ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка 3D комп'ютерної гри в жанрі виживання»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Павло ПРОКОПЧУК
(підпис)

Виконав: здобувач вищої освіти групи ПД-43

Павло ПРОКОПЧУК

Керівник: Оксана ЗОЛОТУХІНА
канд. техн. наук, доц.

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Прокопчуку Павлу Сергійовичу

1. Тема кваліфікаційної роботи: «Розробка 3D комп'ютерної гри в жанрі виживання»
керівник кваліфікаційної роботи канд. техн. наук, доцент Оксана ЗОЛОТУХІНА, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про розробку 3D-ігор жанру survival, документація ігрового рушія Unity 6, бібліотек Mirror Networking та FizzySteamworks, наукові публікації з питань штучного інтелекту в комп'ютерних іграх.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної області та постановка задачі.
2. Проєктування програмного забезпечення гри TwilightTails.
3. Реалізація програмного забезпечення.
4. Тестування та аналіз результатів.

5. Перелік графічного матеріалу: *презентація*

1. Порівняльний аналіз survival-аналогів.
2. Функціональні та нефункціональні вимоги.
3. Технологічний стек реалізації.
4. Діаграма прецедентів (use-case).
5. Архітектура застосунку.
6. Діаграма послідовності взаємодії гравця з оточенням.
7. UML-діаграми класів підсистем.
8. Екранні форми ігрового процесу.

6. Дата видачі завдання «23» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	23.02-28.02.2026	
2	Аналіз та дослідження існуючих аналогів	01.03-07.03.2026	
3	Огляд підходів до симуляції природного середовища та штучного інтелекту в іграх жанру виживання	08.03-21.03.2026	
4	Проєктування програмного забезпечення гри TwilightTails	22.03-04.04.2026	
5	Програмна реалізація гри	05.04-25.04.2026	
6	Тестування гри	26.04-02.05.2026	
7	Оформлення роботи: вступ, висновки, реферат	03.05-08.05.2026	
8	Розробка демонстраційних матеріалів	09.05-10.05.2026	
9	Попередній захист роботи	11.05-22.05.2026	

Здобувач вищої освіти _____ Павло ПРОКОПЧУК

Керівник
кваліфікаційної роботи _____ Оксана ЗОЛОТУХІНА

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 60 стор., 6 табл., 20 рис., 28 джерел, 3 додатки.

Мета роботи - підвищення занурення гравця у віртуальне середовище survival-гри за рахунок розробки 3D-додатку TwilightTails з реалістичним штучним інтелектом NPC-тварин, динамічним світом і кооперативним мультиплеєром через Steam.

Об'єкт дослідження - процес створення інтерактивного 3D-середовища комп'ютерної гри у жанрі виживання.

Предмет дослідження - програмне забезпечення 3D-гри TwilightTails у жанрі виживання, реалізоване на ігровому рушії Unity з підтримкою кооперативного режиму гри.

Завдання дослідження: проаналізувати сучасний стан ринку та виокремити характерні риси жанру виживання; виконати порівняльний аналіз провідних програмних аналогів та ігрових рушіїв і обґрунтувати технологічний стек; спроєктувати архітектуру застосунку і структуру підсистем (контролер персонажа, штучний інтелект, pickup, будівництво, цикл день/ніч, збереження); реалізувати програмне забезпечення гри TwilightTails у форматі MVP; провести функціональне тестування і вимірювання продуктивності.

Методи дослідження: системний аналіз для дослідження предметної області та порівняння аналогів; об'єктно-орієнтоване проєктування для побудови моделей підсистем; експериментальний підхід для вимірювання продуктивності у Unity Profiler; тестування методом «чорної скриньки» для перевірки функціональних вимог.

У результаті роботи розроблено програмне забезпечення гри TwilightTails у форматі MVP (мінімально життєздатного продукту) на ігровому рушії Unity 6 мовою C#. Реалізовано керування персонажем-єнотом у відкритому тривимірному середовищі з підтримкою скелетної анімації та інверсної кінематики лап; інтерактивну систему pickup і носіння одного предмета у передніх лапах; режим вільного будівництва з попереднім переглядом, обертанням і фіксацією споруд; штучний інтелект шести видів тварин (єнот-NPC, заєць, лисиця, вовк, ведмідь, жаба) на основі поведінкових дерев Behavior Designer з диференційованими стратегіями для хижаків і здобичі; добовий цикл з впливом на поведінку фауни і параметри середовища; автоматичне збереження стану світу у форматі JSON у каталозі persistentDataPath; локалізацію інтерфейсу українською та англійською

мовами на базі пакета Unity Localization. Загальний обсяг власного коду - близько 4500 рядків у 86 скриптах C#, доповнених інтеграцією комерційних пакетів Malbers Animal Controller, Behavior Designer, BioIK, Enviro 3 та The Vegetation Engine. Технологічну ефективність обраного стека підтверджено серією навантажувальних тестів у трьох сценаріях (денна гра, активний рух з присутніми тваринами, нічна сцена з дощем і вітром) на двох референсних апаратних конфігураціях. За результатами продуктивностного тестування середня кадрова частота становить від 45 до 60 кадрів за секунду на ноутбучному стенді нижчого класу і стабільні 60 кадрів за секунду на стенді вищого класу.

