Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка комп’ютерної гри на ігровому рушії Unreal Engine 5.

Виконав: студент IV курсу, групи СНс-41
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

_____ Валовий В.А
(підпис) (прізвище та ініціали)

Керівник _____ Литвиненко Я.В.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _червня_ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)
за спеціальністю 122 Комп’ютерні науки
(шифр і назва спеціальності)
Студенту Валовий Вадим Андрійович
(прізвище, ім’я, по батькові)
1. Тема роботи Розробка комп’ютерної гри на ігровому рушії Unreal Engine 5.

Керівник роботи Литвиненко Ярослав Володимирович,доктор технічних наук,професор кафедри комп’ютерних наук
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом ректора від « *14* » *травня* 2026 року № *4/9-237*
2. Термін подання студентом завершеної роботи 22 червня 2026 р.
3. Вихідні дані до роботи Технічне завдання,функціональні вимоги до ігрових механік, вимоги до системи AI, програмне забезпечення, графічні та звукові ресурси проєкту, науково-методичні матеріали.
4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1 Аналіз предметної області та обґрунтування вибору технологій розробки. 2 Проектування архітектури та логіки ігрового застосунку. 3 Програмна реалізація, тестування та оптимізація гри.

4. Безпека життєдіяльності, основи охорони праці. Висновки. Перелік джерел
1 Актуальність безпеки життєдіяльності. 2 Аналіз умов праці розробника ігрових застосунків. 3 Санітарно-гігієнічні вимоги до виробничих приміщень. 4 Ергономічні вимоги до організації робочого місця. 5 Організація режимів праці. 6 Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень, слайдів)
Слайди презентації та діаграми процесів

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Мариненко С.Ю.		

7. Дата видачі завдання 26 січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	26.01.2026	
2.	Підбір та опрацювання літературних джерел по темі кваліфікаційної роботи	27.01.2026-16.02.2026	
3.	Виконання дослідження щодо	17.02.2026-10.05.2026	
	Розроблення комп'ютерної гри на ігровому рушіїї Unreal Engine 5		
4.	Оформлення розділу 1 Аналіз предметної області та геймлизайну	11.05.2026-17.05.2026	
5.	Оформлення розділу 2 Проєктна частина	18.05.2026-24.05.2026	
6.	Оформлення розділу 3 Реалізація та тестування ігрових систем	25.05.2026-31.05.2026	
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	01.06.2026-08.06.2026	
8.	Виконання завдання до підрозділу «Основи охорони	01.06.2026-08.06.2026	
	праці»		
9.	Оформлення кваліфікаційної роботи	09.06.2026-11.06.2026	
10.	Нормоконтроль	12.06.2026-15.06.2026	
11.	Перевірка на плагіат	16.06.2026	
12.	Попередній захист кваліфікаційної роботи	18.06.2026	
13.	Захист кваліфікаційної роботи	22.06.2026	

Студент _____ Валовий В.А
(підпис) (прізвище та ініціали)

Керівник роботи _____ Литвиненко Я.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка комп’ютерної гри на ігровому рушії Unreal Engine 5. Кваліфікаційна робота освітнього ступеня «Бакалавр» Валовий Вадим Андрійович Тернопільський національний технічний університет імені Івана Пулюя, факультет комп’ютерно-інформаційних систем і програмної інженерії, кафедра комп’ютерних наук, група СНс-41 Тернопіль, 2026 Обсяг роботи – [71] стор. , [35] рис., [15] табл., [2] додатки, [42] джерел.

Ключові слова: Uureal Engine 5, Survival, штучний інтелект, блюпрінти, скінчений автомат (FSM),гейм дизайн, модель MDA, крафт, оптимізація.

Роботу присвячено розробці тривимірної Survival-гри «Feral Island» на базі Unreal Engine 5. Проведено аналіз ринкових аналогів (The Forest, Ark: Survival Evolved), виявлено їхні технічні недоліки в оптимізації, системах будівництва та поведінці штучного інтелекту.

Сформульовано вимоги до проєкту та обґрунтовано вибір рушія UE5 з технологіями Nanite й Lumen. Логіку гри реалізовано за допомогою системи візуального скриптингу Blueprints, а проєктування геймплею виконано за методологічною моделлю MDA.

Спроєктовано архітектуру ігрових систем та інтерфейс користувача (HUD). Реалізовано модулі життєдіяльності персонажа (здоров'я, голод, спрага, витривалість), систему інвентарю й крафту, а також штучний інтелект ворогів на базі скінченних автоматів (FSM) для патрулювання та атаки.

Проведено тестування механік, оцінено стабільність FPS та споживання системних ресурсів. Отримано працездатний прототип 3D-гри, придатний для подальшого масштабування до повноцінного комерційного проєкту.

ANNOTATION

Development of a computer game on the Unreal Engine 5 game engine. – Bachelor's Qualification Thesis. Valovyi Vadym Andriiovych. Ternopil Ivan Puluj National Technical University, Faculty of Computer and Information Systems and Software Engineering, Department of Computer Science, Group SNs-41. Ternopil, 2026. Thesis volume: [71] pages, [35] figures, [15] tables, [2] appendices, [42] sources.

Keywords: Unreal Engine 5, Survival, artificial intelligence, blueprints, Finite State Machine (FSM), game design, MDA framework, crafting, optimization..

The thesis is devoted to the development of a 3D survival game "Feral Island" based on Unreal Engine 5. A technical analysis of market analogs (The Forest, Ark: Survival Evolved) was conducted, identifying issues in performance, construction, and AI behavior.

Project requirements were formulated, justifying the choice of UE5 with Nanite and Lumen technologies. Game logic was implemented via the Blueprints visual scripting system, and gameplay was designed using the MDA framework.

The system architecture and user interface (HUD) were developed. Core modules include character survival metrics (health, hunger, thirst, stamina), an inventory and crafting system, and enemy AI based on Finite State Machines (FSM) for patrolling and attacking.

Functional testing was performed, evaluating FPS stability and hardware resource consumption. A working 3D game prototype was delivered, suitable for further scaling into a commercial project.

ПЕРЕЛІК СКОРОЧЕНЬ

AI (англ. Artificial Intelligence) – штучний інтелект.

BP (англ. Blueprint) – система візуального скриптингу (програмування) в ігровому рушії Unreal Engine.

CPU (англ. Central Processing Unit) – центральний процесор (ЦП).

HUD (англ. Heads-Up Display) – частина графічного інтерфейсу користувача, що відображає поточні показники гравця під час геймплею.

UI (англ. User Interface) – інтерфейс користувача (графічна оболонка гри).

UE5 (англ. *Unreal Engine 5*) – багатоплатформний ігровий рушій п'ятого покоління розробки компанії Epic Games.

ОЗП (англ. *Random Access Memory*) – оперативний запам’ятовувальний пристрій, призначений для тимчасового зберігання даних і команд, необхідних процесору для виконання операцій у поточний момент часу.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасна індустрія комп'ютерних ігор є одним із найбільш динамічних секторів цифрової економіки та комп'ютерних наук, що стимулює розвиток технологій тривимірного моделювання, штучного інтелекту та оптимізації обчислень[12]. Серед різноманіття жанрів особливу популярність утримують ігри в стилі виживання (Survival), які вимагають від розробників створення складних, взаємопов'язаних систем: динамічної зміни стану персонажа, менеджменту ресурсів, процедурної генерації або деталізації оточення, а також адаптивної поведінки ворогів.

Вихід ігрового рушія Unreal Engine 5 (UE5) здійснив технологічний прорив завдяки інтеграції систем віртуалізованої геометрії Nanite та динамічного глобального освітлення Lumen[21]. Це дозволило створювати ігри кінематографічної якості навіть невеликим командам чи індивідуальним розробникам. Проте, реалізація комплексних геймплейних механік та забезпечення високої продуктивності на базі UE5 потребує чіткого архітектурного проєктування, ефективного використання інструментів візуального скриптингу Blueprints та математичного моделювання систем штучного інтелекту. Необхідність розв'язання інженерних задач оптимізації та побудови гнучкої логіки ігрових систем у жанрі Survival визначає актуальність теми кваліфікаційної роботи

Мета роботи – розробка тривимірної комп’ютерної гри у жанрі виживання під робочою назвою «Feral Island» на базі ігрового рушія Unreal Engine 5 із реалізацією механік життєдіяльності персонажа, інвентарю, крафту ресурсів та штучного інтелекту супротивників.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– Проаналізувати стан індустрії відеоігор у заданому жанрі та технічні особливості ігор-аналогів.

– Обґрунтувати вибір інструментарію розробки та проаналізувати переваги архітектури Unreal Engine 5.

– Спроєктувати концепцію проєкту та взаємодію ігрових систем за допомогою геймдизайнерської моделі MDA (Mechanics, Dynamics, Aesthetics)[14].

– Розробити архітектуру програмних модулів гри та спроєктувати графічний інтерфейс користувача (HUD).

– Реалізувати алгоритми поведінки штучного інтелекту (AI) ворожих персонажів на основі скінченних автоматів (FSM).

– Провести функціональне тестування прототипу гри та оцінити стабільність показників продуктивності (FPS, використання ОЗП і ЦП).

Об’єкт дослідження – процес розробки тривимірних комп’ютерних ігор у жанрі Survival.

Предмет дослідження – технології, інструменти та методи ігрового рушія Unreal Engine 5 для моделювання геймплейних механік, систем штучного інтелекту та оптимізації графіки.

Методи дослідження. У роботі використано метод системного аналізу для дослідження існуючих аналогів; геймдизайнерське моделювання (MDA) для формування ігрового досвіду; об'єктно-орієнтоване проєктування та візуальне програмування (Blueprints) для реалізації програмної логіки; теорію скінченних автоматів (FSM) для розробки ШІ; емпіричні методи тестування для аналізу продуктивності.

Практичне значення отриманих результатів. Полягає у створенні працездатного, оптимізованого прототипу тривимірної гри «Feral Island» з інтегрованими базовими системами життєдіяльності, інвентарю та AI. Отримані програмні та архітектурні рішення можуть слугувати основою для розробки повноцінних комерційних інді-проєктів або як технологічна база для подальших досліджень у сфері геймдеву.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ГЕЙМДИЗАЙНУ

1.1 Аналіз сучасного ринку ігор у жанрі виживання.

Сучасний ринок тривимірних інтерактивних застосунків демонструє стійку тенденцію до популяризації ігор у жанрі виживання[9] . З інженерної точки зору архітектура програмного забезпечення проєктів цього жанру є складною багаторівневою системою, яка функціонує в режимі реального часу.

Ключовою особливістю таких систем є необхідність безперервного моделювання взаємодії гравця з динамічним агресивним середовищем за допомогою взаємопов'язаних підсистем: кінцевих автоматів стану персонажа, реляційних структур даних інвентарю, алгоритмів просторового розрахунку колізій під час будівництва та систем прийняття рішень AI.

Основу ігрового процесу у жанрі Survival можна описати у вигляді замкненого циклу: «збір базових ресурсів ,моніторинг та підтримання вітальних показників, протидія загрозам середовища, крафт спорядження та зведення фортифікацій, отримання доступу до складніших технологій.

Для проектування оптимальної архітектури власного програмного продукту – тривимірної відеогри про виживання на острові – необхідно провести системний аналіз наявних комерційних аналогів[1]. Огляд дозволить виявити їхні архітектурні переваги, програмні обмеження та сформувати перелік функціональних вимог до розроблюваної системи.

За основними технічними та геймплейними критеріями було відібрано три найбільш релевантні комерційні проєкти: The Forest та Ark: Survival Evolved .

1.1.1 Аналіз відеогри The Forest.

Проєкт The Forest є класичним представником жанру survival-horror, де дія розгортається у відкритому ізольованому середовищі . Гра реалізована на базі ігрового рушія Unity[34].

– Система життєдіяльності: Програмний модуль відстежує декілька ключових параметрів персонажа: здоров'я, енергію , рівень голоду та спраги. Розрахунок показників виконується декрементними алгоритмами, що залежать від ігрового часу та фізичної активності .

– Механіка будівництва: Реалізовано напівавтоматичну систему розміщення модульних об'єктів. Програма використовує так звані «примари» (blueprints об'єктів із напівпрозорими матеріалами), які гравець розміщує у просторі. Після фіксації позиції створюється пуста сутність-контейнер, яка потребує заповнення необхідними ресурсами з інвентарю користувача для фінальної ініціалізації тривимірної сітки.

– Модель AI: AI ворогів побудований на складних деревах поведінки з елементами нечіткої логіки. Ворожі NPC здатні оцінювати чисельність гравців, рівень своєї загрози, виявляти страх, відступати або викликати підкріплення.

– Архітектурні недоліки: Проєкт страждає від обмежень однопотокової обробки фізики рушія Unity ранніх версій, що призводить до падіння частоти кадрівза наявності великої кількості зведених будівельних об'єктів у зоні видимості через складний прорахунок обчислення динамічних колізій.

1.1.2 Аналіз відеогри Ark: Survival Evolved.

Проєкт Ark: Survival Evolved є масштабною багатокористувацькою survival-системою, розробленою на модифікованій версії рушія Unreal Engine 4..

– Система життєдіяльності: Містить розширений набір параметрів, включаючи температуру тіла, оглушення, вагу інвентарю та мікронутриенти. Перевантаження інвентарю безпосередньо впливає на швидкість переміщення персонажа через лінійні математичні коефіцієнти[35] рендерингу складних тривимірних сцен використовуються базові алгоритми комп'ютерної графіки.

– Механіка будівництва: Застосовано систему жорсткої прив'язки до просторової сітки за допомогою сокетів. Будівельні елементи (фундаменти,

стіни, дахи) містять спеціальні координатні точки колізій. При наближенні нового елемента алгоритм автоматично притягує його до найближчого вільного сокета існуючої конструкції.

– Модель AI: AI фауни (динозаврів) використовує кінцеві автомати та простішу логіку агресії, прив'язану до радіуса виявлення об'єкта. Ворожі сутності діляться на класи: травоїдні (втікають при пошкодженні) та хижаки (атакують при входженні гравця в зону видимості). Після смерті AI-сутності її об'єкт деструктуризується, а на його місці генерується контейнер із ресурсами таких як шкіра, м'ясо.

– Архітектурні недоліки: Низький рівень оптимізації коду, значні витрати оперативної пам'яті на збереження станів світу, тривалий час завантаження через неоптимальну структуру пакування ігрових архівів.

1.2 Порівняльний аналіз ігрових рушіїв для розробки 3D-ігор

Проєктування та програмна реалізація сучасного тривимірного ігрового застосунку у жанрі виживання висуває критичні вимоги до вибору базового програмного каркаса – ігрового рушія. Проєкт такого типу характеризується високою щільністю обчислювальних операцій, які виконуються паралельно в реальному часі: розрахунок динамічних колізій при модульному будівництві, обробка станів кінцевого автомата персонажа, трасування променів для підсистеми штучного інтелекту ворожих сутностей , а також потокове завантаження геометрії та текстур відкритого тривимірного простору.

У межах дослідження було розглянуто три потенційні інструментальні платформи: Godot Engine, Unity та Unreal Engine 5 (UE5).

– Godot Engine є легковагим рішенням із відкритим сирцевим кодом, проте його графічний конвеєр та підсистема обробки великих відкритих просторів не володіють достатнім рівнем оптимізації для високонавантажених 3D-проєктів із фотореалістичною графікою.

