігри

Ключові особливості та натхнення



Жанр характеризується обмеженим простором, акцентом на **втечі та прихованості**, створюючи **високу напругу**. Аналіз референс-ігор, таких як **Alien: Isolation** (адаптивний ШІ), **Outlast** (прихованість), **Amnesia** (психологічний жах) та **The Forest** (виживання, крафт), дозволив виділити важливі аспекти для розробки ШІ та геймплею. Деталі аналізу представлені у роботі.

Рисунок Г.5 – Слайд презентації 5

Аналіз Ігрових Рушіїв: Чому Unreal Engine 5?

Вибір Рушія

Порівняння **Unity**, **UE5**, **Godot** за графікою, інструментами ШІ та масштабованістю. UE5 визнано оптимальним для «Втечі» завдяки його передовим можливостям.

Lumen & Nanite
Реалістична графіка та оптимізація.

Behavior Tree & EQS
Передові інструменти для адаптивного ШІ.

Blueprint + C++
Гнучка розробка логіки та продуктивність.

Масштабованість
Для майбутнього розвитку проєкту.

Детальний аналіз та порівняння рушіїв наведено у теоретичній частині роботи.

Рисунок Г.6 – Слайд презентації 6

Технологічний Стек Розробки

Інструменти та технології, що використовувались для створення гри

Unreal Engine 5
Основна платформа для розробки.

C++
Мова програмування для базової логіки.

Blueprint
Візуальне програмування для швидкої реалізації функціоналу.

Visual Studio
Інтегроване середовище розробки.

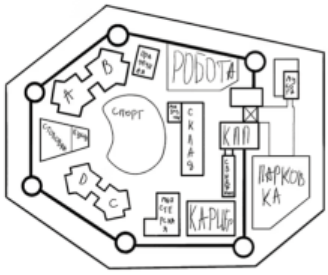
Git
Система контролю версій для командної роботи.

DataTables, Gameplay Tags, EQS, Behavior Tree
Додаткові інструменти для ігрових систем та ШІ.

Рисунок Г.7 – Слайд презентації 7

Концепція Гри «Втеча»

Занурення у світ гри та її механіки



Сеттинг
Вас відправили до тюрьми без причіни

Роль Гравця
Виживший, що шукає шлях до свободи, уникаючи небезпек та ворожих фракцій.

Геймплей
Дослідження, прихованість, взаємодія з NPC та фракціями, квести, стратегічна втеча.

Нелінійність
Кілька сценаріїв проходження та різноманітні фракційні моделі впливають на сюжет.