Сферою застосування результатів є подальший розвиток гри з виходом на платформу Steam, а також використання реалізованих модулів і архітектурних рішень у навчальних курсах з ігрової розробки на ігровому рушії Unity та як референсного прикладу інтеграції поведінкових дерев у самостійних студентських проєктах. Окремий потенціал результатів полягає у формуванні бази для портування проєкту на платформи macOS та Linux, розширення на кооперативний онлайн-режим на основі вже інтегрованої архітектурної заготовки Mirror Networking, а також у популяризації жанру survival-ігор з персонажем-твариною на українському ринку незалежної розробки.

КЛЮЧОВІ СЛОВА: КОМП'ЮТЕРНА ГРА, ВИЖИВАННЯ, ВІДКРИТИЙ СВІТ, UNITY, ШТУЧНИЙ ІНТЕЛЕКТ, ПОВЕДІНКОВІ ДЕРЕВА, ЦИКЛ ДЕНЬ-НІЧ, ЛОКАЛІЗАЦІЯ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AI - штучний інтелект (artificial intelligence)
API - програмний інтерфейс додатку (Application Programming Interface)
BT - дерево поведінки (Behavior Tree)
DOI - цифровий ідентифікатор об'єкта (Digital Object Identifier)
FPS - кадрів за секунду (frames per second)
FSM - скінченний автомат (Finite State Machine)
GOAP - планувальник дій на основі цілей (Goal-Oriented Action Planning)
GPU - графічний процесор (Graphics Processing Unit)
GUI - графічний користувацький інтерфейс (Graphical User Interface)
GUID - глобальний унікальний ідентифікатор
HDRP - конвеєр високої якості рендерингу Unity (High Definition Render Pipeline)
HUD - індикатор стану гравця (Heads-Up Display)
ID - ідентифікатор (Identifier)
IK - інверсна кінематика (Inverse Kinematics)
ISBN - міжнародний стандартний книжковий номер (International Standard Book Number)
JSON - текстовий формат обміну даними (JavaScript Object Notation)
LOD - рівень деталізації моделі (Level of Detail)
MVC - архітектурний патерн «Модель-Подання-Контролер» (Model-View-Controller)
MVP - мінімально життєздатний продукт (Minimum Viable Product)
NavMesh - навігаційна сітка
NPC - неігровий персонаж (Non-Player Character)
P2P - взаємодія типу «рівний з рівним» (peer-to-peer)
PC - персональний комп'ютер (Personal Computer)
RPC - віддалений виклик процедури (Remote Procedure Call)
SDK - набір засобів розробника (Software Development Kit)
SOLID - п'ять принципів об'єктно-орієнтованого проєктування
SSD - твердотільний накопичувач (Solid State Drive)
TVE - The Vegetation Engine
UI - користувацький інтерфейс (User Interface)
UML - уніфікована мова моделювання (Unified Modeling Language)
URL - уніфікований локатор ресурсу (Uniform Resource Locator)
URP - універсальний конвеєр рендерингу Unity (Universal Render Pipeline)
UX - користувацький досвід (User Experience)
WASD - клавіші W, A, S, D для керування персонажем у комп'ютерних іграх
3D - three-dimensional, тривимірний
ІПЗ - Інженерія програмного забезпечення
ДСТУ - Державний стандарт України
ДУІКТ - Державний університет інформаційно-комунікаційних технологій
ОС - операційна система
ПЗ - програмне забезпечення

ВСТУП

Актуальність теми. За даними аналітичного агентства Newzoo, у 2024 році світовий ринок комп'ютерних ігор перевищив 184 млрд доларів США. Близько чверті цього обсягу формує сегмент персональних комп'ютерів. Серед драйверів зростання дослідники окремо виділяють розквіт незалежної (інді) розробки. У звіті аналітичної агенції GameDiscoverCo за 2024 рік зафіксовано, що інді-сегмент на платформі Steam дав понад 35 % нових ігор року, причому продажі п'яти найвідоміших одиночних survival-проєктів перетнули позначку в один мільйон копій.

Жанр виживання у цьому процесі займає особливе місце. Ігри The Forest [3], Rust [4], Subnautica [5], ARK: Survival Evolved [6] і Green Hell [7] поєднують механіки збору ресурсів, харчування, будівництва й боротьби з ворожим середовищем. Аналіз історії розробки декількох успішних інді-проєктів цього жанру [19] свідчить, що навіть з обмеженим бюджетом і малою командою можна досягти комерційного успіху у survival-сегменті. Це робить survival привабливим для невеликих команд та одиночних розробників.

Технічна реалізація таких проєктів пов'язана зі значною кількістю інженерних задач. Серед них - проєктування ландшафту відкритого світу, побудова штучного інтелекту тварин з різною поведінкою для хижаків і здобичі, симуляція динамічного дня й погодних умов, підсистема збереження прогресу, система pickup-взаємодії з оточенням, режим вільного будівництва, інтерактивний туторіал і підтримка декількох мов інтерфейсу. Розв'язання цих задач у форматі дипломного проєкту дозволяє продемонструвати володіння кількома галузями інженерії програмного забезпечення - об'єктно-орієнтованим проєктуванням, ШІ, оптимізацією рендерингу, серіалізацією даних.