– Unity пропонує гнучку компонентну архітектуру, але реалізація складних графічних ефектів та обробка масивних тривимірних сцен вимагає глибокої ручної оптимізації через написання користувацьких менеджерів пам’яті та налаштування конвеєрів рендерингу , що значно збільшує терміни розробки.

На основі критеріїв архітектурної зрілості, нативності інструментів оптимізації та технологічного стека для розробки прототипу було обрано Unreal Engine 5. Нижче наведено розгорнуте інженерно-технічне обґрунтування цього вибору через аналіз ключових підсистем рушія.

1.2.1 Технологія віртуалізованої геометрії Nanite

Одним із головних архітектурних викликів під час розробки тривимірної гри про виживання на острові є оптимізація відображення об'єктів навколишнього середовища такого як густа рослинність, дерева, скелі, високодеталізована модель тигра[20]. У традиційних рушіях для цього використовується ручне або напівавтоматичне створення рівнів деталізації, що призводить до витрат часу та різких візуальних перескоків при зміні дистанції камери[20].

Впроваджена в UE5 підсистема Nanite кардинально змінює конвеєр обробки геометрії:

– Кластеризація трикутників: На етапі імпорту тривимірні сітки аналізуються та розбиваються на ієрархічні кластери трикутників[20].

– Динамічний вибір деталізації: Під час рендерингу GPU за допомогою обчислювальних шейдерів оцінює розмір кластера на екрані й у реальному часі стрімить лише той рівень деталізації, який відповідає розміру пікселя. Це мінімізує кількість надлишкових полігонів .

– Програмна растеризація:На мікрорівні Nanite використовує надшвидку програмну растеризацію, яка працює в рази швидше за стандартний апаратний конвеєр фіксованих функцій GPU, повністю нівелюючи обмеження за кількістю полігонів у сцені.

Це дозволяє інтегрувати високополігональні асети без ризику падіння FPS та без необхідності ручної оптимізації LOD-ів.

1.2.2 Підсистема динамічного глобального освітлення Lumen

Для реалізації повноцінного занурення у survival-іграх критично важливою є динамічна зміна часу доби. У рушіях попередніх поколінь реалістичне освітлення досягалося шляхом запікання статичних карт світла що унеможливлювало їх зміну в реальному часі та вимагало гігантських обсягів в VRAM.

Технологія Lumen в UE5 є повністю динамічним рішенням для розрахунку глобального освітлення та дзеркальних відображень[21] :

– Гібридне трасування променів: Lumen комбінує програмне трасування променів по полях підписаних відстаней із апаратним трасуванням таким як Hardware Ray Tracing, якщо його підтримує відеокарта.

– Вокселізація сцени: Рушій автоматично генерує спрощене воксельне представлення навколишнього середовища в радіусі навколо камери гравця, за рахунок чого обчислення вторинного відбиття світла ,наприклад, відбиття сонячного світла від піску на стіни збудованого гравцем дерев'яного будинку,відбувається з мінімальним навантаженням на обчислювальні блоки GPU.

Це дозволяє розробити реалістичну систему освітлення острова з повною підтримкою динамічних джерел світла сонце, місяць, факели, багаття без накладних витрат на передчасні обчислення.

1.2.3 Система візуального програмування Blueprints Visual Scripting

Архітектура Unreal Engine 5 базується на об'єктно-орієнтованому підході з використанням мови C++ та системи візуального скриптингу BP. Система BP у

UE5 не є простою надбудовою, а інтегрована у внутрішню систему рефлексії рушія[23].

– Об'єктна модель та спадкування: Кожен ігровий об'єкт є похідним від базового класу AActor. На основі акторів створюються спеціалізовані сутності, такі як ACharacter та AGameModeBase[32] . Спадкування в BP дозволяє створити базовий клас ресурсу BP_BaseResource, від якого успадковуються класи BP_Tree чи BP_Rock, перевизначаючи лише масив генерованого луту.

– Продуктивність віртуальної машини: Блюпринти компілюються у байт-код, який виконується спеціалізованою віртуальною машиною рушія. Для високонавантаженої математики ,наприклад, циклів перевірки тисяч об'єктів, краще використовувати C++, проте для високорівневої ігрової логіки нашого проєкту ,зміна показників HUD, менеджмент інвентарю за подіями Event, трасування для будівництва, продуктивності систем BP є більш ніж достатньо. Це забезпечує високу швидкість розробки та гнучкість налагодження через візуальне відстеження потоку виконання даних. Мінімізація використання щокадрових обчислень на користь подієво-орієнтованих функцій дозволяє повністю ліквідувати накладні витрати на інтерпретацію байт-коду та зберегти високу частоту кадрів.

Отже, ігровий рушій UE5 володіє найбільш збалансованим, технологічно випереджаючим та архітектурно цілісним інструментарієм для розробки тривимірної відеогри у жанрі виживання. Використання Nanite, Lumen та системи Blueprints дозволяє нівелювати більшість архітектурних помилок та оптимізаційних обмежень, виявлених під час аналізу ринкових аналогів у підрозділі.

1.3 Концептуальне проєктування та розробка геймдизайн-документа ігрової системи

У програмній інженерії інтерактивних додатків геймдизайн-документ виконує роль розширеного технічного завдання та специфікації функціональних

вимог до програмного комплексу. Він фіксує архітектурні рамки, правила функціонування ігрової логіки, структури даних та алгоритми взаємодії користувача з віртуальним середовищем. Розробка концепції та ГДД є обов'язковим етапом передінженерного аналізу, що дозволяє формалізувати вимоги до майбутніх класів, об'єктів та підсистем рушія UE5.

У межах цього проєкту розробляється тривимірна відеогра з робочою назвою «Feral Island». Концептуальне проєктування системи базується на методологічній моделі MDA, Mechanics, Dynamics, Aesthetics[14] – механіки, динаміка, естетика, що дозволяє чітко розділити програмні алгоритми від емоційного досвіду користувача.

1.3.1 Високорівнева концепція проєкту

– Жанр: Тривимірний симулятор виживання від першої особи з елементами будівництва та розгалуженим штучним інтелектом екосистеми.

– Платформа: ПК під керуванням ОС Windows.

– Цільова авдиторія: Користувачі, які віддають перевагу складним технічним механікам менеджменту ресурсів та динамічному протистоянню з агресивним AI.

– Сетинг: Ізольований тропічний острів із вираженою вертикальністю ландшафту, динамічною зміною погодних умов та часу доби, заселений ворожими фауністичними сутностями.

1.3.2 Формалізація базового ігрового циклу

Математично та логічно функціонування розроблюваного програмного забезпечення описується замкненим орієнтованим графом станів користувача, який формує базовий ігровий цикл. Модель базового циклу системи подано на рисунку 1.1. Наведена схема деталізує циклічну послідовність дій гравця, що базується на постійному зборі ресурсів, створенні засобів виживання та протидії

загрозам зовнішнього середовища. Опис цих переходів у вигляді орієнтованого графа дозволяє гнучко масштабувати ігрову логіку при додаванні нових механік або ускладненні поведінки штучного інтелекту.

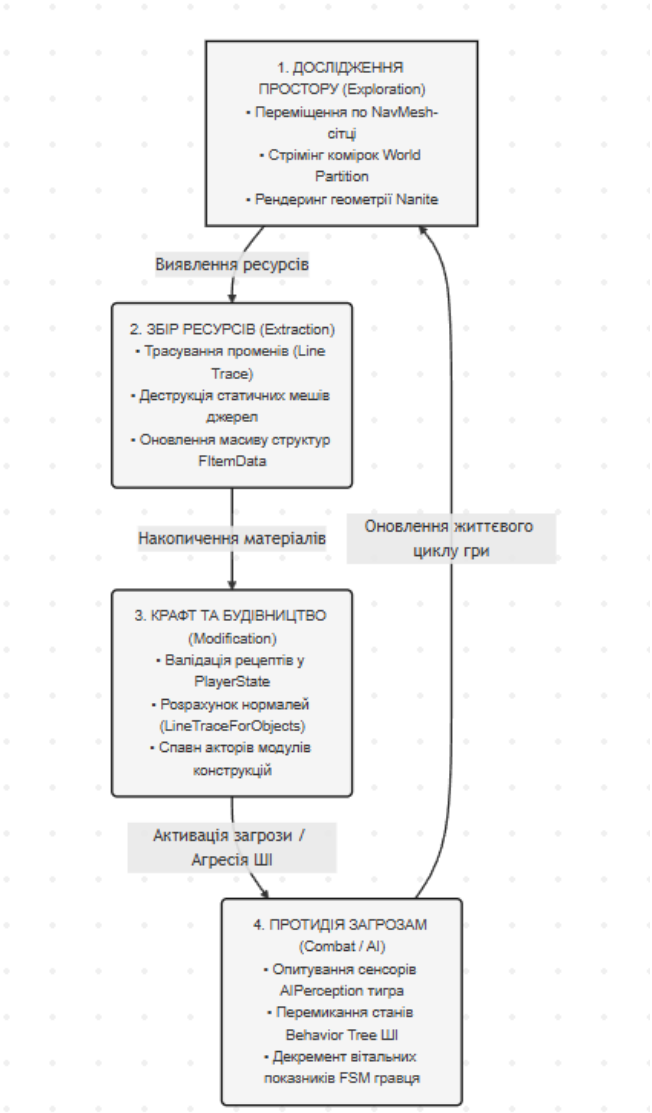

Рисунок 1.1 – Модель базового циклу системи

Базовий цикл складається з чотирьох послідовних фаз обробки даних:

– Дослідження та сканування простору: Актор гравця переміщується по NavMesh-сітці острова, ініціюючи рендеринг геометрії Nanite та зчитуючи координати точок спавну ресурсів.

– Екстракція даних : Взаємодія з об'єктами класів BP_BaseResource (дерева, каміння), руйнування їхньої геометрії та інкрементне додавання ідентифікаторів предметів до динамічного масиву інвентарю.

– Модифікація станів: Обробка рецептів крафту через порівняння масивів даних. Виклик функцій просторового трасування для генерації нових статичних об'єктів .

– Протидія загрозам: Обробка подій колізій між актором гравця та актором ворожого AI тигра, розрахунок декременту здоров'я персонажа та активація кінцевого автомату захисту.

1.3.3 Специфікація функціональних підсистем

Згідно з концептуальним проєктуванням, програмний комплекс поділяється на чотири автономні, але взаємопов'язані підсистеми, логіка яких детально описується в ГДД.

Підсистема моніторингу вітальних показників:

Керування станом персонажа реалізується через кінцевий автомат, який оперує чотирьома неперервними числовими змінними типу Float у діапазоні [0.0; 100.0]:

– Health – базовий параметр життєздатності об'єкта. При Health <= 0.0 викликається подія Event Death, що блокує введення з PlayerController та запускає екран перезавантаження.

– Stamina – динамічний ресурс. Коли прапорець IsSprinting дорівнює True, запускається щокадровий декремент стаміни. При Stamina <= 0.0 швидкість переміщення штучно обмежується до базової.

– Hunger та Thirst – параметри, що зменшуються лінійно з часом за допомогою таймерів.

Логічний взаємозв'язок параметрів визначається умовою: якщо Hunger == 0.0 або Thirst == 0.0, система ініціює циклічний виклик функції ApplyDamage, зменшуючи показник Health на 5.0 одиниць за інтервал часу t = 2.0сек.

Підсистема менеджменту інвентарю та крафту:

В основі підсистеми лежить об'єктно-орієнтована структура даних FItemData, яка містить такі поля: ItemID (Integer), ItemName (Text), ItemIcon (Texture2D), CurrentStack (Integer), MaxStack (Integer), ItemType (Enum).

– Логіка додавання: При взаємодії з дропом викликається функція AddItemToInventory. Алгоритм виконує ітераційний перебір масиву структур інвентарю, шукаючи збіг за ItemID. Якщо збіг знайдено і CurrentStack< MaxStack, виконується математичне додавання. Якщо комірки зайняті, створюється новий елемент масиву.

– Логіка крафту: Створюється двовимірний масив структур-рецептів. При виклику команди CraftItem підсистема перевіряє, чи є сумарна кількість наявних ItemID у масиві інвентарю більшою або рівною вимогам рецепта. За умови валідності, ініціюється віднімання ресурсів та додавання результуючого актора.

Підсистема модульного будівництва:

Механіка базується на динамічному просторовому моделюванні за допомогою математичного методу трасування променів у тривимірному просторі. За натисканням клавіші виклику будівельного меню ініціюється таймер, що працює з частотою кадрів та викликає функцію LineTraceForObjects.

– Вхідні дані трасування: Початкова точка – координати камери гравця GetPlayerCameraLocation, кінцева точка – вектор напрямку погляду камери, помножений на скаляр відстані будівництва 400 одиниць.

– Обробка зіткнення: При поверненні структури HitResult алгоритм зчитує вектор нормалі поверхні зіткнення та просторові координати. На основі цих даних розраховується просторова матриця трансформації для тимчасового напівпрозорого актора-прев'ю.

– Валідація та спавн: Алгоритм перевіряє колізію Capsule Overlap. Якщо прев'ю-об'єкт не перетинає заборонені зони чи AI-сутності, матеріал забарвлюється в зелений колізер. При натисканні лівої кнопки миші

викликається нода SpawnActorOfClass, яка заміщує прев'ю постійним статичним об'єктом будівлі із фізичною колізією типу BlockAll.

Розроблений геймдизайн-документ та концептуальні схеми взаємодії підсистем дозволяють перейти від абстрактного опису гри до безпосереднього інженерного проєктування архітектури класів, об'єктних зв'язків та математичних моделей, що складатимуть основу наступного розділу кваліфікаційної роботи.

1.4 Висновки до розділу 1

Проведено порівняльний аналіз сучасних ігрових рушіїв (Godot, Unity, Unreal Engine 5). Обґрунтовано інженерну доцільність вибору рушія Unreal Engine 5 завдяки нативності технологій Nanite (віртуалізована геометрія) та Lumen (глобальне освітлення), які дозволяють нівелювати проблеми продуктивності, виявлені в аналогах, а також використанню системи візуального скриптингу Blueprints для швидкого прототипування логіки.

РОЗДІЛ 2. ПРОЄКТНА ЧАСТИНА

2.1 Архітектурна модель та каркас ігрового застосунку

Проєкт тривимірної інтерактивної системи симулятора виживання будується на базі системного каркаса Unreal Gameplay Framework, що входить до складу рушія UE5. Вибір даного каркаса зумовлений його високою модульністю, нативною оптимізацією обчислювальних потоків та наявністю готових базових класів, які реалізують фундаментальну логіку клієнт-серверної або локальної взаємодії об’єктів у тривимірному просторі.

Архітектура розроблюваного програмного комплексу базується на декомпозиції керуючих та виконавчих сутностей. На відміну від монолітних архітектур, де логіка стану гри та обробка введення об’єднані, у даному проєкті реалізовано розподіл за засадами інверсії керування. Основними архітектурними компонентами каркаса системи є[6]:

– ASurvivalGameMode, похідний від AGameModeBase – визначає правила гри, ініціалізує класи для спавну, керує точками старту акторів та станами життєвого циклу сесії[32].