Рисунок Г.8 – Слайд презентації 8

Рисунок Г.9 – Слайд презентації 9

Рисунок Г.10 – Слайд презентації 10

Рисунок Г.11 – Слайд презентації 11

Рисунок Г.12 – Слайд презентації 12

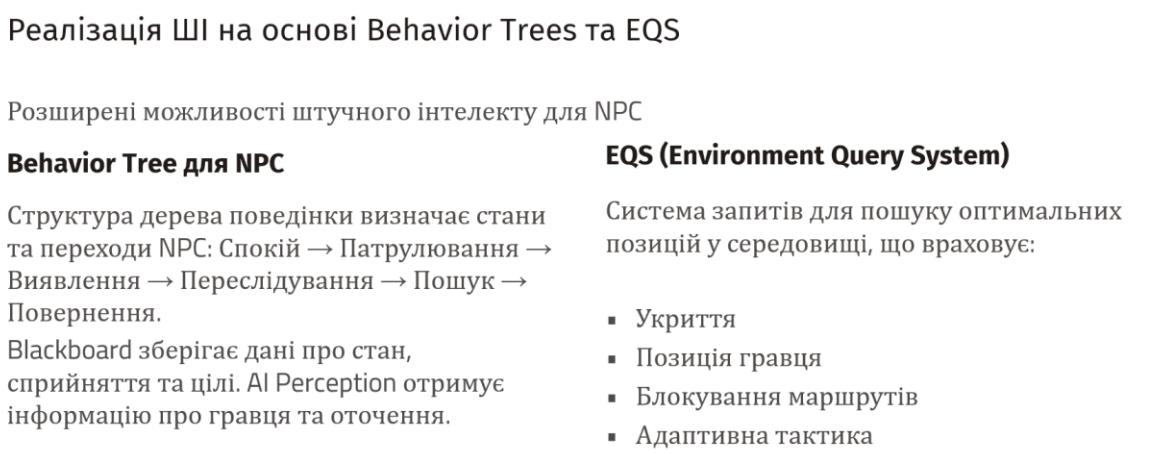


Рисунок Г.13 – Слайд презентації 13

Рисунок Г.14 – Слайд презентації 14

Тестове Середовище та Апаратна Конфігурація

Специфікація системи розробки та тестування

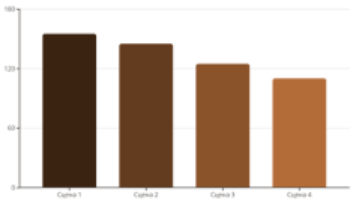
CPU	Intel Core i7-12700K
GPU	NVIDIA GeForce RTX 3080
RAM	32GB DDR4
Накопичувач	NVMe SSD (1TB)
ОС	Windows 11 Pro
Unreal Engine	5.3
Роздільна здатність	2560x1440
Налаштування графіки	High (Custom)

Тестування проводилось також на конфігурації з GPU NVIDIA GeForce RTX 2060 для перевірки сумісності та продуктивності на різних рівнях обладнання.

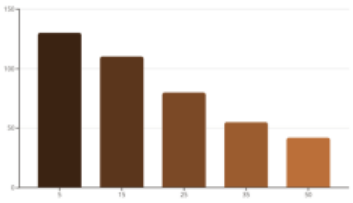
Рисунок Г.15 – Слайд презентації 15

Результати Тестування

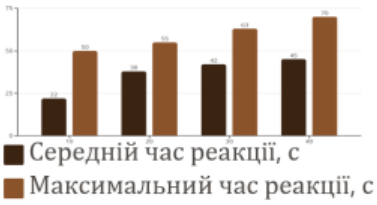
Графіки та аналіз продуктивності



Графік залежності середнього FPS від типу ігрової сцени



Графік залежності часу реакції ШІ від кількості NPC



Результати вимірювання часу реакції ШІ

Рисунок Г.16 – Слайд презентації 16

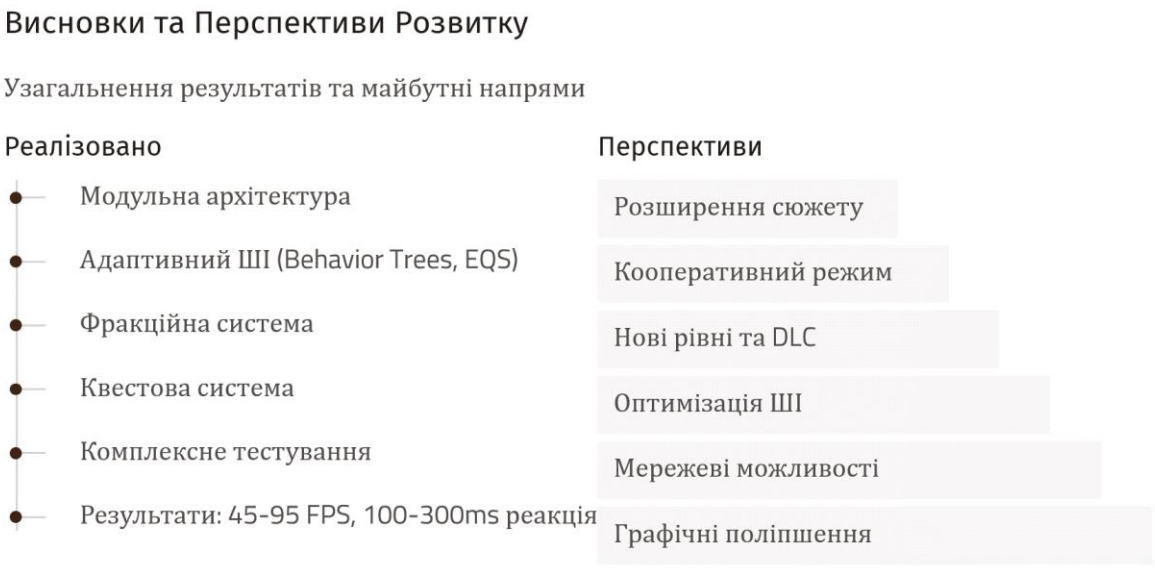


Рисунок Г.17 – Слайд презентації 17

Рисунок Г.18 – Слайд презентації 18