Виходячи з наведеного, тема розробки тривимірної одиночної гри жанру виживання є актуальною і має практичну цінність.

Мета роботи - підвищення занурення гравця у віртуальне середовище survival-гри за рахунок розробки 3D-додатку TwilightTails з реалістичним штучним інтелектом NPC-тварин, динамічним світом і кооперативним мультиплеєром через Steam.

Об'єкт дослідження - процес створення інтерактивного 3D-середовища комп'ютерної гри у жанрі виживання.

Предмет дослідження - програмне забезпечення 3D-гри TwilightTails у жанрі виживання, реалізоване на ігровому рушії Unity з підтримкою кооперативного режиму гри.

Завдання дослідження. Для досягнення мети роботи поставлено такі завдання:

1) проаналізувати сучасний стан ринку комп'ютерних ігор жанру виживання, визначити характерні риси жанру і виокремити вимоги до власної реалізації;

2) виконати огляд провідних програмних аналогів і порівняти ігрові рушії для обґрунтованого вибору технологічного стека;

3) розглянути підходи до побудови штучного інтелекту в іграх і обрати найбільш доцільну модель для застосування в роботі;

4) спроєктувати загальну архітектуру застосунку і структуру окремих підсистем - контролера персонажа, штучного інтелекту, системи pickup, режиму будівництва, циклу день/ніч, збереження стану;

5) реалізувати програмне забезпечення гри TwilightTails у форматі MVP - мінімально життєздатного продукту з основними механіками жанру;

6) провести функціональне тестування і вимірювання продуктивності розробленого програмного забезпечення;

Методи дослідження. У роботі застосовано системний аналіз для дослідження предметної області та порівняння аналогів; об'єктно-орієнтоване проєктування для побудови моделей підсистем; експериментальний підхід для вимірювання продуктивності у Unity Profiler; тестування методом «чорної скриньки» для перевірки функціональних вимог.

Практичне значення отриманих результатів. Розроблене програмне забезпечення реалізує ключові механіки жанру виживання: керування персонажем-єнотом у 3D-середовищі, систему pickup і носіння предметів, харчування, полювання на дрібну дичину, режим вільного будівництва, добовий цикл з впливом на поведінку фауни, штучний інтелект тварин шести видів на основі дерев поведінки (Behavior Tree, далі - BT), інтерактивний туторіал, локалізований інтерфейс шістьма мовами. Розроблені модулі - підсистема BT-задач поведінки тварин, система збереження стану у форматі JSON, контролер циклу день/ніч, режим вільного будівництва - є самодостатніми компонентами, які можуть бути перевикористані в подальших проєктах ігрового програмного забезпечення на ігровому рушії Unity.

Структура та обсяг роботи. Робота складається зі вступу, чотирьох розділів, висновків, переліку використаних джерел та додатків.

У першому розділі проаналізовано предметну область: розглянуто сучасний стан жанру виживання у комп'ютерних іграх, виконано порівняльний аналіз п'яти провідних програмних аналогів, зіставлено ігрові рушії та готові рішення для симуляції природного середовища, розглянуто підходи до побудови штучного інтелекту в іграх. На основі проведеного аналізу сформульовано вимоги до власної реалізації.

У другому розділі представлено результати проєктування: загальну архітектуру застосунку, опис контролера персонажа, модель ігрового світу, структуру підсистем штучного інтелекту, збереження стану та користувацького інтерфейсу.

У третьому розділі описано реалізацію програмного забезпечення: обґрунтовано вибір інструментальних засобів та наведено реалізацію персонажа-єнота, системи pickup і режиму будівництва, підсистеми штучного інтелекту тварин, циклу день і ніч, підсистеми збереження стану та локалізації.

У четвертому розділі наведено результати функціонального та продуктивностного тестування розробленого програмного забезпечення, описано

конфігурацію тестового стенду, перелік виконаних тестових сценаріїв та отримані показники кадрової частоти на двох референсних апаратних конфігураціях.

Апробацію основних результатів роботи проведено на двох всеукраїнських науково-технічних конференціях у Державному університеті інформаційно-комунікаційних технологій у листопаді 2025 та квітні 2026 років; за матеріалами роботи опубліковано дві тези доповідей у співавторстві з науковим керівником. Перша теза присвячена аналізу ключових особливостей геймплею комп'ютерних ігор жанру виживання та порівняльному дослідженню провідних аналогів сегменту. Друга теза висвітлює розробку системи штучного інтелекту NPC-тварин у тривимірній грі на виживання на ігровому рушії Unity, описує застосовану архітектуру поведінкових дерев та диференційовані стратегії поведінки для хижаків і здобичі.