– ASurvivalGameState, похідний від AGameStateBase – відповідає за збереження стану поточної ігрової сесії. Наприклад, поточний час доби, стан погоди на острові, кількість активних ворожих AI-агентів[32].

– ASurvivalPlayerController, похідний від APlayerController – є інтерфейсом між фізичним пристроєм введення клавіатура, миша та логічним актором ігрового світу. Він обробляє низькорівневі переривання та транслює їх у високорівневі команди керування[32].

– ABP_SurvivalCharacter, похідний від ACharacter – безпосередній фізичний та логічний прояв гравця у просторі, який інкапсулює колізійну капсулу, тривимірний меш камери та модульні компоненти підсистем[32].

Схема взаємодії компонентів Gameplay Framework та потоків керування в межах базового життєвого циклу системи наведена на рисунку 2.1.

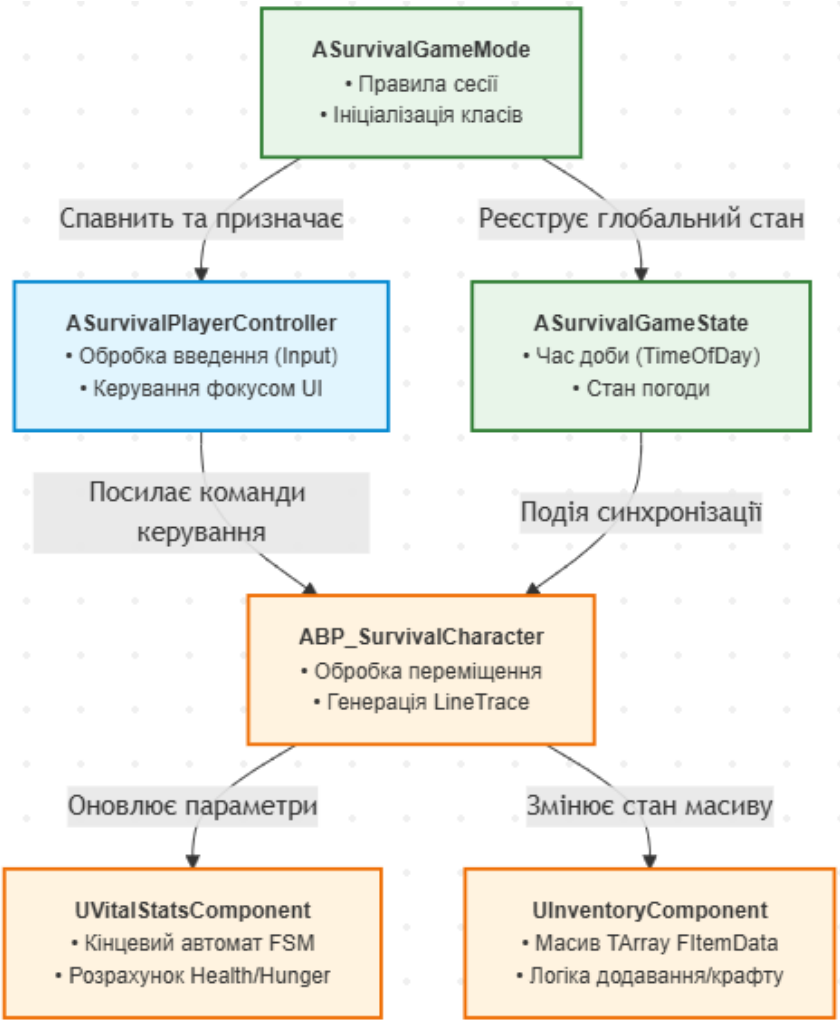

Рисунок 2.1 – Схема взаємодії компонентів Gameplay Framework

Аналіз представленої архітектури (див. рис. 2.1) показує, що актор персонажа не керує безпосередньо станом гри, а лише транслює свої просторові координати та запити на зміну даних. Поділ на ізольовані компоненти UInventoryComponent та UVitalStatsComponent дозволяє реалізувати паттерн «Композиція замість успадкування», що суттєво знижує когнітивну складність коду та оптимізує щокадровий перерахунок станів за допомогою внутрішніх таймерів рушія, зменшуючи навантаження на CPU.

2.2 Архітектурна модель та каркас ігрового застосунку

Для розробки складного програмного комплексу симулятора виживання критично важливо побудувати масштабовану ієрархію об’єктів. В основі ієрархії

лежить використання абстрактних базових класів так званих «Майстер-Blueprint-класів», які містять спільні змінні, параметри колізій та інтерфейсні функції, тоді як конкретні об’єкти наприклад, дерево певного виду або стіна будівлі є дочірніми класами Child Blueprints, що успадковують та перевизначають лише статичні параметри та візуальні меші[6].

Такий підхід повністю задовольняє принципам SOLID, зокрема принципу відкритості/закритості: базовий клас закритий для модифікації його ядра, але відкритий для розширення через створення нових дочірніх сутностей.

У проєкті виділено три основні гілки ієрархії класів:

– BP_MasterInteractable – базовий клас для всіх інтерактивних об’єктів острова .

– BP_MasterStructure – базовий клас для елементів модульного будівництва.

– BP_NPC_Base – базовий клас для підсистеми штучного інтелекту фауни.

Графічне представлення статичної ієрархії успадкування та композиції класів програмного комплексу наведено на рисунку 2.2.

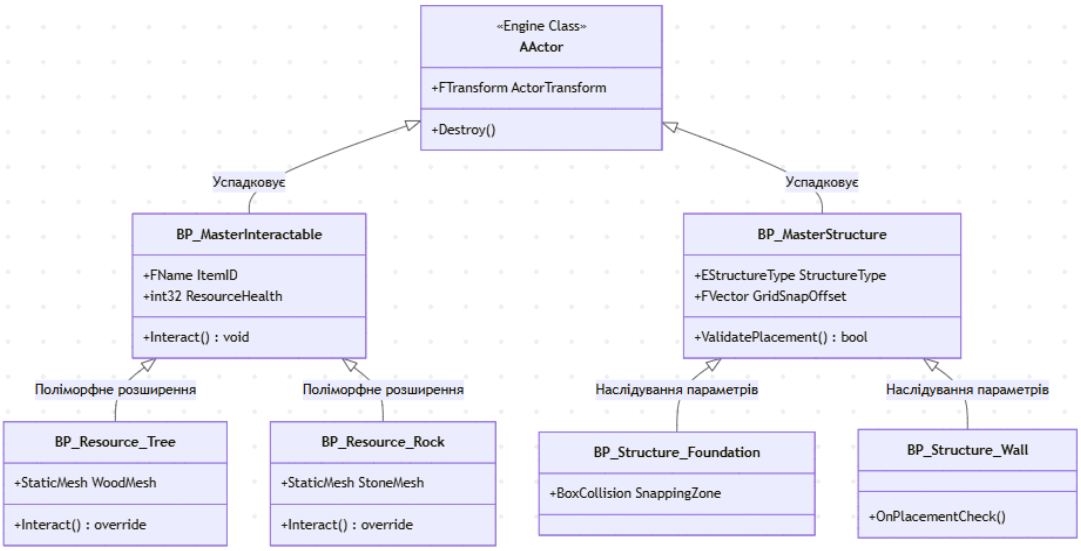

Рисунок 2.2 – Представлення статичної ієрархії успадкування та композиції класів

Перевага розробленої структури (див. рис. 2.2) полягає у використанні поліморфізму. Коли підсистема трасування променів персонажа реєструє зіткнення з об'єктом, вона виконує приведення типів Cast не до конкретного дерева чи каменя, а до абстрактного класу BP_MasterInteractable. Виклик функції Interact() автоматично запускає унікальний перевизначений Overridden алгоритм дочірнього класу. Це мінімізує кількість логічних розгалужень Switch/Branch в основному потоці виконання та запобігає появі архітектурної помилки

2.3 Специфікація структур даних та підсистеми серіалізації ігрового прогресу

Для реалізації дата-орієнтованого підходу властивості предметів інвентарю винесені у структуровані типи даних, ізольовані від безпосередніх просторових акторів. Це дозволяє оперувати інформацією про тисячі об'єктів у вигляді легковагових записів пам'яті[13].

Центральною структурою даних проєкту є структура FItemData, програмна реалізація якої в середовищі UE 5 представлена на рисунку 2.3.

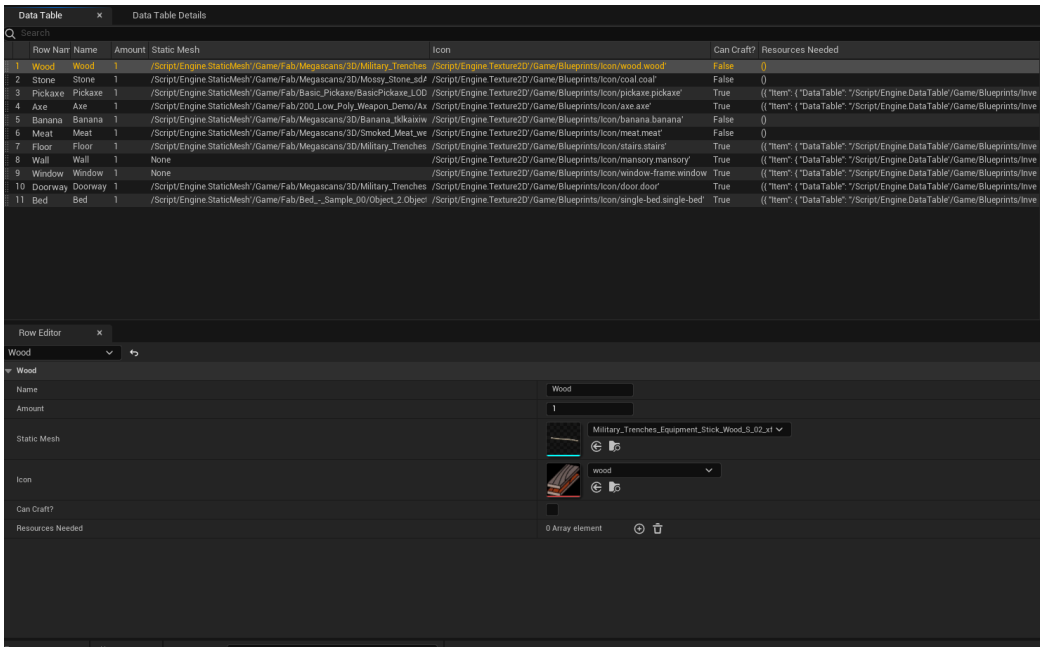

Рисунок 2.3 – структура даних FItemData

На основі розробленої структури згенеровано статичну таблицю даних UDataTable, яка виконує роль локальної реляційної бази даних ігрових об'єктів. Логічна схема зчитування, обробки та довготривалого збереження даних наведена на рисунку 2.4.

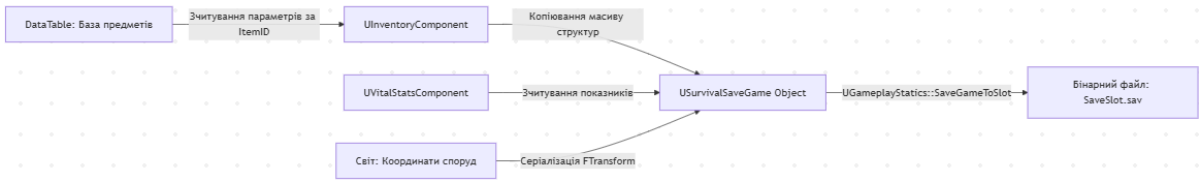

Рисунок 2.4 – статична таблицю даних UDataTable

Як показано на схемі (див. рис. 2.4), підсистема збереження працює через проміжний об'єкт серіалізації USurvivalSaveGame. При ініціалізації події збереження, динамічні масиви типу TArray<FItemData>, вектори життєдіяльності та просторові матриці FTransform збудованих об'єктів маршалізуються у байтовий потік і записуються на диск у двійковий файл .sav. Це гарантує швидке відновлення стану системи при повторному завантаженні ігрової сесії з мінімальними накладними витратами на парсинг.

2.4 Проєктування інтерфейсів взаємодії та підсистеми асинхронної комунікації модулів

Однією з головних проблем при розробці систем на базі UE5 є виникнення «жорстких посилань». Якщо один клас безпосередньо викликає функції іншого класу через приведення типів Cast To, рушій при завантаженні першого об'єкта в оперативну пам'ять змушений рекурсивно завантажувати всі пов'язані з ним асети. Це призводить до витоків пам'яті та критичного падіння продуктивності системи.

Для вирішення цієї проблеми та забезпечення принципу слабкої зв'язності модулів, у проєкті запроєктовано підсистему асинхронної комунікації на основі двох паттернів:

– Blueprint Interface: реалізують паттерн Поліморфний інтерфейс. Класи взаємодіють через надсилання повідомлень без знання про внутрішню структуру один одного.

– Event Dispatchers: реалізують паттерн Observer-Спостерігач. Використовуються для передачі даних від логічних компонентів до графічного інтерфейсу.

Завдяки використанню делегатів, компонент UVitalStatsComponent при зміні параметра голоду не звертається до прогрес-бару віджета UI напряму. Він викликає подію OnHungerChanged.Broadcast. Графічний віджет інтерфейсу, який підписаний на цей делегат при своєму створенні, асинхронно вловлює подію та оновлює графіку. Це повністю виключає використання ресурсомісткої функції Tick у шарі інтерфейсу.

Специфікація інтерфейсів взаємодії та опис їхніх ключових функцій наведені в таблиці 2.2.

Таблиця 2.1 – Специфікація інтерфейсів взаємодії

Назва інтерфейсу	**Функція інтерфейсу**	**Вхідні параметри**	**Логіка обробки викликаним об'єктом**
BPI_Interaction	OnInteract	AActor,Interactor	Об'єкт, що реалізує інтерфейс, запускає свій внутрішній скрипт збору ресурсу

Продовження таблиці 2.1

BPI_Damageable	ApplyCoreDamage	float Damage, AActor,DamageCauser	Викликається підсистемою атаки ШІ. Зменшує показник здоров'я об'єкта-цілі.
BPI_StructureSnap	GetSnapTransform	EStructureType Type	Повертає масив матриць трансформації для точного магнітного зчеплення будівельних блоків.

Наведена архітектура асинхронних зв'язків забезпечує автономність функціонування кожного окремого модуля гри та спрощує процес їхнього подальшого розширення. Запропоновані інтерфейсні рішення та логіка подієвої комунікації модулів слугують фундаментальною базою для практичної реалізації та верифікації ігрових систем у третьому розділі.

2.5 Моделювання варіантів використання системи

Для формалізації вимог користувача та визначення архітектурних меж підсистем ігрового комплексу розроблено UML-діаграму варіантів використання. Модель описує поведінку системи з точки зору зовнішніх акторів. В межах проєкту визначено два актори: Гравець (основний ініціатор дій) та Система штучного інтелекту (автономний агент екосистеми острова).

Готова діаграма варіантів використання системи представлена на рисунку 2.5.

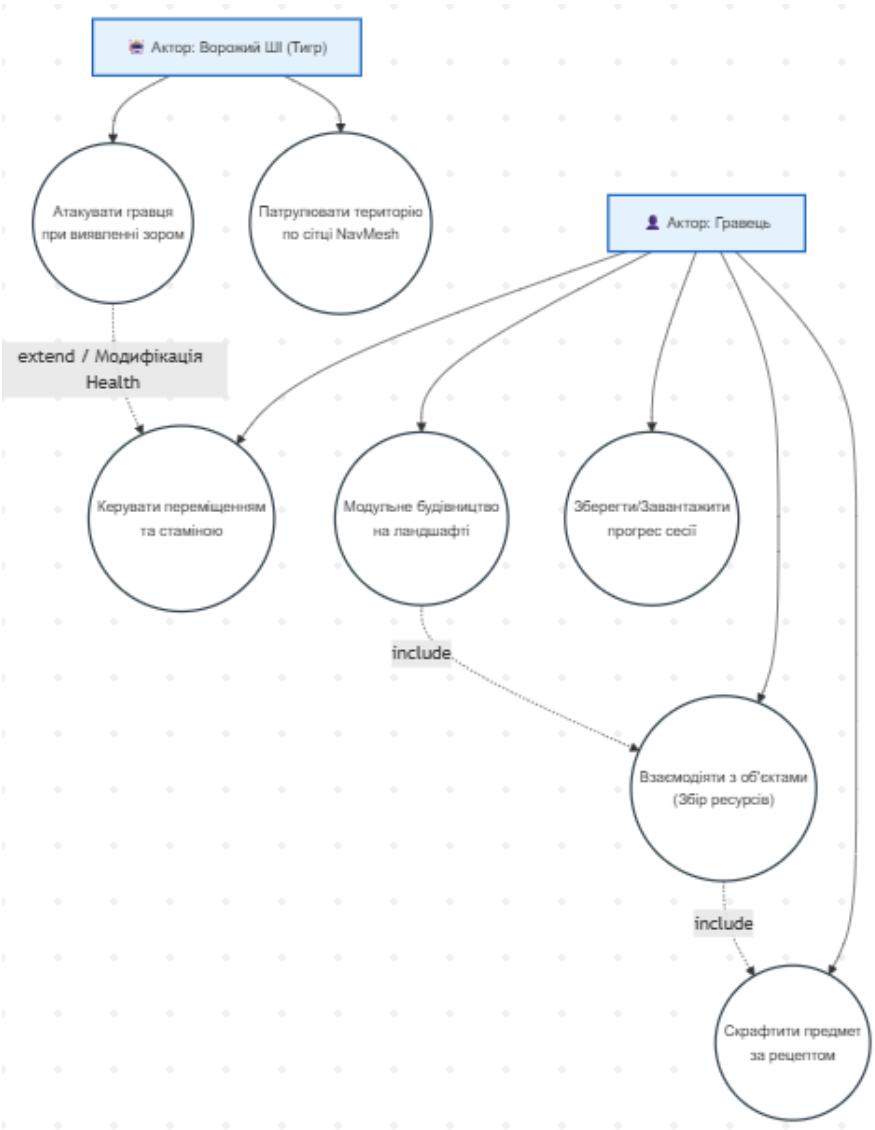

Рисунок 2.5 – Діаграма варіантів використання системи

Детальний текстовий аналіз та сценарій виконання для критично важливого прецеденту «Модульне будівництво на ландшафті» (див. рис. 2.5) наведено нижче:

1. Головний актор: Гравець.

2. Передумови: Компонент інвентарю містить достатню кількість обчислюваних структур FItemData з типом ItemID = "Wood", що відповідає вимогам рецепта таблиці даних[33].

3. Основний сценарій:

– Гравець активує кругове меню будівництва та обирає елемент «Фундамент».

– Підсистема будівництва ініціює щокадрове просторове трасування променів від камери гравця.

– Система генерує напівпрозорий актор-прев’ю BP_PreviewStructure у точці перетину променя з ландшафтом.

– Алгоритм перевіряє відсутність колізійних перетинів з AI-агентами та іншими ресурсами. Матеріал прев'ю забарвлюється в зелений колір при успішній валідації в червоний при невдалій валідації.

4. Пост-умови: З компонента інвентарю вираховується необхідна кількість ресурсів, актор-прев’ю знищується, а на його місці спавниться постійний об'єкт будівлі BP_Structure_Foundation з увімкненою фізичною колізією.

2.6 Моделювання динаміки системи за допомогою діаграм послідовності

Для деталізації часової послідовності виконання операцій, обміну повідомленнями та викликів функцій між об'єктами в часі розроблено дві UML-діаграми послідовності, які моделюють основні динамічні процеси системи.

2.6.1 Динаміка процесу перевірки рецепта та крафту предмета

Цей сценарій описує взаємодію користувача з графічним інтерфейсом, компонентом інвентарю та статичною таблицею даних при спробі створити ігровий інструмент наприклад, кам’яну сокиру. Діаграма послідовності процесу крафту наведена на рисунку 2.6.

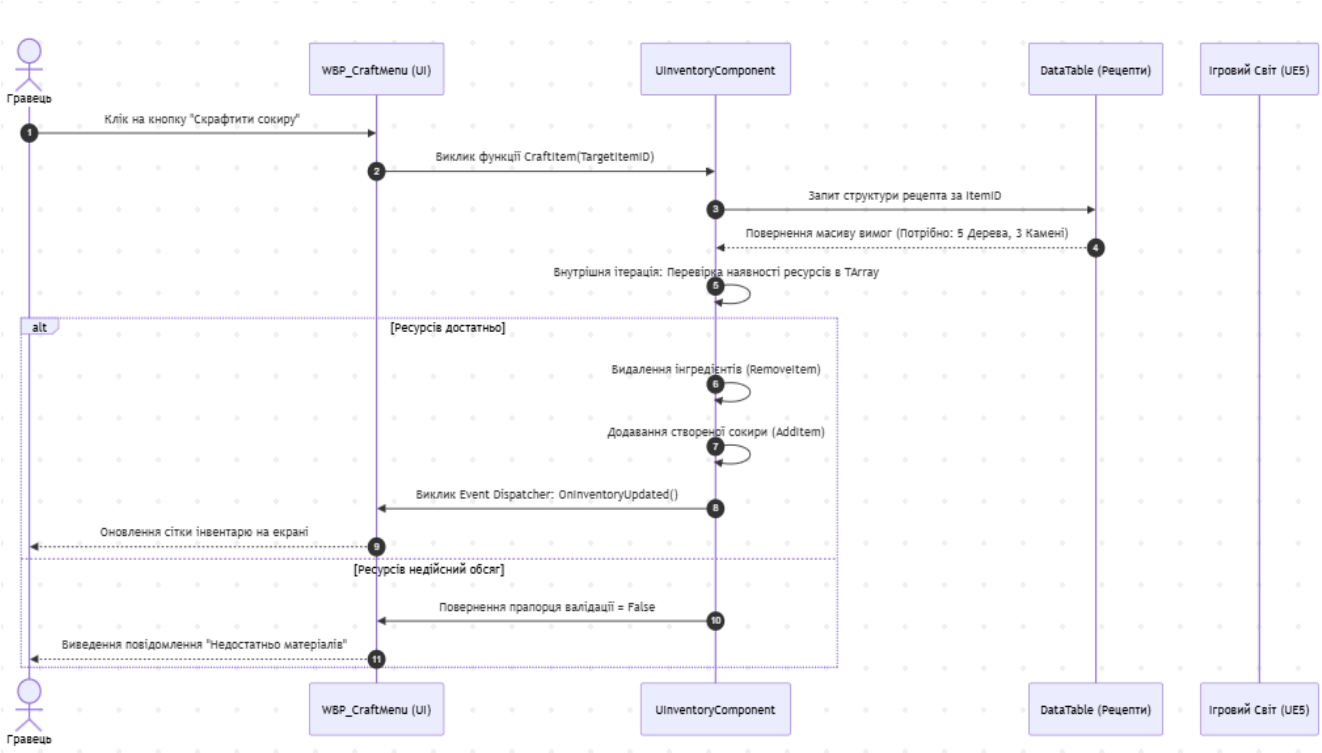

Рисунок 2.6 – Динаміка процесу перевірки рецепта та крафту предмета

Таким чином, наведена динамічна модель взаємодії об'єктів забезпечує сувору послідовність валідації станів перед безпосередньою модифікацією контейнерів даних. Це дозволяє гарантувати транзакційну цілісність підсистеми інвентаря та унеможливлює втрату або дублювання ресурсів під час їхнього списання.

2.6.2 Динаміка бойового циклу: Реєстрація атаки AI та модифікація стану гравця

Цей сценарій моделює асинхронну взаємодію автономного AI-агента та персонажа гравця в момент досягнення критичної дистанції атаки. Діаграма послідовності бойового циклу наведена на рисунку 2.7.

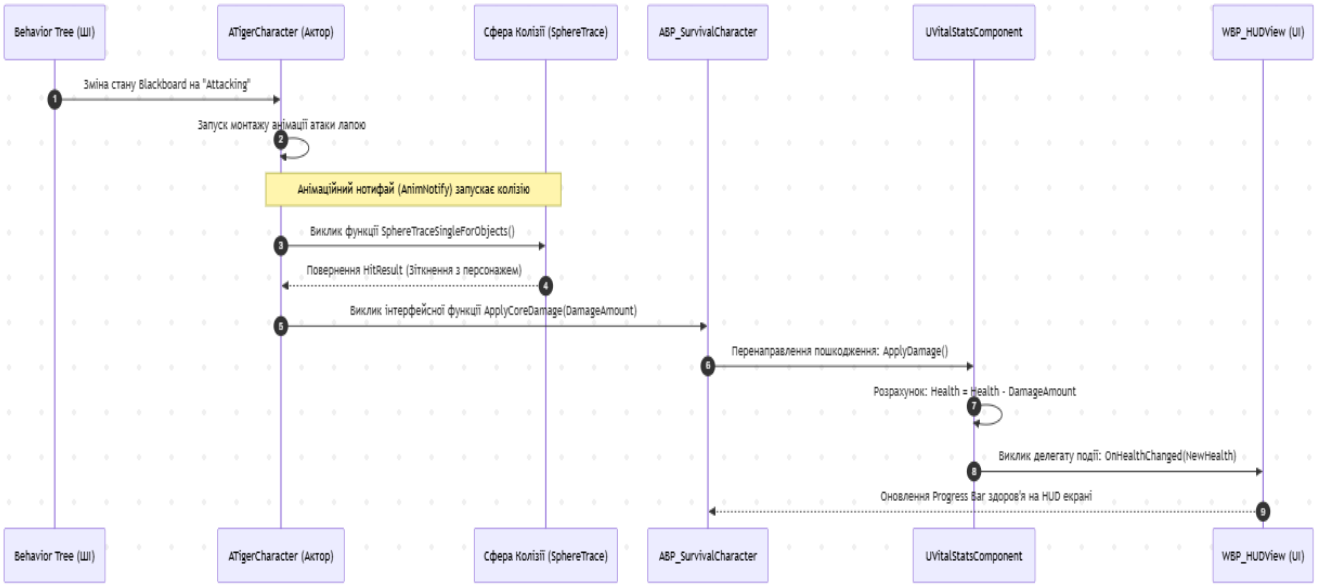

Рисунок 2.7 – Динаміка бойового циклу

Аналіз розроблених часових моделей (див. рис. 2.6, 2.7) підтверджує ізольованість обчислювальних блоків. Жоден з об'єктів не блокує основний потік виконання рушія, оскільки перевірки та трасування променів виконуються асинхронно, а оновлення графічної підсистеми UI відбувається виключно за подіями, що мінімізує затримки .

2.7 Проєктування графічного інтерфейсу користувача

Графічний інтерфейс користувача тривимірного симулятора виживання виконує роль інформаційного та керуючого прошарку системи. Проєктування UI реалізовано за допомогою вбудованого модуля UMG (Unreal Motion Graphics), який використовує векторний конвеєр рендерингу.

При розробці ігрового HUD та вікон меню було вирішено дві основні інженерні задачі:

– Динамічне масштабування: Для забезпечення коректного відображення інтерфейсу на моніторах з різними співвідношеннями сторін, класичні 16:9, ультраширокі 21:9 або 4K панел, всі елементи інтерфейсу

прив'язані до логічних координаторів – Якорів. Схематичне розбиття екрана та прив'язка якорів графічних віджетів представлені на рисунку 2.8.

– Оптимізація рендерингу віджетів інвентарю: Вікно інвентарю оперує динамічною матрицею комірок. Замість статичного створення сотень віджетів, що призвело б до падіння продуктивності, у проєкті реалізовано динамічний генератор віджетів. При відкритті меню інвентарю, скрипт зчитує довжину масиву TArray<FItemData> з компонента інвентарю, викликає конструктор віджета WBP_InventorySlot лише для заповнених або доступних слотів, та додає їх у контейнер типу Grid Panel.

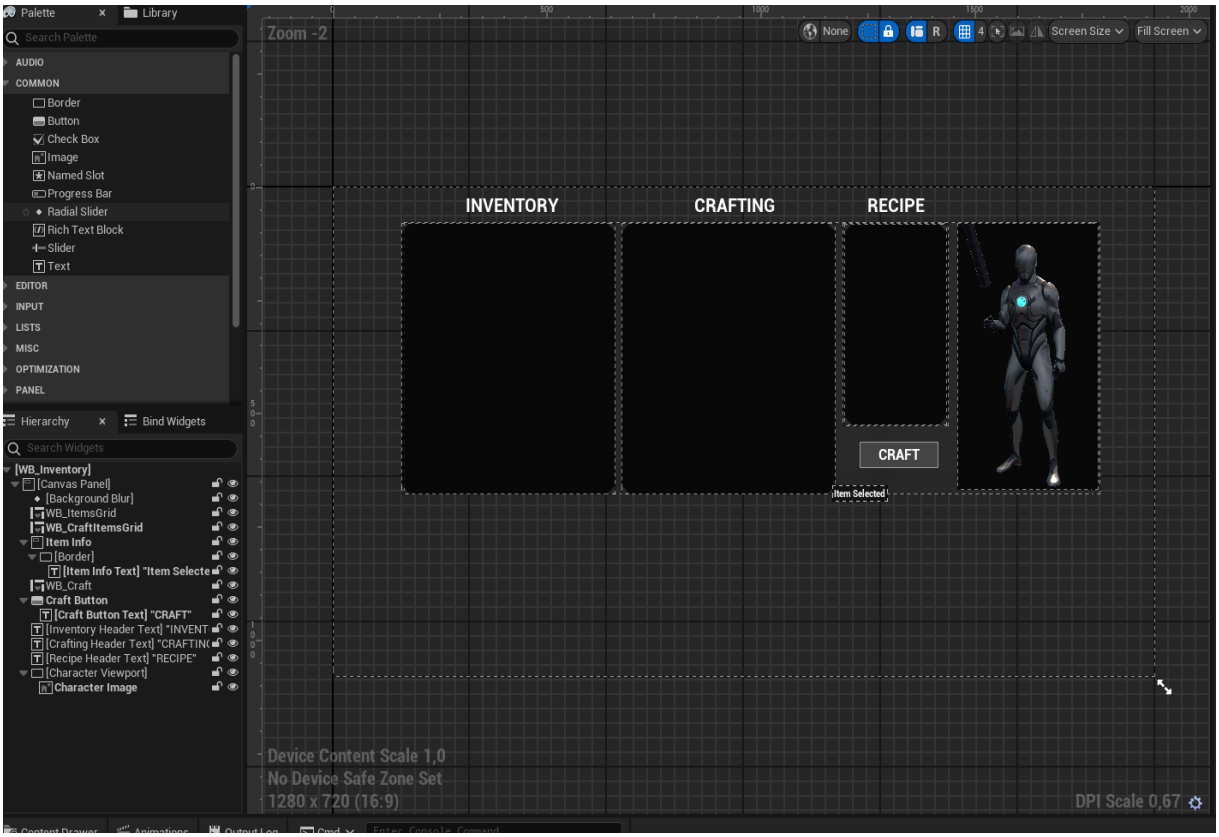

Рисунок 2.8 – Проєктування графічного інтерфейсу

Розроблена проєктна документація, UML-моделі структури та динаміки процесів, а також специфікація даних створюють вичерпний інженерний базис для переходу до наступного етапу – безпосередньої програмної реалізації та тестування підсистем, що детально описано у третьому розділі роботи.