Практичне значення розробленого програмного забезпечення полягає у створенні працездатного MVP комп'ютерної гри жанру виживання з повною українською локалізацією, який може бути використаний як основа для подальшої комерційної гри з виходом на платформу Steam. Окрім того, реалізовані модулі pickup, режиму будівництва, поведінкових дерев тварин і циклу день/ніч сформовано як перевикористовувані компоненти і можуть бути взяті за основу у самостійних навчальних проєктах студентів старших курсів спеціальностей з інженерії програмного забезпечення та комп'ютерної графіки. Отриманий під час виконання роботи технологічний досвід інтеграції стороннього інструментарію (Malbers Animal Controller, Behavior Designer, Enviro 3, The Vegetation Engine) у єдину архітектурну систему є цінним і для подальшої професійної діяльності розробника.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Жанр виживання у сучасній ігровій індустрії

Жанр виживання (survival) як окреме явище в комп'ютерних іграх формувався поступово. Дослідники [14] пов'язують його остаточне виокремлення з виходом ігор Don't Starve у 2013 році та The Forest у 2014-му. Згодом до жанру долучилися Rust [4], ARK: Survival Evolved [6], Subnautica [5], Valheim, Green Hell [7], а в 2020-х роках Sons of the Forest, Palworld та інші. За даними аналітичних оглядів [8], жанр перебуває у стійкій фазі зростання популярності протягом останнього десятиліття.

Аналіз сорока найпопулярніших ігор цього жанру за статистикою сервісу Steam Store [9] дозволяє виокремити такі типові ознаки:

1) обмеженість ресурсів - гравець на старті має мінімальний набір предметів і змушений видобувати все необхідне самостійно;

2) обов'язкові процеси життєдіяльності персонажа - голод, спрага, втома, температура тіла, що знижуються з часом і вимагають уваги гравця;

3) вороже або нейтральне ігрове середовище - дикі тварини, монстри, погодні умови, що становлять загрозу;

4) наявність крафту та будівництва - можливість створювати предмети і споруди із зібраних ресурсів;

5) відсутність чітко визначеної кінцевої цілі - ігровий процес відкритий, рекордами виступають кількість прожитих днів, обсяг побудованого, кількість здобутих ресурсів.

Помітна тенденція останнього десятиліття - зростання частки одиночних інді-survival з виразним авторським стилем. У звіті GameDiscoverCo за 2024 рік [15] зафіксовано, що серед двадцяти найуспішніших нових релізів жанру на Steam дванадцять є проєктами невеликих команд (до десяти осіб) з фокусом на single-

player або асинхронну взаємодію. Перехід пояснюється насиченістю ринку шумними AAA-проєктами і запитом аудиторії на «тихі», атмосферні переживання.

У літературі [14, 16] виділяють підкласи жанру, які мають значення для побудови власного проєкту:

- hardcore survival (Rust, DayZ): акцент на ворожому середовищі, перманентній смерті, високій складності;

- cosy survival (Don't Starve, Among Trees): атмосферне дослідження, спокійний ритм, увага до естетики;

- crafting-survival (Minecraft, Valheim): акцент на будівництві, прогресії технологій;

- wildlife-survival (Shelter, Endling, TwilightTails): головний герой - тварина, акцент на емоційному вимірі.

Гра TwilightTails, що розробляється в межах цієї роботи, належить до останнього підкласу. Головним персонажем виступає єнот, ігровий світ є європейським лісом з циклом день і ніч, основний акцент зроблено на дослідженні і взаємодії з фауною, а не на змаганні гравців між собою (player versus player, PvP) чи на механіках накопичення ресурсів.

1.2. Огляд програмних аналогів

Для обґрунтованого добору функціональних вимог до власної реалізації проведено огляд п'яти провідних програмних аналогів. Перелік цих аналогів сформовано з огляду на показники продажів у Steam Store [9], спорідненість жанрових механік та доступність відкритих відомостей про технологічний стек.

1.2.1. The Forest

Розробник Endnight Games, дата виходу 2018 рік [3]. Тривимірна гра жанру horror-survival з підтримкою кооперативу до восьми учасників. Реалізована на ігровому рушії Unity. AI-екосистема включає основних агентів-канібалів з груповою поведінкою (розвідка, оточення, переслідування), мутантів-босів та фонових тварин - оленів, кролів, черепах і риб, що формують харчовий ланцюг.

Сильні сторони: атмосфера психологічного жаху, продумана система AI канібалів-противників з груповою тактикою і денно-нічними циклами активності.

Слабкі сторони: складний онбординг, відсутність повноцінного туторіалу, орієнтація на досвідчених гравців. Архітектурне рішення, важливе для запозичення, - система динамічних подій з тваринами та ворогами на основі тригерів дистанції до гравця. Екранна форма гри наведена на рисунку 1.1.

Рис. 1.1 Екранна форма гри The Forest

1.2.2. Rust

Розробник Facepunch Studios, оригінальний реліз 2018 рік [4]. Тривимірна PvP-орієнтована survival-гра з підтримкою до 250 гравців на одному сервері. Реалізована на Unity. AI-екосистема мінімальна: ведмеді, вовки, олені і кабани з простою поведінкою «спокій - переслідування/втеча - атака» без міжвидової взаємодії; основна загроза гравця - інші гравці.

Сильні сторони: масштабна процедурно згенерована карта, глибока система крафту з понад 100 рецептами.

Слабкі сторони: ворожа атмосфера до новачків, спрощена AI-екосистема. Корисне для запозичення - архітектура серверного автохосту через виділені сервери. Екранна форма гри наведена на рисунку 1.2.

Рис. 1.2 Екранна форма гри Rust

1.2.3. Subnautica

Розробник Unknown Worlds Entertainment, реліз 2018 рік [5]. Підводна survival-гра від першої особи. Реалізована на Unity з власним візуальним стеком. AI-екосистема структурована за біомами: дрібні риби-фуражери у мілководді, середні нейтральні види (тімі, гарпуни) у мангровому поясі, агресивні хижаки (ріпер-левіафан, гульпер) у глибоких зонах; усі агенти мають радіуси сприйняття, добові цикли активності та реакцію на джерело світла гравця.