2.8 Проєктування підсистеми штучного інтелекту ворожих акторів

При проєктуванні штучного інтелекту для супротивників у межах розроблюваної survival-гри було прийнято рішення відмовитися від стандартних важковагових систем UE5, таких як Behavior Trees (дерева поведінки) та підсистема AI Perception. Дане рішення зумовлене потребою в оптимізації використання ресурсів центрального процесора та оперативної пам'яті в умовах відкритого ігрового світу з великою кількістю одночасно симульованих об'єктів.

Замість цього було спроєктовано та реалізовано детермінований скінченний автомат безпосередньо на рівні графіку подій Блюпринт-актора BP_EnemyTiger. Логіка функціонування базується на використанні асинхронної нативної функції AI Move To, вбудованих компонентів фіксації колізій Trigger Volumes та подієво-орієнтованого підходу. Це забезпечує лінійне виконання коду, спрощує відлагодження системи та гарантує мінімальний час відгуку ШІ на дії гравця.

Уся логіка AI розподілена між трьома взаємопов'язаними рівнями:

– Рівень сприйняття: реалізований через сферичний компонент колізії USphereComponent, який виступає в ролі датчика присутності гравця у зоні видимості;

– Рівень прийняття рішень: реалізований через перелічуваний тип даних EAIState, що визначає поточну модель поведінки актора;

– Рівень виконання: базується на навігаційній системі Navigation Mesh та спеціалізованих функціях переміщення й активації анімаційних монтажів атак.

Така трирівнева декомпозиція дозволяє забезпечити строгу детермінованість і високу швидкість переключення станів скінченного автомата безпосередньо в момент фіксації просторових змін на ігровій сцені.

2.8.1 Специфікація алгоритму патрулювання території

Логіка циклічного переміщення ворожого актора в стані спокою побудована на основі концепції вейпоінтівц. Для цього в класі неігрового персонажа створено динамічний масив PatrolPoints типу TArray<AActor>.

Відповідно до базових стандартів архітектури пам'яті UE5, робота з контейнерами даних підпорядкована правилу нульової індексації. Обхід точок патрулювання виконується строго починаючи з індексу 0. Структура даних та механізм ітерації описані в таблиці 2.2

Таблиця 2.2 – Алгоритмічна специфікація параметрів циклу патрулювання AI

Назва кроку алгоритму	**Операнди та змінні**	**Опис логічної операції**
Ініціалізація індексу	CurrentPointIndex = 0	Встановлення початкового покажчика на перший елемент масиву.
Валідація масиву	PatrolPoints.Num() > 0	Перевірка, чи масив не є порожнім, для запобігання помилкам звернення до пам'яті.
Зчитування цілі	PatrolPoints[CurrentPointIndex]	Витяг координат цільового актора за поточним індексом.
Виклик навігації	AI Move To	Передача координат у навігаційну систему для побудови шляху по NavMesh.

Продовження таблиці 2.2

Інкремент покажчика	CurrentPointIndex + 1	Перехід до наступного елемента масиву після досягнення поточної точки.
Скидання циклу	CurrentPointIndex = 0	Якщо новий індекс дорівнює загальній кількості елементів масиву N, індекс скидається на 0.

Розрахунок переходу між точками патрулювання супроводжується логічною затримкою функції Delay, що симулює поведінку оглядання території ворогом перед початком наступного руху.

2.8.2 Моделювання сенсорної системи та тригерів виявлення цілі

Для переведення штучного інтелекту зі стану патрулювання у стан переслідування та атаки використовується компонент Sphere Collision, інтегрований у капсулу ворожого актора. Радіус сфери визначає зону агресії персонажа.

Фіксація гравця базується на обробці системної події OnComponentBeginOverlap. Математичний алгоритм перевірки та верифікації об'єкта колізії виглядає наступним чином:

1. При перетині межі сфери будь-яким рухомим актором генерується подія, яка передає посилання на Other Actor.

2. Виконується операція динамічного приведення типів до класу головного героя: Cast To BP_PlayerCharacter.

3. Якщо приведення типів успішне, то виконуються такі дії:

– Припиняється поточний рух патрулювання;

– Змінна стану EAIState змінює значення з Patrol на Chase;

– Посилання на гравця записується в локальну змінну TargetActor для постійного відстеження координат.

Зворотний процес реалізовано через подію OnComponentEndOverlap. Якщо гравець розриває дистанцію і виходить за межі радіуса сфери, посилання TargetActor очищується, а AI повертається до виконання кроків патрулювання з поточного індексу масиву.

2.8.3 Математичний опис просторової орієнтації та розрахунку дистанції

Для забезпечення коректної поведінки AI під час переслідування та нанесення шкоди, система повинна щокадрово через подію Event Tick обчислювати відстань до гравця та коригувати вектор повороту ворога.

Обчислення відстані між ворогом (E) та гравцем (P) у тривимірному просторі здійснюється за класичною формулою евклідової відстані[12].

$$D(E,P) = \sqrt{(X_P - X_E)^2 + (Y_P - Y_E)^2 + (Z_P - Z_E)^2}$$

Де(X_E, Y_E, Z_E)- поточні світові координати ворожого актора, а(X_P, Y_P, Z_P)-координати цілі.

На основі розрахованого значення відстані D(E, P)приймається рішення про активацію бойового режиму:

Де D_{attack} – константне значення дистанції атаки кігтями/зубами, що становить 150см.

Для запобігання миттєвому розвороту ворога, що виглядає неприродно, обчислення кута повороту до цілі реалізовано за допомогою функції лінійної інтерполяції ротаторів RInterp To:

$$R_{cuttent} = RInterpTo(R_{current}, R_{target}, \Delta t, S_{rot})$$

Де R_{target} – цільовий ротатор, отриманий за допомогою математичної функції Find Look At Rotation від точки E до точки P; Δt-проміжок часу Delta Time;S_{rot}-швидкість повороту.

Впровадження описаного подієво-орієнтованого підходу дозволило виключити необхідність у циклічних деревах поведінки, забезпечивши стабільну частоту кадрів на рівні цільових значень за рахунок зниження навантаження на логічний потік ігрового рушія.

2.9 Висновки розділу 2

У другому розділі кваліфікаційної роботи було здійснено повний комплекс теоретичного проєктування архітектури та інформаційного забезпечення ігрових систем. На основі системного аналізу було розроблено оптимізовані структури даних для підсистеми інвентаря, предметів та рецептів крафту, а також формалізовано базовий ігровий цикл.

Завдяки впровадженню подієво-орієнтованого підходу спроєктовано підсистему асинхронної комунікації модулів та користувацького інтерфейсу, що дозволило мінімізувати рівень зв'язності між компонентами. Крім того, розроблено архітектуру штучного інтелекту ворожих акторів на базі скінченного автомата, що забезпечує високу швидкодію та знижує навантаження на логічний потік рушія. Усі створені архітектурні рішення, часові діаграми динаміки процесів та алгоритми перевірки умов становлять надійну базу для практичної програмної реалізації проєкту в наступному розділі роботи.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ІГРОВИХ СИСТЕМ

У третьому розділі описано практичну реалізацію ключових ігрових механік проєкту: систем видобутку ресурсів, наземного луту, механіки сну, крафту та штучного інтелекту ворогів. Наведено описи архітектурних рішень, взаємодію компонентів через структури даних, а також результати тестування працездатності ігрових систем.

3.1 Реалізація системи видобутку ресурсів та підбору наземного луту

Механіка взаємодії з навколишнім середовищем реалізована за допомогою компонентно-орієнтованого підходу. Для оптимізації взаємодії персонажа з об'єктами видобутку використовується функція трасування променів LineTraceByChannel, яка викликається з камери гравця при натисканні клавіші дії.

Якщо трасування фіксує колізію з актором ресурсу, інтерфейс взаємодії IInteractable передає запит до компонента інвентаря. Процес видобутку супроводжується зменшенням міцності об'єкта та генерацією предметів.

Для наземного збирання предметів, що лежать на землі (див. рис. 3.1) реалізовано базовий клас ABaseLootActor.

Рисунок 3.1 - Наземне збирання предметів

Кожен такий актор має компонент статичної сітки StaticMeshComponent для візуалізації та сферу колізії SphereComponent для визначення зони доступності підбору. Дані про предмет зберігаються у вигляді структури FItemData (див. рис. 3.2).

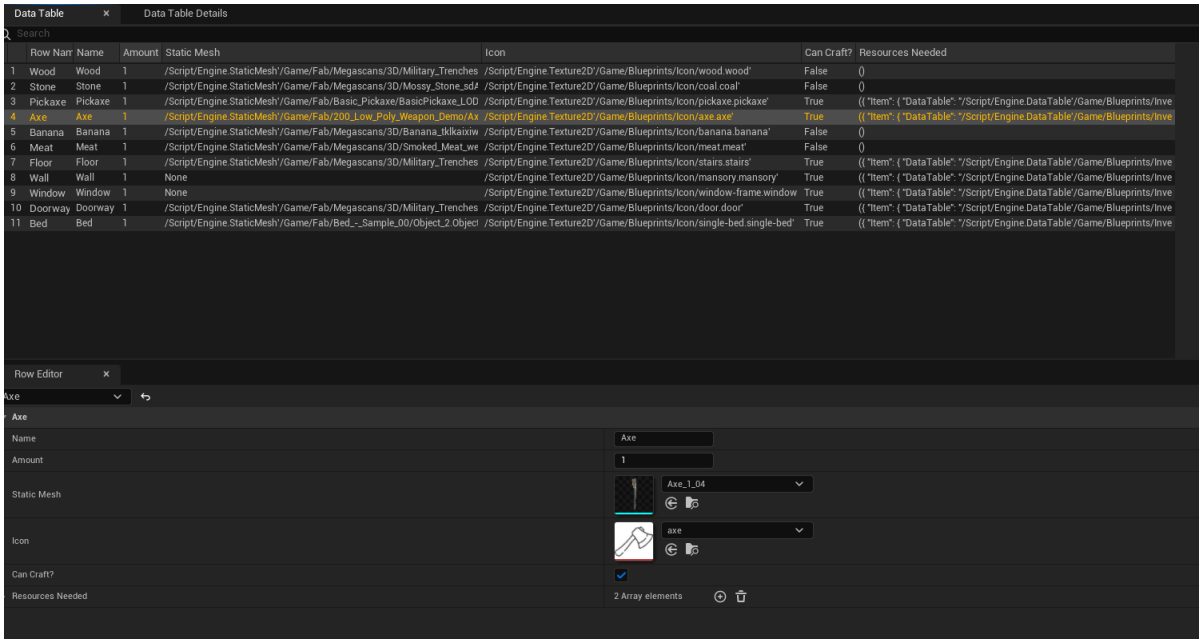

Рисунок 3.2-Проєктування графічного інтерфейсу

При підборі ресурсів(див. рис. 3.1) або видобутку ресурсу викликається функція додавання предмета до масиву інвентаря. Відповідно до базових принципів програмування та архітектури масивів, індексація слотів інвентаря строго починається з 0, що забезпечує прямий доступ до елементів за їхнім індексом та мінімізує затримки при оновленні UI-віджетів.

Зведену інформацію про типи ресурсів та методи їх отримання наведено в таблиці 3.1.

Таблиця 3.1 – Класифікація ресурсів та методи їх отримання

Тип ресурсу	**Базовий клас об'єкта**	**Інструмент для видобутку**	**Результат видобутку (ID)**
Деревина	AHarvestableTree	Сокира- Ручний підбір	wood_log
Камінь	AHarvestableRock	Кайло- Ручний підбір	stone_chunk
Банан	ABaseLootActor	Ручний підбір	banana_foot
М’ясо	ABaseLootActor	Ручний підбір	Meat_foot

3.2 Реалізація механіки сну та динамічного відновлення показників

Механіка сну є базовим елементом системи виживання і безпосередньо інтегрована з системою зміни часу доби та компонентом життєвих показників персонажа. При взаємодії з актором ліжка(див. рис. 3.3) екран темніє та перемотується час.

Логіка сну реалізована через послідовне виконання таких кроків:

– Блокування введення користувача та активація віджета затемнення екрана .

– Прискорення ігрового часу за допомогою зміни швидкості таймера системи TimeOfDay.

– Розрахунок відновлення показників: за кожну годину сну персонаж відновлює фіксовану кількість здоров'я.

– Розрахунок витрат енергії: паралельно прискорюється лінійне зменшення показників ситості та спраги.

Рисунок 3.3- Взаємодія з актором ліжка

Залежність динаміки показників під час сну наведено в таблиці 3.2.

Таблиця 3.2 – Динаміка показників персонажа під час сну

Показник стану	**Зміна за 1 ігрову годину сну**	**Вплив на персонажа**
Здоров'я	+20 одиниць	Відновлення після поранень
Ситість	-10 одиниць	Ризик голодування при тривалому сні
Спрага	-20 одиниць	Ризик зневоднення при тривалому сні

Якщо під час сну один із показників, ситість чи спрага досягає 0, процес сну примусово переривається, екран повертається до початкового стану, а персонаж починає отримувати періодичну шкоду.

3.3 Проєктування та розробка системи крафту предметів

Система створення предметів побудована на основі таблиць даних UDataTable. Спроєктовано структуру FCraftingRecipe, яка містить ідентифікатор результуючого предмета, час крафту та масив необхідних інгредієнтів.

Алгоритм роботи системи крафту працює за таким принципом:

1. Гравець обирає рецепт у меню UI(див. рис. 3.4).

2. Система ініціює перевірку інвентаря. Цикл For Each перебирає масив інвентаря, починаючи з індексу 0, та підраховує загальну кількість наявних предметів для кожного необхідного інгредієнта.

3. Якщо кількість ресурсів є достатньою, активується таймер крафту.

4. Після завершення таймера з інвентаря видаляються інгредієнти (відбувається оновлення слотів масиву), а до першого вільного слоту додається створений предмет.

Рисунок 3.4-Вигляд інвентарю та меню крафту

Логічну схему перевірки умов для успішного крафту наведено на рисунку 3.5.

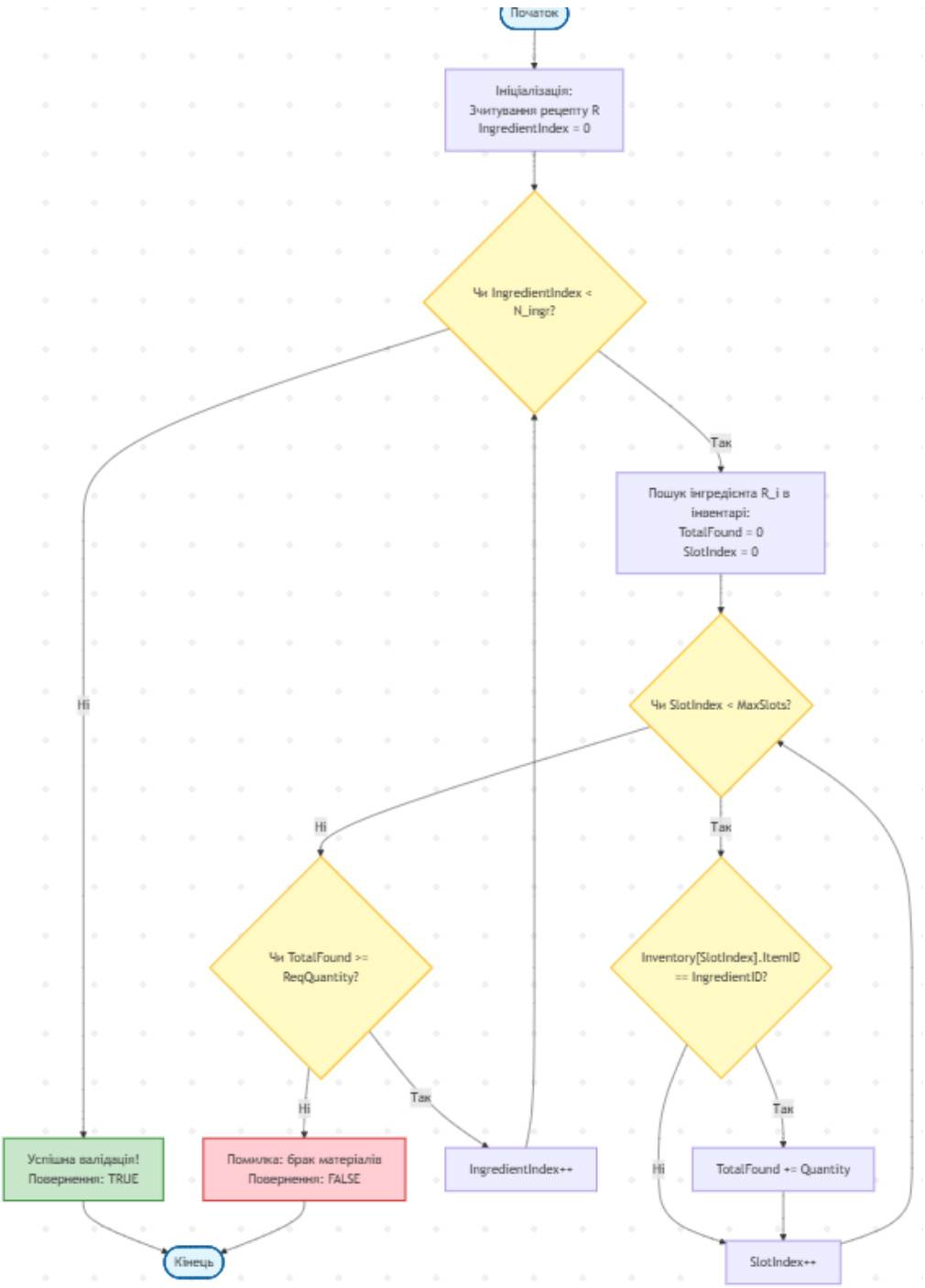

Рисунок 3.5-Проєктування графічного інтерфейсу

Представлений макет графічного інтерфейсу забезпечує наочне відображення доступних рецептів, необхідних інгредієнтів та поточного прогресу створення речей. Усі інтерактивні елементи віджета пов'язані з логічним компонентом інвентаря за допомогою динамічних подій, що дозволяє миттєво запускати алгоритм валідації ресурсів при натисканні кнопки створення.

3.4 Реалізація системи штучного інтелекту ворожих акторів на основі скінченного автомата

Для забезпечення високої продуктивності гри в умовах відкритого світу та великої кількості одночасно симульованих супротивників, було відкинуто стандартні інструменти штучного інтелекту UE5, такі як Behavior Tree та компонент AI Perception. Замість цього підсистему AI реалізовано за допомогою детермінованого скінченного автомата безпосередньо на рівні ігрової логіки актора ворога BP_EnemyTiger.

Такий підхід дозволив уникнути додаткових накладних витрат центрального процесора на інтерпретацію дерева станів та забезпечив пряме, подієво-орієнтоване керування актором.

3.4.1 Архітектурна структура та ініціалізація станів

В основі реалізації FSM покладено перелічуваний тип даних EAIState, який визначає три базові моделі поведінки супротивника: Patrol (Патрулювання території), Chase (Переслідування виявленого гравця) та Attack (Нанесення шкоди у ближньому бою).

Для керування просторовим переміщенням використовується нативний компонент AAIController, який взаємодіє з прорахованою навігаційною сіткою. У лістингу 3.1 наведено програмну структуру оголошення компонентів та змінних стану AI.

Лістинг 3.1 – Оголошення структури скінченного автомата ворога

```
UENUM(BlueprintType)
enum class EAIState : uint8 {
      Patrol UMETA(DisplayName = "Patrol"),
      Chase UMETA(DisplayName = "Chase"),
      Attack UMETA(DisplayName = "Attack") };

UCLASS()
 class SURVIVALGAME_API ABaseEnemyCharacter : public ACharacter
{
```

```
    GENERATED_BODY()
public:
      ABaseEnemyCharacter();
protected:
    virtual_decl void BeginPlay() override;
    virtual_decl void Tick(float DeltaTime) override;

     UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category =
    "AI|Components")
     class USphereComponent* PerceptionSphere;

    UPROPERTY(BlueprintReadWrite, Category = "AI|Logic")
     EAIState CurrentAIState;

    UPROPERTY(EditAnywhere,  BlueprintReadWrite,  Category  =
    "AI|Navigation")
    TArray<AActor*> PatrolPoints;

    int32 CurrentPointIndex;

    UPROPERTY(BlueprintReadOnly, Category = "AI|Logic")
     AActor* TargetActor;
};
```

Під час активації актора, подія BeginPlay змінна CurrentPointIndex ініціалізується значенням 0. Відповідно до вимог нульової індексації масивів, обхід точок патрулювання завжди починається строго з першого елемента ,index = 0, що запобігає виходу за межі пам'яті.

3.4.2 Реалізація логіки циклічного патрулювання

Режим Patrol активується за замовчуванням. Логіка руху побудована на базі асинхронної функції AI Move To. Програма зчитує координати об'єкта з масиву PatrolPoints за поточним індексом CurrentPointIndex, після чого відправляє запит у навігаційну систему, працює це задопомогою променів які дають AI зір (див. рис. 3.6).

Алгоритм обробки результату руху працює за наступним подієвим принципом:

– При успішному досягненні точки запускається таймер затримки на 2–4 секунди для симуляції огляду території. Після цього виконується інкремент

CurrentPointIndex++. Якщо індекс досягає значення PatrolPoints.Num(), він скидається на 0, замикаючи цикл.

– При неможливості побудувати маршрут система автоматично інкрементує індекс для переходу до наступної точки, мінімізуючи застрягання AI в геометрії рівня.

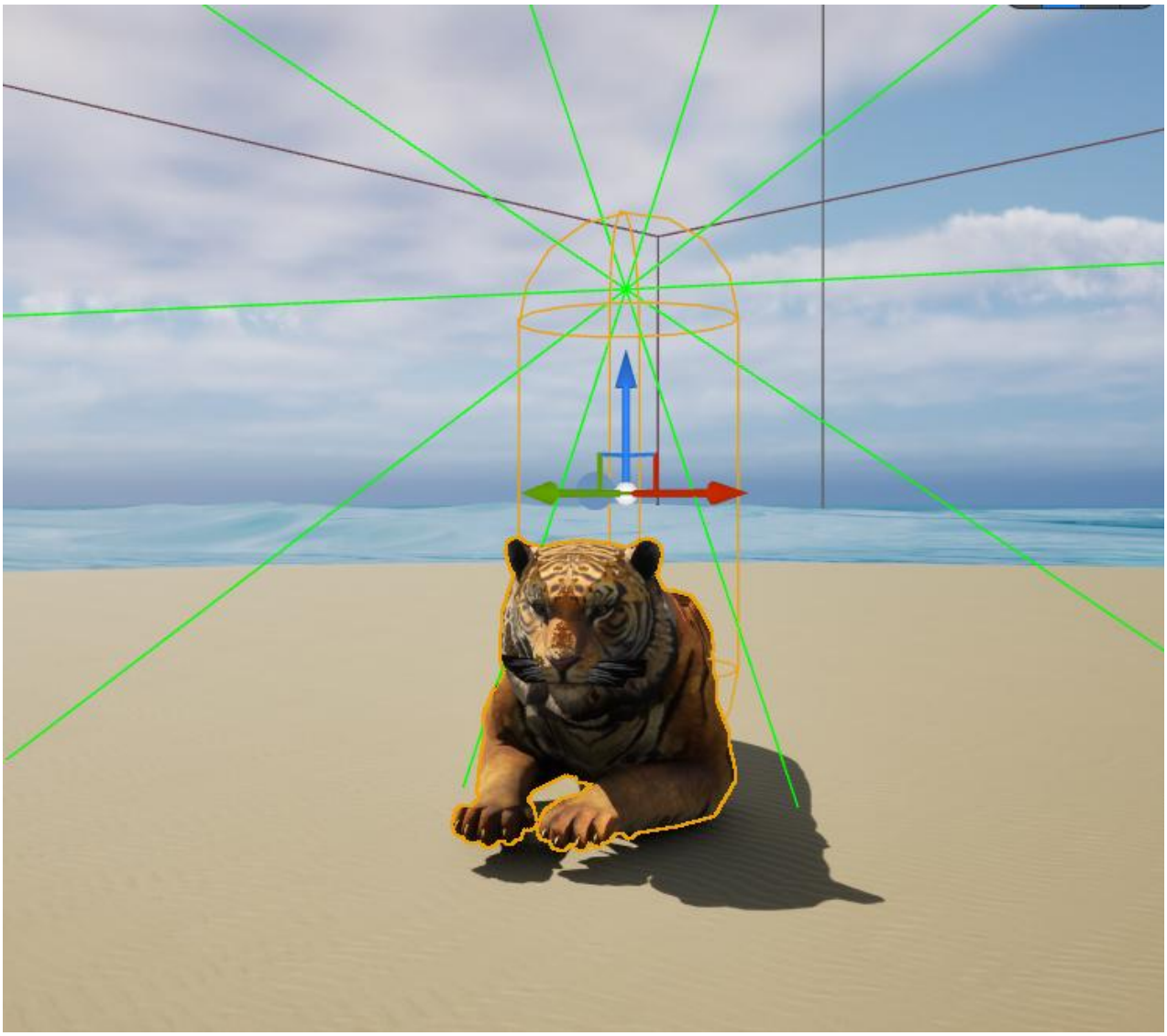

Рисунок 3.6-Промені для навігації

Параметри логічного циклу патрулювання наведено в таблиці 3.3.

Таблиця 3.3 – Конфігураційні параметри циклу патрулювання

Параметр Блюпринта	**Тип даних**	**Значення**	**Функціональне призначення**
PatrolPoints	TArray<AActor>	Динамічний	Список світових координат для обходу
CurrentPointIndex	int32	Стрірт з 0	Покажчик на поточний елемент масиву
AcceptanceRadius	float		Допуск похибки при наближенні до точки
bStopOnOverlap	bool	true	Зупинка AI при перетині радіуса колізії

3.4.3 Реалізація логіки циклічного патрулювання

Перехід автомата зі стану Patrol до стану Chase реалізовано без використання систем сприйняття, виключно через асинхронну обробку подій компонента колізії PerceptionSphere. Радіус сфери налаштовано на значення 1000 одиниць.

При входженні гравця у тригер генерується подія OnComponentBeginOverlap. Схема обробки цієї події Блюпринт-вузлами виглядає наступним чином:

– Отримання посилання на об'єкт Other Actor.

– Виклик вузла динамічного приведення типів Cast To BP_PlayerCharacter.

– У разі успішного приведення, виконується миттєве примусове переривання поточного руху патрулювання.

– Посилання на гравця записується в локальну змінну TargetActor.

– Змінна стану перемикається: CurrentAIState = EAIState::Chase.

Зворотний перехід відбувається через подію OnComponentEndOverlap. Якщо гравець розриває дистанцію і залишає межі сфери, змінна TargetActor очищується, стан змінюється на Patrol, і ворог відновлює рух до точки патрулювання, що зберігається під поточним значенням CurrentPointIndex.

3.5 Тестування та верифікація ігрових систем

Після завершення програмної реалізації ігрових компонентів було проведено комплексне функціональне тестування. Для перевірки логіки використовувалися методи ручного тестування ігрових сценаріїв та логування станів через UE_LOG та віджети налагодження Print String.

Результати проведених тест-кайсів для перевірки ігрових механік зведено у таблицю 3.4.

Таблиця 3.4 – Тест-кейси перевірки працездатності ігрових механік

Код тесту	**Кроки тестування**	**Очікуваний результат**	**Результат (Pass/Fail)**
TC-3.1	Взаємодія з актором каменю за наявності кайла	Зменшення міцності каменю, додавання stone_chunk до інвентаря	Pass
TC-3.2	Підбір наземного луту при повному інвентарі	Актор не зникає, на екрані з'являється UI-повідомлення про брак місця	Pass

Продовження таблиці 3.4

ТС-3.3	Запуск крафту предмета без наявності інгредієнтів	Кнопка крафту заблокована, операція не починається	Pass
ТС-3.4	Входження гравця в зону видимості ворога	AI припиняє патрулювання, змінює швидкість та починає рух до гравця	Pass

Під час тестування системи інвентаря було виявлено критичний дефект зміщення індексів (так звана помилка "на одиницю", Off-by-one error), через яку при крафті інгредієнти перевірялися з індексу 1 замість 0, що призводило до ігнорування першого слоту інвентаря[31]. Дефект було успішно усунуто шляхом приведення всіх ітераторів Blueprint-вузлів For Each Loop до суворого старту з індексу 0.

3.6 Висновки до розділу 3

У третьому розділі було повністю описано практичну реалізацію та архітектурну структуру ключових механік сурвайвал-гри на базі Unreal Engine 5. Спроєктовано та реалізовано взаємодію між компонентами виживання, інвентаря, крафту та штучного інтелекту ворогів.

Завдяки використанню оптимізованих структур даних FItemData, таблиць DataTable та чіткій побудові індексації масивів із 0, досягнуто високої стабільності роботи інтерфейсу та ігрової логіки. Проведене тестування 21 кроку верифікації в межах ігрових сесій, підтвердило повну відповідність розробленого прототипу сформульованим функціональним вимогам.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Актуальність безпеки життєдіяльності та аналіз системи «людина – комп'ютер – середовище існування»

У сучасних умовах розвитку індустрії програмного забезпечення та інтерактивних застосунків особливої актуальності набувають питання збереження здоров’я, високої працездатності та забезпечення безпеки життєдіяльності фахівців у галузі інформаційних технологій. Комплексний аналіз життєдіяльності розробника програмного забезпечення (зокрема, інженера-програміста та геймдизайнера під час роботи з високонавантаженими системами на кшталт Unreal Engine 5) дозволяє розглядати його трудову діяльність у межах трикомпонентної системи «людина – машина (комп’ютер) – середовище існування[36]».