Сильні сторони: глибока атмосфера дослідження, оригінальний підводний сетинг, диференційована за біомами AI-екосистема.

Слабкі сторони: однокористувацька (відсутність кооперативу), повторюваність геймплею у середній частині гри. Корисне для запозичення - система плавної підвантажуваності біомів та LOD-оптимізація для відкритих просторів. Екранна форма гри наведена на рисунку 1.3.

Рис. 1.3 Екранна форма гри Subnautica

1.2.4. ARK: Survival Evolved

Розробник Studio Wildcard, реліз 2017 рік [6]. Survival з елементами одомашнення динозаврів. Підтримує до 70 гравців на сервері. Реалізована на Unreal Engine 4.

Сильні сторони: розширена AI-екосистема з понад 150 видами істот, рольова прогресія.

Слабкі сторони: високі системні вимоги, тривалі сесії, надто складна для нових гравців. Корисне для запозичення - архітектура AI-екосистеми з диференційованою поведінкою хижаків і здобичі. Екранна форма гри наведена на рисунку 1.4.

Рис. 1.4 Екранна форма гри ARK: Survival Evolved

1.2.5. Green Hell

Розробник Creepy Jar, реліз 2019 рік [7]. Survival-симулятор у тропічних джунглях Амазонки. Реалізована на Unity з підтримкою кооперативу до чотирьох учасників. AI-екосистема включає диких тубільців з груповими патрулями, хижаків (ягуар, анаконда, кайман), отруйних змій і павуків, а також дрібну дичину (тапіри, броненосці, риби) - кожен вид має власні зони мешкання у джунглях, ріці чи на узліссі та може стати джерелом ризику захворювання або їжі.

Сильні сторони: висока реалістичність симуляції (хвороби, психічний стан, рани), деталізована флора і фауна з біологічно правдоподібним поділом на ареали.

Слабкі сторони: складність балансу, ризик ранніх фрустрацій новачків. Корисне для запозичення - модель симуляції фізіологічних потреб персонажа з диференційованими параметрами (голод, спрага, температура тіла, психіка). Екранна форма гри наведена на рисунку 1.5.

Рис. 1.5 Екранна форма гри Green Hell

1.2.6. Узагальнений порівняльний аналіз

Зіставлення розглянутих аналогів за єдиною системою критеріїв наведено в таблиці 1.1.

Таблиця 1.1

Порівняння програмних аналогів за ключовими критеріями

Критерій	The Forest	Rust	Subnautica	ARK	Green Hell	TwilightTails
Сетинг	хвойний ліс	відкритий ландшафт	підводний світ	острів динозаврів	джунглі Амазонки	європейський ліс
Складність AI	висока	низька	помірна	дуже висока	помірна	висока (BT, 6 видів)
Будівництво з попереднім переглядом	так	так	ні	так	ні	так
Багатокористувацький режим	кооп до 8	PvP до 250	немає	кооп/PvP до 70	кооп до 4	немає
Інтерактивний туторіал	ні	ні	так	ні	так	так
Українська локалізація	ні	ні	ні	ні	ні	так
Перспектива камери	перша особа	перша особа	перша особа	третя особа	перша особа	третя особа
Протагоніст	людина	людина	людина	людина	людина	тварина (єнот)
Цикл день/ніч з впливом на AI	часткový	ні	ні	часткový	ні	так

Як видно з таблиці 1.1, жоден з п'яти провідних survival-проєктів не пропонує одночасно повного набору рис, бажаних для гри TwilightTails: штучний інтелект тварин на основі поведінкових дерев із залежністю від часу доби, повну українську локалізацію, інтерактивний туторіал, перспективу від третьої особи з персонажем-твариною. Така комбінація рис формує вільну ринкову нішу, у якій гра TwilightTails може стати нішевим продуктом, не вступаючи у пряму конкуренцію з провідними проєктами галузі.

1.3. Порівняння ігрових рушіїв

Сучасний ринок ігрових рушіїв включає декілька великих гравців. У межах роботи розглянуто три найпопулярніші варіанти: Unity, Unreal Engine, Godot.

Unity (компанія Unity Technologies) [11]. Версія Unity 6, випущена у 2024 році, є основним інструментом для незалежних розробників. Близько 60 % мобільних ігор та значна частка PC-ігор індустрії використовують саме цей рушій. Мова програмування скриптів - C#. Має дві основні системи рендерингу: Universal Render Pipeline (URP) для широкої сумісності з обладнанням та High Definition Render Pipeline (HDRP) для фотореалістичного рендерингу. Підтримує велику кількість сторонніх інструментів через сервіс Asset Store.

Unreal Engine 5 (Epic Games). Технічно потужніший, орієнтований на AAA-проєкти. Мова Blueprints (візуальне програмування) та C++. Має кращу графіку «з коробки» (Lumen, Nanite), але вища складність освоєння та вищі системні вимоги. Ліцензія передбачає роялті 5 % з продажів понад 1 млн доларів.

Godot 4. Безкоштовний рушій з відкритим вихідним кодом. Власна мова GDScript, схожа на Python, плюс підтримка C#. Швидко розвивається, але має значно менший пул готових інструментів і обмеженіші засоби 3D-рендерингу.

З огляду на масштаб дипломного проєкту, обмежений час розробки та потребу використовувати готові бібліотеки для тварин, рослинності й мережі, доцільним є вибір Unity 6 [11]. Цей рушій забезпечує найбільш прийнятне співвідношення продуктивності, доступності інструментарію та зрілості екосистеми сторонніх пакетів для жанру виживання. Unreal Engine 5 у межах одиночного проєкту з персонажем-твариною надмірно складний у налаштуванні, потребує значно більших обчислювальних ресурсів для розробки та має нижчу зрілість пакетів для AI поведінкових дерев тварин у порівнянні з Behavior Designer для Unity. Godot 4, попри відкритість і просту мову GDScript, поступається обом конкурентам за якістю готових рішень для рослинності, динамічної погоди та інтеграції зі Steam, що для проєкту з планованим виходом на цю платформу є критично важливим.

Обраний Unity 6 також надає сучасні конвеєри рендерингу URP та HDRP, що дозволяє масштабувати графічну якість відповідно до конфігурації цільового апаратного забезпечення без переписування коду. Окремим аргументом на користь Unity 6 є наявність зрілої екосистеми навчальних матеріалів українською та англійською мовами, активна україномовна спільнота розробників, а також підтвердження ефективності рушія для жанру survival численними успішними релізами на платформі Steam (зокрема, проаналізованими у попередніх підрозділах The Forest, Subnautica, Green Hell).

За результатами розглянутих критеріїв побудовано підсумкову таблицю порівняння трьох рушіїв за вісьмома ключовими ознаками - мовою програмування, типом ліцензії, цільовими платформами, наявністю готових AI-рішень, інтеграцією зі Steam, документацією, обсягом спільноти та порогом входження для нового розробника.

Зіставлення ключових характеристик подано в таблиці 1.2.

Таблиця 1.2

Порівняння ігрових рушіїв

Критерій	Unity 6	Unreal Engine 5	Godot 4
Мова	C#	C++ / Blueprints	GDScript / C#
Ліцензія	безкоштовна до певного порогу	роялті 5 %	повністю безкоштовна
Якість графіки «з коробки»	висока	дуже висока	помірна
Поріг входження	низький	високий	низький
Розмір спільноти і документації	дуже велика	велика	помірна
Кількість готових ассетів	велика	велика	обмежена
Підтримка Steam SDK	повна	повна	обмежена
Підтримка мобільних платформ	сильна	помірна	сильна
Вимоги до апаратури під час розробки	помірні	високі	низькі

1.4. Підходи до симуляції динамічного середовища в Unity

Атмосфера survival-гри значною мірою тримається на живій навколишній сцені - на видимих змінах часу доби, погоди, поведінки рослин. У середовищі Unity ці задачі розв'язують або власним кодом, або готовими комерційними рішеннями. Базові принципи рендерингу неба, об'ємних хмар та атмосферного розсіювання, що покладено в основу таких пакетів, докладно описано у фундаментальному посібнику з комп'ютерної графіки [17]. У роботі розглянуто чотири пакети, що мають найбільшу частку використання серед інді-проєктів, за критеріями якості рендеру, продуктивності, інтеграції з Universal Render Pipeline і вартості.

Enviro 3 від Hendrik Haupt [25]. Комерційний пакет на Asset Store, спрямований на повний цикл день/ніч і динамічну погоду. Має готові пресети сезонів, прорахунок об'ємних хмар, систему опадів, інтеграцію з URP і HDRP. Перевагою є відносно невелике навантаження на GPU за рахунок використання skybox-шейдерів замість реального трасування світла.

AzureSky Dynamic Sky System. Інший популярний пакет, орієнтований лише на небо й освітлення. Має фізично-коректний розрахунок розсіювання Релея і Мі. Слабша підтримка погодних явищ - дощ і сніг реалізовані базово і потребують додаткового допрацювання.

UniStorm. Один з найстаріших пакетів категорії. Підтримує сезонні зміни, погоду, температурні впливи. Слабше тримає продуктивність у відкритому світі через велику кількість окремих компонентів.

The Vegetation Engine + NatureManufacture Assets [26, 27]. Не самі по собі симулятори погоди, а сполучна ланка між шейдером рослинності та глобальними параметрами вітру, дощу, температури. Дозволяють реалістично «гнути» дерева і кущі на вітрі, змінювати колір листя за сезонами без перебудови мешів. Працюють у зв'язці з Enviro 3.З огляду на потребу в комплексному рішенні зі сценарієм день/ніч, погодою і узгодженою рослинністю, для гри TwilightTails обрано зв'язку Enviro 3 [25] і The Vegetation Engine [26] з активами NatureManufacture [27]. Цей набір покриває майже всі підпотреби симуляції природного середовища без потреби писати власні шейдери, а вільні параметри (швидкість вітру, насиченість дощу, інтенсивність сонячного світла) виносяться у єдиний контролер DayCycleManager для подальшого керування з ігрового коду.

Зіставлення розглянутих рішень наведено в таблиці 1.3.