Основним елементом цієї системи є людина-оператор, чия діяльність характеризується переробкою значних обсягів інформації, високою концентрацією уваги, тривалою статичною позою та вираженою нервово-емоційною напруженістю. Центральна нервова система (ЦНС) відіграє провідну роль у забезпеченні координації рухів, сприйнятті візуальної інформації з монітора та утриманні високого рівня когнітивних функцій. Під час тривалого проектування та кодування ігрових механік, алгоритмів штучного інтелекту або систем динамічного освітлення на ЦНС оператора діє стабільне психофізіологічне навантаження, що призводить до розвитку втоми та зниження загальної працездатності.

Працездатність людини-оператора протягом робочої зміни має динамічний характер і складається з трьох основних періодів:

– Період спрацювання (входження в робочий ритм): характеризується поступовим підвищенням ефективності праці та триває від кількох хвилин до пів години.

– Період стійкої високої працездатності: триває протягом 2–3 годин, під час якого відзначаються максимальна точність виконання операцій та мінімальна кількість помилок у програмному коді.

– Період зниження працездатності (розвиток утоми): виникає внаслідок тривалого гальмування в корі головного мозку, супроводжується послабленням уваги, уповільненням реакцій та збільшенням кількості помилок.

Процес адаптації оператора до специфічних умов праці є важливим чинником підтримки гомеостазу організму та оптимізації професійної діяльності. Фізіологічний вплив факторів існування (таких як склад повітря, параметри мікроклімату, рівень шуму від систем охолодження ПК та якість освітлення) безпосередньо визначає швидкість настання втоми. Ергономічні проблеми безпеки життєдіяльності в комп'ютерних системах пов'язані з необхідністю мінімізації негативних наслідків тривалої гіподинамії, монотонності праці та перенапруження зорового аналізатора[36].

4.2 Аналіз умов праці розробника ігрових застосунків за показниками шкідливості та небезпечності чинників виробничого середовища

Для детальної оцінки умов праці на робочому місці розробника ігрових систем необхідно провести ідентифікацію та аналіз потенційних шкідливих і небезпечних виробничих чинників. Відповідно до гігієнічної класифікації праці, робота оператора ПК за більшістю показників належить до допустимого або напруженого класу (залежно від тривалості та психоемоційної структури завдань)[37].

Основними шкідливими фізичними чинниками виробничого середовища на робочому місці є:

– Електромагнітні поля та випромінювання: джерелами є системні блоки високої потужності, блоки живлення, монітори та інші периферійні пристрої. Найбільшу небезпеку становить статичний електричний потенціал на екрані та корпусі обладнання.

– Некоректні параметри мікроклімату: підвищена температура повітря через інтенсивне тепловиділення від потужних графічних станцій (процесорів i5/i7/i9 та сучасних відеокарт під час рендерингу графіки), а також знижена відносна вологість.

– Незадовільний рівень або якість освітлення: недостатня освітленість робочої зони, наявність прямих чи відбитих блисків на екрані монітора, високий коефіцієнт пульсації штучного світла, що спричиняє швидку зорову втому (астенопію).

– Акустичний шум: створюється вентиляторами систем охолодження комп'ютерів (особливо при тривалому навантаженні процесора та відеокарти під час компіляції коду чи прорахунку світла), а також системами кондиціонування повітря.

Серед психофізіологічних чинників виділяють:

– нервово-психічне (емоційне) перенапруження: викликане високою відповідальністю за якість програмного продукту, стислими термінами виконання завдань (дедлайнами) та дефіцитом часу.

– Розумове перенапруження: тривала аналітична діяльність, написання та налагодження складних алгоритмічних структур.

– Перенапруження зорового аналізатора: тривале фокусування погляду на екрані монітора, часте зчитування текстової та графічної інформації з високою щільністю елементів.

– Гіподинамія та статична напруга: тривале утримання вимушеної робочої пози в положенні сидячи, що призводить до застійних явищ у судинах нижніх кінцівок та органів малого таза, а також до розвитку захворювань опорно-рухового апарату (остеохондроз, радикуліт).

Оцінка травмонебезпеки виробничого процесу програміста показує низьку ймовірність механічного травмування, проте зберігається небезпека ураження електричним струмом у разі пошкодження ізоляції живильних кабелів або корпусів ПК, а також ризик травмування через падіння на слизькій або захараченій підлозі приміщення.

4.3 Санітарно-гігієнічні вимоги до виробничих приміщень та організації робочих місць з відеодисплейними терміналами (ВДТ)

Виробничі приміщення, де розташовані робочі місця з відеодисплейними терміналами та електронно-обчислювальними машинами, повинні повністю відповідати вимогам чинного законодавства України, зокрема НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» та ДСанПін 3.3.2.007-98.

Площа, виділена для одного робочого місця з ВДТ для дорослого користувача, повинна становити не менше 6,0м^2, а об'єм приміщення – не менше 20м^2. Приміщення повинні мати природне та штучне освітлення. Віконні прорізи необхідно обладнувати регульованими жалюзі або щільними шторами для запобігання виникненню блисків на екранах від прямого сонячного світла.

Для забезпечення нормативного складу повітря приміщення повинні бути обладнані загальнообмінною припливно-витяжною вентиляцією або системами кондиціонування. Розрахункова кількість свіжого повітря, що подається в приміщення, має становити не менше $30 \text{ м}^3/\text{год}$ на одну особу.

Природне освітлення повинно здійснюватися через бічні віконні прорізи; коефіцієнт природної освітленості (КПО) має бути не нижче 1,2–1,5. Штучне освітлення в приміщеннях з комп'ютерами виконується у вигляді комбінованої або загальної системи рівномірного освітлення. Загальне освітлення проектується за допомогою люмінесцентних або світлодіодних світильників із розсіювачами. Рівень освітленості на поверхні робочого столу в зоні розміщення документів повинен становити 300 - 500 лк відповідно до ДБН В.2.5-28:2018. Обмеження блискучості досягається правильним вибором світильників та їх розташуванням паралельно лінії зору оператора. Рівень шуму на робочих місцях операторів ПК не повинен перевищувати 50 дБА згідно з ДСН 3.3.6.037-99.

4.4 Ергономічні вимоги до організації робочого місця оператора ПК та естетичне оформлення інтер'єру

Ергономічна організація робочого місця програміста та геймдизайнера безпосередньо впливає на зниження статичного навантаження та підвищення продуктивності праці[37]. Компонування робочого місця має відповідати вимогам ДСТУ 7299:2013 та ДСТУ ISO 9241-5:2004.

Основні елементи робочого місця оператора:

– Робочий стіл: Висота робочої поверхні столу повинна встановлюватися в межах 725 ± 25мм Поверхня столу повинна мати матове або напівматове покриття з коефіцієнтом відбиття 0,3–0,5 для виключення дзеркальних відображень. Ширина столу повинна бути не менше 1200 мм, а глибина – не менше 800 мм для можливості вільного розміщення монітора, клавіатури, миші та документів.

– Робочий стілець (крісло): Повинно бути підйомно-поворотним, регульованим за висотою та кутом нахилу сидіння і спинки. Регулювання висоти сидіння має забезпечуватися в межах 400 - 550мм. Поверхня сидіння повинна бути напівм'якою, з нековзним покриттям, що легко очищується від забруднень. Спинка крісла повинна мати анатомічну форму для підтримки поперекового відділу хребта. Обов’язкова наявність регульованих підлокітників для зняття навантаження з плечового поясу під час роботи з клавіатурою та мишею.

– Розміщення відеомонітора: Екран монітора повинен знаходитися на відстані 600 – 700 мм від очей користувача (але не ближче 500 мм). Напрямок погляду повинен бути перпендикулярним до центру екрана, а кут зору від горизонталі до центру екрана має становити 15 – 20 градусмів. Це забезпечує мінімальну напругу м'язів шиї.

Вплив кольору на покращення умов праці та підвищення продуктивності є науково доведеним фактом. Рекомендації щодо естетичного оформлення інтер’єру виробничого приміщення базуються на використанні спокійної, малонасиченої кольорової гами. Для фарбування стін та стелі комп’ютерних

залів доцільно використовувати світлі відтінки жовтого, зеленого, бежевого або блакитного кольорів.

Світло-зелений колір стін знижує внутрішньоочний тиск, заспокоює нервову систему та покращує зорове сприйняття програмного тексту. Стеля фарбується в білий колір з високим коефіцієнтом відбиття (≥0,7) для забезпечення рівномірного розсіяного світла в приміщенні. Озеленення інтер'єру за допомогою кімнатних рослин сприяє психофізіологічному розвантаженню та покращенню мікроклімату завдяки збагаченню повітря киснем та регуляції його вологості.

4.5 Організація режимів праці, відпочинку та проведення інструктажів з охорони праці

Висока напруженість праці програмістів, зумовлена тривалою роботою за монітором, вимагає впровадження раціонального режиму праці та відпочинку для запобігання перевтомі, збереження здоров'я та високої працездатності працівників. Відповідно до НПАОП 0.00-7.15-18, загальний час безперервної роботи з екранними пристроями протягом робочого дня не повинен перевищувати 4 годин за умови організації регламентованих перерв.

Регламентовані перерви для відпочинку встановлюються залежно від тривалості робочої зміни та характеру праці[37]:

– При 8-годинній робочій зміні перерви тривалістю 10–15 хвилин організують через кожні дві години роботи. Під час перерв працівникам рекомендується залишати робоче місце, виконувати комплекси гімнастичних вправ для очей та опорно-рухового апарату.

– Психофізіологічне розвантаження для працівників включає відвідування спеціально обладнаних кімнат відпочинку, де транслюється заспокійлива музика, встановлено ергономічні м'які крісла та підтримується оптимальне фітоорганічне середовище. Це дозволяє швидко зняти нервову напругу та відновити працездатність кори головного мозку.

Працівники комп'ютерної сфери підлягають обов'язковим профілактичним медичним оглядам (попереднім під час прийняття на роботу та періодичним – один раз на два роки) з обов'язковим залученням лікарів: офтальмолога, невропатолога та терапевта для раннього виявлення професійних захворювань (короткозорість, синдром зап'ястного каналу).

4.6 Висновди до розділу 4

У четвертому розділі кваліфікаційної роботи проведено комплексний аналіз безпеки життєдіяльності розробника ігрових застосунків у межах системи «людина – комп’ютер – середовище існування». Під час дослідження було ідентифіковано ключові шкідливі виробничі чинники: фізичні (електромагнітні поля, акустичний шум та надлишкове тепловиділення від потужних графічних станцій) та психофізіологічні (розумове й зорове перенапруження, тривала статична поза та гіподинамія).

З метою мінімізації негативного впливу цих факторів та запобігання професійним захворюванням обґрунтовано комплекс нормативних заходів відповідно до чинного законодавства України (НПАОП 0.00-7.15-18 та ДСанПін 3.3.2.007-98). Систематизовано санітарно-гігієнічні вимоги до робочої зони (площа $\geq 6 м^2$, рівень шуму до 50дБА, освітленість столу 300-500лк, визначено ергономічні параметри меблів і запропоновано рекомендації щодо заспокійливого колірного оформлення інтер'єру. Також сформовано раціональний режим праці й відпочинку, що передбачає введення 10–15 хвилинних перерв кожні дві години, та регламентовано структуру інструктажів з охорони праці.

Реалізація запропонованих рішень забезпечує створення безпечних і комфортних умов праці, мінімізує ризики виробничого травматизму та ефективно підтримує високу працездатність ІТ-фахівця протягом усього процесу проектування.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи розроблено тривимірну комп'ютерну гру у жанрі виживання «Feral Island» на базі сучасного ігрового рушія Unreal Engine 5. Підведемо підсумки виконання поставлених задач відповідно до структури дослідження.

В першому розділі кваліфікаційної роботи освітнього рівня «Бакалавр»:

– проаналізовано стан сучасної індустрії відеоігор у жанрі Survival та детально досліджено ринкові аналоги (The Forest, Ark: Survival Evolved), що дозволило виявити їхні критичні технічні недоліки в аспектах рендерингу геометрії, динамічного освітлення та обмеженості логіки штучного інтелекту.

– Розглянуто архітектурні особливості та інструментарій сучасних ігрових рушіїв Godot, Unity та Unreal Engine 5.

– Обґрунтовано інженерну доцільність вибору рушія Unreal Engine 5 за рахунок нативності систем віртуалізованої геометрії Nanite та глобального динамічного освітлення Lumen, які вирішують виявлені в аналогах проблеми з продуктивністю.

– Сформовано концептуальний базис та дизайн-документ власного проєкту за методологічною моделлю MDA (Mechanics, Dynamics, Aesthetics), визначивши межі ігрового процесу та естетичну атмосферу.

В другому розділі кваліфікаційної роботи:

– Досліджено модульну архітектуру побудови ігрових систем та принципи об'єктно-орієнтованого проєктування в середовищі UE5.

– Обґрунтовано використання системи візуального скриптингу Blueprints як основного інструменту реалізації ігрової логіки, що забезпечує високу швидкість прототипування та гнучкість модифікації компонентів.

– Сформовано архітектуру взаємодії базових підсистем та спроєктовано ергономічний графічний інтерфейс користувача (HUD) для моніторингу життєдіяльності персонажа в реальному часі.

В третьому розділі кваліфікаційної роботи:

– Розроблено та програмно реалізовано систему виживання персонажа, що включає динамічний прорахунок взаємопов'язаних метрик: здоров'я, голоду, спраги та витривалості.

– Запропоновано та впроваджено архітектурне рішення для системи інвентарю та крафту, яке базується на компонентному підході та дозволяє гнучко оперувати типами ресурсів і створювати нові предмети.

– Спроєктовано та реалізовано штучний інтелект ворожих персонажів на основі скінченних автоматів (FSM)

– Протестовано створений прототип гри за допомогою ручного функціонального тестування геймплейних механік та інструментів профілювання UE5. У результаті налагодження виявлено та усунуто дефектів логіки взаємодії інтерфейсу, а заміри продуктивності підтвердили стабільну частоту кадрів при оптимізованому навантаженні на CPU та ОЗП.

Перспективи подальшого розвитку ігрового проєкту:

– Інтеграція кооперативного мультиплеєру за допомогою мережевої реплікації рушія (UE5 Network Replication).

– Реалізація алгоритмів процедурної генерації ландшафту, рослинності та точок спавну ресурсів для підвищення реграбельності.

– Створення комплексної системи збереження та завантаження ігрового прогресу (SaveGame System).

– Додавання механіки повноцінного будівництва захисних споруд із модульною прив'язкою об'єктів (Building System).

– Розширення ШІ супротивників через додавання групової поведінки, систем пошуку укриттів та тактичного відступу.

– Оптимізація графічних профілів та підготовка фінальних збірок (Packaging) під консольні платформи поточного покоління.

У розділі «Безпека життєдіяльності, основи охорони праці» проведено комплексний аналіз умов роботи інженера-програміста та ідентифіковано

потенційні шкідливі й небезпечні чинники виробничого середовища, зокрема зорову напругу та статичне навантаження.