Таблиця 1.3

Порівняння рішень для симуляції природного середовища

Критерій	Enviro 3	AzureSky	UniStorm	Власна реалізація
Цикл день/ніч	повний	повний	повний	потребує розробки
Динамічна погода	дощ, сніг, туман, гроза	базова	повна, з градієнтами	потребує розробки
Сезонні зміни	так	ні	так	потребує розробки
Підтримка URP	повна	повна	часткова	-
Інтеграція з рослинністю	через TVE	відсутня	через власні шейдери	-
Документація	детальна	помірна	застаріла	-
Час освоєння	помірний	низький	високий	дуже високий

1.5. Підходи до реалізації штучного інтелекту в іграх

Поведінка нейгрових персонажів у комп'ютерних іграх історично будувалася на різних формальних моделях. Порівняльний аналіз чотирьох основних підходів для робототехніки та ігрового AI наведено у дослідженні Iovino та інших [1].

Кінцеві автомати (Finite State Machines, FSM). Найпростіша модель, у якій персонаж перебуває в одному з визначених станів (наприклад, «спокій», «переслідування», «атака», «втеча») і переходить між ними за фіксованими правилами. Перевагою є простота та передбачуваність. Недоліком є швидке зростання складності діаграми станів зі збільшенням числа поведінкових ситуацій [1].

Дерева поведінки (Behavior Trees, BT). Ієрархічна модель, у якій поведінка описується деревом вузлів. Вузли поділяються на композитні (послідовність, селектор, паралель), модифікатори та листи (умови і дії). Перевагою порівняно з FSM є модульність: окремі підгілки можна повторно використовувати в різних

контекстах [1]. Цей підхід широко застосовується в комерційних іграх, починаючи з серії Halo. Поєднання поведінкових дерев із системами сприйняття дозволяє побудувати правдоподібну поведінку віртуальних тварин у відкритому ігровому світі [2]. У Unity-середовищі найпоширенішим інструментом для роботи з деревами поведінки є комерційний пакет Behavior Designer від Opsive [13].

Утилітарний штучний інтелект (Utility AI). Кожній можливій дії персонажа ставиться у відповідність числова функція корисності, яка враховує поточний стан світу. Персонаж обирає дію з найвищим значенням. Перевагою є природна поведінка з плавними переходами. Недоліком є складність налагодження [18]. У дослідженні [21] зазначено, що цей підхід найкраще працює у комбінації з іншими моделями, наприклад, для вибору між гілками поведінкового дерева.

Алгоритми планування (Goal-Oriented Action Planning, GOAP). Персонаж формулює мету та автоматично будує послідовність дій для її досягнення. Найвідоміший приклад використання - гра F.E.A.R. У сучасних проєктах застосовуються рідше через складність реалізації [18].

Для проєкту TwilightTails обрано підхід на основі дерев поведінки, реалізований через пакет Behavior Designer [13]. Такий вибір обумовлений трьома міркуваннями: візуальний редактор спрощує налагодження окремих гілок поведінки; є можливість створення власних задач (Action, Condition) без зміни ядра плагіну; існує готова інтеграція з пакетом Malbers Animal Controller, який використовується для контролю руху тварин.

1.6. Постановка задачі

На основі наведеного аналізу та опрацювання технологічного стека сформульовано вимоги до власної реалізації.

1.6.1. Функціональні вимоги

Програмне забезпечення гри TwilightTails повинне забезпечувати:

1) вільне переміщення головного персонажа (єнота) у 3D-середовищі від третьої особи з підтримкою скелетної анімації, інверсної кінематики лап і фізичної взаємодії з рельєфом, перешкодами і предметами;

2) систему потреб персонажа з параметрами здоров'я, голоду, спраги, температури і витривалості, з прогресивним зниженням значень у часі та візуальною індикацією критичних станів;

3) систему pickup-взаємодії з оточенням: персонаж тримає в передніх лапах один предмет за раз (їжа, дрібна тварина, будівельний модуль), має змогу його використати, з'їсти або скинути; одночасне утримання декількох предметів свідомо обмежено для збереження тваринної натуральності керування;

4) режим будівництва з попереднім переглядом конструкції напівпрозорим контуром, перевіркою колізій, списанням ресурсів та базовим набором споруд;

5) AI-екосистему з шести базових видів тварин (єнот-NPC, заєць, лисиця, вовк, ведмідь, жаба) на основі поведінкових дерев Behavior Designer з диференційованою поведінкою хижаків, здобичі та нейтральних видів;

6) симуляцію динамічного середовища через плагін Enviro 3 з циклом день/ніч, погодними умовами (дощ, туман, гроза), сезонними змінами рослинності, з впливом на ігрові механіки;

7) систему збереження і завантаження ігрового сеансу на основі серіалізації стану світу у JSON-файл у каталозі persistentDataPath, з можливістю продовжити гру після виходу;

8) механіку полювання й харчування з різними сценаріями для кожного виду здобичі та правилами насичення/отруєння залежно від типу їжі;

9) звуковий супровід світу - фоновий ембієнт лісу з нашаруванням голосів птахів і гризунів, погодні звуки, музичні теми денного й нічного часу через MusicManager;

10) інтегрований туторіал, що поступово знайомить гравця з ключовими механіками (керування, pickup, харчування, режим будівництва) через спливаючі підказки і відмічені цілі на HUD;

11) оптимізаційні механізми (динамічне завантаження ассетів через Addressables, відключення поведінкового дерева у віддалених тварин, LOD-системи рослинності) для забезпечення стабільних понад 60 кадрів за секунду на цільовому обладнанні.