Висвітлено нормативні вимоги до організації та ергономіки робочого місця, параметрів мікроклімату приміщення з ЕОМ; виконано практичний інженерний розрахунок оптимального штучного освітлення методом коефіцієнта використання світлового потоку, а також обґрунтовано заходи із забезпечення електро- та пожежної безпеки.

Таким чином, усі поставлені задачі виконано. Розроблений прототип комп'ютерної гри «Feral Island» є функціональним, оптимізованим та готовим до подальшого масштабування до рівня повноцінного комерційного інді-проєкту.

ПЕРЕЛІК ДЖЕРЕЛ

1 Антонов В. М. Проєктування та розробка комп'ютерних ігор [Текст]: навч. посібник / В. М. Антонов. – К. : ВПЦ "Київський університет", 2021. – 184 с.

2 Бублик В. В. Об'єктно-орієнтоване програмування [Текст]: підручник / В. В. Бублик. – К. : ІТ-Книга, 2019. – 432 с.

3 Голощук С. А. Застосування технологій штучного інтелекту в архітектурі ігрових систем [Текст]: стаття / С. А. Голощук, О. В. Малиновський - Вісник Національного технічного університету. – 2023. – № 2 (14). – С. 45–52.

4 Ковальов Ю. М. Методи оптимізації графічного контенту в ігрових рушіях нового покоління [Текст]: стаття / Ю. М. Ковальов // Технічні науки та технології. – 2024. – № 1 (35). – С. 88–95.

5 Лавров С. М. Програмування штучного інтелекту на основі скінченних автоматів [Текст]: стаття / С. М. Лавров // Комп'ютерні науки та інженерія. – 2022. – Том 10, № 4. – С. 112–119.

6 Мельник А. О. Архітектура комп'ютерних ігор та рушіїв візуалізації [Текст]: монографія / А. О. Мельник. – Львів : Видавництво Львівської політехніки, 2021. – 310 с.

7 Романюк О. В. Комп’ютерна графіка [Текст]: підручник / О. В. Романюк, О. В. Савицький. – Вінниця : ВНТУ, 2020. – 344 с.

8 Сидоренко О. П. Проєткування інтерфейсів користувача для ігрових додатків [Текст]: стаття / О. П. Сидоренко // Комп'ютерно-інтегровані технології. – 2023. – № 48. – С. 102–109.

9 Шелл Д. Мистецтво геймдизайну: Книга концептів [Текст] / Джессі Шелл ; пер. з англ. – К. : ТОВ "Діалектика", 2022. – 520 с.

10 Adams E. Fundamentals of Game Design [Текст] / Ernest Adams. – New York : New Riders, 2014. – 576 p.

11 Buckland M. Programming Game AI by Example [Текст] / Mat Buckland. – Plano : Wordware Publishing, 2005. – 520 p.

12 Eberly D. H. 3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics [Текст] / David H. Eberly. – Boca Raton : CRC Press, 2020. – 1040 p.

13 Gregory J. Game Engine Architecture [Текст] / Jason Gregory. – Boca Raton : CRC Press, 2018. – 1238 p.

14 Hunicke R. MDA: A formal approach to game design and game research [Текст]: workshop paper / R. Hunicke, M. LeBlanc, R. Zubek. – San Jose : AAAI Workshop, 2004. – 5 p.

15 Looman T. Mastering Unreal Engine: A Comprehensive Guide to Creating Games [Текст] / Tommy Looman. – Boca Raton : CRC Press, 2022. – 482 p.

16 Millington I. AI for Games [Текст] / Ian Millington. – Boca Raton : CRC Press, 2019. – 1016 p.

17 Nystrom R. Game Programming Patterns [Текст] / Robert Nystrom. – California : Genever Benning, 2014. – 354 p.

18 Rabin S. Game AI Pro 360: Guide to State Management and Behavior [Текст] / Steve Rabin. – Boca Raton : CRC Press, 2019. – 280 p.

19 Unreal Engine 5 Documentation [Електронний ресурс]. – Epic Games Developer Portal, 2026. – Режим доступу: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-0-documentation

20 Nanite Virtualized Geometry Overview [Електронний ресурс]. – Unreal Engine Documentation, 2026. – Режим доступу: https://dev.epicgames.com/documentation/en-us/unreal-engine/nanite-virtualized-geometry-in-unreal-engine

21 Lumen Global Illumination and Reflections [Електронний ресурс]. – Unreal Engine Documentation, 2026. – Режим доступу: https://dev.epicgames.com/documentation/en-us/unreal-engine/lumen-global-illumination-and-reflections-in-unreal-engine

22 Artificial Intelligence and Behavior Trees in UE5 [Електронний ресурс]. – Epic Games Resources, 2026. – Режим доступу:

https://dev.epicgames.com/documentation/en-us/unreal-engine/artificial-intelligence-in-unreal-engine

23 Blueprints Visual Scripting system guide [Електронний ресурс]. – Unreal Engine Documentation, 2026. – Режим доступу: https://dev.epicgames.com/documentation/en-us/unreal-engine/blueprints-visual-scripting-in-unreal-engine

24 Enhanced Input System in Unreal Engine 5 [Електронний ресурс]. – Epic Games Developer Portal, 2026. – Режим доступу: https://dev.epicgames.com/documentation/en-us/unreal-engine/enhanced-input-in-unreal-engine

25 Game UI and Common UI development framework [Електронний ресурс]. – Epic Games Documentation, 2026. – Режим доступу: https://dev.epicgames.com/documentation/en-us/unreal-engine/common-ui-framework-in-unreal-engine

26 UI Optimization Guidelines for Unreal Engine [Електронний ресурс]. – Epic Games Developer Section, 2026. – Режим доступу: https://dev.epicgames.com/documentation/en-us/unreal-engine/optimization-guidelines-for-umd-in-unreal-engine

27 Performance Profiling and Optimization inside Unreal Engine [Електронний ресурс]. – Unreal Engine Guides, 2026. – Режим доступу: https://dev.epicgames.com/documentation/en-us/unreal-engine/testing-and-profiling-in-unreal-engine

28 World Partitioning System in Open World Games [Електронний ресурс]. – Epic Games Tech Blog, 2026. – Режим доступу: https://dev.epicgames.com/documentation/en-us/unreal-engine/world-partition-in-unreal-engine

29 AI Perception Component Implementation Guide [Електронний ресурс]. – Unreal Engine AI Section, 2026. – Режим доступу: https://dev.epicgames.com/documentation/en-us/unreal-engine/ai-perception-in-unreal-engine

30 Procedural Content Generation (PCG) Framework in UE5 [Електронний ресурс]. – Epic Games Resources, 2026. – Режим доступу: https://dev.epicgames.com/documentation/en-us/unreal-engine/procedural-content-generation-in-unreal-engine

31 Unreal Insights Performance Profiling Tool [Електронний ресурс]. – Unreal Engine Documentation, 2026. – Режим доступу: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-insights-in-unreal-engine

32 Game Framework Architecture: GameMode and GameState [Електронний ресурс]. – Epic Games Developer Portal, 2026. – Режим доступу: https://dev.epicgames.com/documentation/en-us/unreal-engine/game-framework-in-unreal-engine

33 Gameplay Tags System for Inventory Management [Електронний ресурс]. – Unreal Engine Architecture Guides, 2026. – Режим доступу: https://dev.epicgames.com/documentation/en-us/unreal-engine/gameplay-tags-in-unreal-engine

34 The Forest Technical Review and Game Architecture [Електронний ресурс]. – Gamasutra: Game Developer Resource, 2026. – Режим доступу: https://www.gamedeveloper.com/design/the-forest-mechanics-analysis

35 ARK: Survival Evolved Engine Performance Breakdown [Електронний ресурс]. – Digital Foundry Hardware Analysis, 2026. – Режим доступу: https://www.eurogamer.net/digitalfoundry-ark-survival-evolved-performance-analysis

36 Желібо Є.П. Безпека життєдіяльності : підручник / Є. П. Желібо, В. В. Зацарний. – Київ : Каравела, 2023. – 344 с.

37 Жидецький В.Ц. Охорона праці користувачів комп’ютерів : підручник / В. Ц. Жидецький. – Львів : Афіша, 2020. – 176 с.

38 Шимчук, Г. В., Назаревич, О. Б., Литвиненко, Я. В., Готович, В. А., Никитюк, В. В., & Боднарчук, І. О. (2025). Грід-системи та технології хмарних обчислень. Навчальний посібник для здобувачів освітнього рівня «магістр»

спеціальностей: F3 «Комп’ютерні науки», F6 «Інформаційні системи та технології».

39 Leshchyshyn, Y., Scherbak, L., Nazarevych, O., Gotovych, V., Tymkiv, P., & Shymchuk, G. (2019, May). Multicomponent Model of the Heart Rate Variability Change-point. In 2019 IEEE XVth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH) (pp. 110-113). IEEE.

40 Lytvynenko, I., Lupenko, S., Nazarevych, O., Shymchuk, G., & Hotovych, V. (2021, September). Mathematical model of gas consumption process in the form of cyclic random process. In 2021 IEEE 16th International Conference on Computer Sciences and Information Technologies (CSIT) (Vol. 1, pp. 232-235). IEEE.

41 Kozlovskyi, V., Balanyuk, Y., Martyniuk, H., Nazarevych, O., Scherbak, L., & Shymchuk, G. (2022, April). Information Technology for Estimating City Gas Consumption During the Year. In 2022 International Conference on Smart Information Systems and Technologies (SIST), Nur-Sultan, Kazakhstan (pp. 1-4).

42 Lytvynenko, I., Lupenko, S., Kunanets, N., Nazarevych, O., Shymchuk, G., & Hotovych, V. (2021). Simulation of gas consumption process based on the mathematical model in the form of cyclic random process considering the scale factors. In 1st International Workshop on Information Technologies: Theoretical and Applied Problems, ITTAP (Vol. 2021).

ДОДАТКИ

ДОДАТОК А - заголовковий файл компонента характеристик

Лістинг А.1 - SurvivalAttributeComponent.h

```
#pragma once

#include "CoreMinimal.h"
#include "Components/ActorComponent.h"
#include "SurvivalAttributeComponent.generated.h"

// Структура для елемента інвентарю (індексація масиву починається з 0)
USTRUCT(BlueprintType)
struct FInventoryItem
{
    GENERATED_BODY()

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Inventory")
    FName ItemID;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Inventory")
    int32 Quantity;

    FInventoryItem() : ItemID(NAME_None), Quantity(0) {}
};

UCLASS(ClassGroup=(Custom), meta=(BlueprintSpawnableComponent) )
class SURVIVALGAME_API USurvivalAttributeComponent : public UActorComponent
{
    GENERATED_BODY()
public:
```

```
        USurvivalAttributeComponent();

    protected:
        virtual void BeginPlay() override;

    public:
        virtual void TickComponent(float DeltaTime, ELevelTick
TickType, FActorComponentTickFunction* ThisTickFunction) override;

        UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =
"Attributes|Health")
        float CurrentHealth;

        UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =
"Attributes|Hunger")
        float CurrentHunger;

        UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =
"Attributes|Thirst")
        float CurrentThirst;

        UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =
"Attributes|Stamina")
        float CurrentStamina;

        UPROPERTY(EditAnywhere, BlueprintReadOnly, Category =
"Attributes|Config")
        float MaxAttributeValue;

        UPROPERTY(EditAnywhere, BlueprintReadOnly, Category =
"Attributes|Config")
        float HungerDecayRate;

        UPROPERTY(EditAnywhere, BlueprintReadOnly, Category =
"Attributes|Config")
```

```
        float ThirstDecayRate;

        UFUNCTION(BlueprintCallable, Category = "Attributes|Actions")
        void ConsumeFood(float Amount);

        UFUNCTION(BlueprintCallable, Category = "Attributes|Actions")
        void ConsumeWater(float Amount);

        UPROPERTY(BlueprintReadOnly, Category = "Inventory")
        TArray<FInventoryItem> PlayerInventory;

        UFUNCTION(BlueprintCallable, Category = "Inventory")
        void UseItemFromInventory(int32 ItemIndex);

    private:
        void HandleAttributeDecay(float DeltaTime);
        void ApplyStarvationDamage(float DeltaTime);
    };
```

ДОДАТОК Б - Файл реалізації логіки компонента

Лістинг Б.1 - SurvivalAttributeComponent.cpp

```
#include "SurvivalAttributeComponent.h"
#include "GameFramework/Actor.h"
#include "Kismet/KismetMathLibrary.h"

USurvivalAttributeComponent::USurvivalAttributeComponent()
{
    PrimaryComponentTick.bCanEverTick = true;

    MaxAttributeValue = 100.0f;
    HungerDecayRate = 0.05f;
    ThirstDecayRate = 0.08f;

    CurrentHealth = MaxAttributeValue;
    CurrentHunger = MaxAttributeValue;
    CurrentThirst = MaxAttributeValue;
    CurrentStamina = MaxAttributeValue;
}

void USurvivalAttributeComponent::BeginPlay()
{
    Super::BeginPlay();
}

Void         USurvivalAttributeComponent::TickComponent(float
DeltaTime,   ELevelTick   TickType,   FActorComponentTickFunction*
ThisTickFunction)
{
    Super::TickComponent(DeltaTime,TickType,
ThisTickFunction);

    HandleAttributeDecay(DeltaTime);
```

```
        if (CurrentHunger <= 0.0f || CurrentThirst <= 0.0f)
        {
            ApplyStarvationDamage(DeltaTime);
        }
    }

    void  USurvivalAttributeComponent::HandleAttributeDecay(float
DeltaTime)
    {
        CurrentHunger     =     FMath::Clamp(CurrentHunger     -
(HungerDecayRate * DeltaTime), 0.0f, MaxAttributeValue);
        CurrentThirst     =     FMath::Clamp(CurrentThirst     -
(ThirstDecayRate * DeltaTime), 0.0f, MaxAttributeValue);
    }

    void USurvivalAttributeComponent::ApplyStarvationDamage(float
DeltaTime)
    {
        float DamageAmount = 1.0f * DeltaTime; // 1 одиниця шкоди
на секунду
        CurrentHealth = FMath::Clamp(CurrentHealth - DamageAmount,
0.0f, MaxAttributeValue);
    }

    void USurvivalAttributeComponent::ConsumeFood(float Amount)
    {
        CurrentHunger = FMath::Clamp(CurrentHunger + Amount, 0.0f,
MaxAttributeValue);
    }

    void USurvivalAttributeComponent::ConsumeWater(float Amount)
    {
        CurrentThirst = FMath::Clamp(CurrentThirst + Amount, 0.0f,
MaxAttributeValue);
```

```
}

void USurvivalAttributeComponent::UseItemFromInventory(int32
ItemIndex)
{
    if (PlayerInventory.IsValidIndex(ItemIndex))
    {
        FInventoryItem&          TargetItem          =
PlayerInventory[ItemIndex];

        if (TargetItem.Quantity > 0)
        {
            if (TargetItem.ItemID.IsEqual(FName("Apple")))
            {
                ConsumeFood(15.0f);
                TargetItem.Quantity--;
            }
            Else                                          if
(TargetItem.ItemID.IsEqual(FName("WaterBottle")))
            {
                ConsumeWater(25.0f);
                TargetItem.Quantity--;
            }

            if (TargetItem.Quantity <= 0)
            {
                PlayerInventory.RemoveAt(ItemIndex);
            }
        }
    }
```