1.6.2. Нефункціональні вимоги

1) Продуктивність: середня частота кадрів не нижче 60 кадрів за секунду на типовому ігровому комп'ютері (відеокарта рівня NVIDIA GeForce RTX 3060, 16 ГБ оперативної пам'яті) при стандартних налаштуваннях якості.

2) Системні вимоги: ОС Windows 10 або новіша 64-розрядна; процесор не нижче Intel Core i5 шостого покоління; обсяг оперативної пам'яті від 8 ГБ; обсяг відеопам'яті від 4 ГБ; вільне місце на диску близько 15 ГБ.

3) Масштабованість світу: можливість збільшення розміру локації і кількості одночасно активних NPC-тварин без принципової зміни архітектури підсистеми ШІ.

4) Розширюваність: модульна структура коду, що дозволяє додавати нові види тварин, ресурсів і будівель без зміни існуючих модулів. Окремо передбачено архітектурну заготовку Mirror Networking для майбутнього розширення програмного забезпечення на кооперативний режим.

5) Локалізованість: рядки інтерфейсу винесено в окремі таблиці, додавання нової мови не вимагає перекомпіляції коду.

6) Зрозумілість: наявність туторіалу, що навчає базовим механікам нових гравців без читання зовнішньої документації.

7) Межі реалізації: одна ігрова локація - європейський ліс площею близько одного квадратного кілометра; шість видів NPC-тварин (єнот, заєць, лисиця, вовк, ведмідь, жаба); відкритий геймплей без сюжетної кампанії - ігровий процес безперервний, кінцева мета визначається гравцем.

8) Локалізація користувацького інтерфейсу: багатомовний інтерфейс з підтримкою української та англійської мов на базі пакета Unity Localization.

У першому розділі розглянуто стан сучасної ігрової індустрії в сегменті жанру виживання. Встановлено, що жанр демонструє стійке зростання, а помітна частина успішних релізів останніх років припадає на інді-команди з фокусом на атмосферний одиночний геймплей [9, 15].

Виконано порівняльний аналіз п'яти провідних програмних аналогів (The Forest [3], Rust [4], Subnautica [5], ARK: Survival Evolved [6], Green Hell [7]). Встановлено, що жодна з розглянутих ігор не покриває одночасно усього набору рис, бажаних для survival-гри з персонажем-твариною та повною українською локалізацією, - саме ця ніша обрана для гри TwilightTails.

Обґрунтовано вибір ігрового рушія Unity 6 [11] як платформи з найкращим балансом між гнучкістю, доступністю інструментарію та вартістю ліцензії. Для симуляції природного середовища обрано зв'язку Enviro 3 [25] і The Vegetation Engine [26]. Для підсистеми штучного інтелекту обрано підхід на основі дерев поведінки з використанням комерційного пакета Behavior Designer [13] - такий вибір відповідає висновкам наукових праць [1, 2] про переваги ієрархічних моделей прийняття рішень над класичними скінченними автоматами.

Сформульовано функціональні та нефункціональні вимоги до програмного забезпечення гри TwilightTails, визначено обмеження реалізації. Зокрема функціональні вимоги охоплюють керування персонажем-єнотом, систему підбирання предметів, режим будівництва, штучний інтелект шести видів тварин, цикл день і ніч, збереження стану та локалізацію інтерфейсу шістьма мовами; нефункціональні - продуктивність на рівні 60 кадрів за секунду на середній якості графіки, цільові апаратні конфігурації та інтеграцію з платформою Steam.

Отримані у першому розділі результати створюють фундамент для подальших етапів роботи: визначений технологічний стек і сформульовані вимоги слугують вхідними даними для проєктування архітектури застосунку, опису його ключових підсистем та обґрунтування вибору патернів реалізації - ці питання розкрито у другому розділі кваліфікаційної роботи.

2. ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Загальна архітектура застосунку

На основі вимог, сформульованих у попередньому розділі, побудовано загальну архітектуру майбутнього застосунку. Як методологія обрано об'єктно-орієнтовану розробку з елементами компонентної архітектури, що є природним підходом у середовищі Unity [11]. Рушій ґрунтується на патерні Entity-Component, у якому ігровий об'єкт (GameObject) виступає сутністю, до якої прикріпляються компоненти (MonoBehaviour-скрипти), що описують його поведінку. Базовий каталог патернів проєктування ігрових систем (Component, Game Loop, Observer, State, Object Pool) узагальнено у [20]; обрана архітектура спирається на ці патерни в адаптованій до Unity формі.

Архітектурно додаток поділено на чотири шари (див. рис. 2.1):

- презентаційний шар - візуалізація ігрового світу, користувацький інтерфейс, звукове супроводження;

- шар ігрової логіки - контролери ігрових об'єктів, штучний інтелект, ігрові правила;

- шар сервісів - збереження стану, локалізація, обробка вводу, симуляція довкілля;

- шар платформи - ігровий рушій Unity та інтегровані сторонні плагіни.

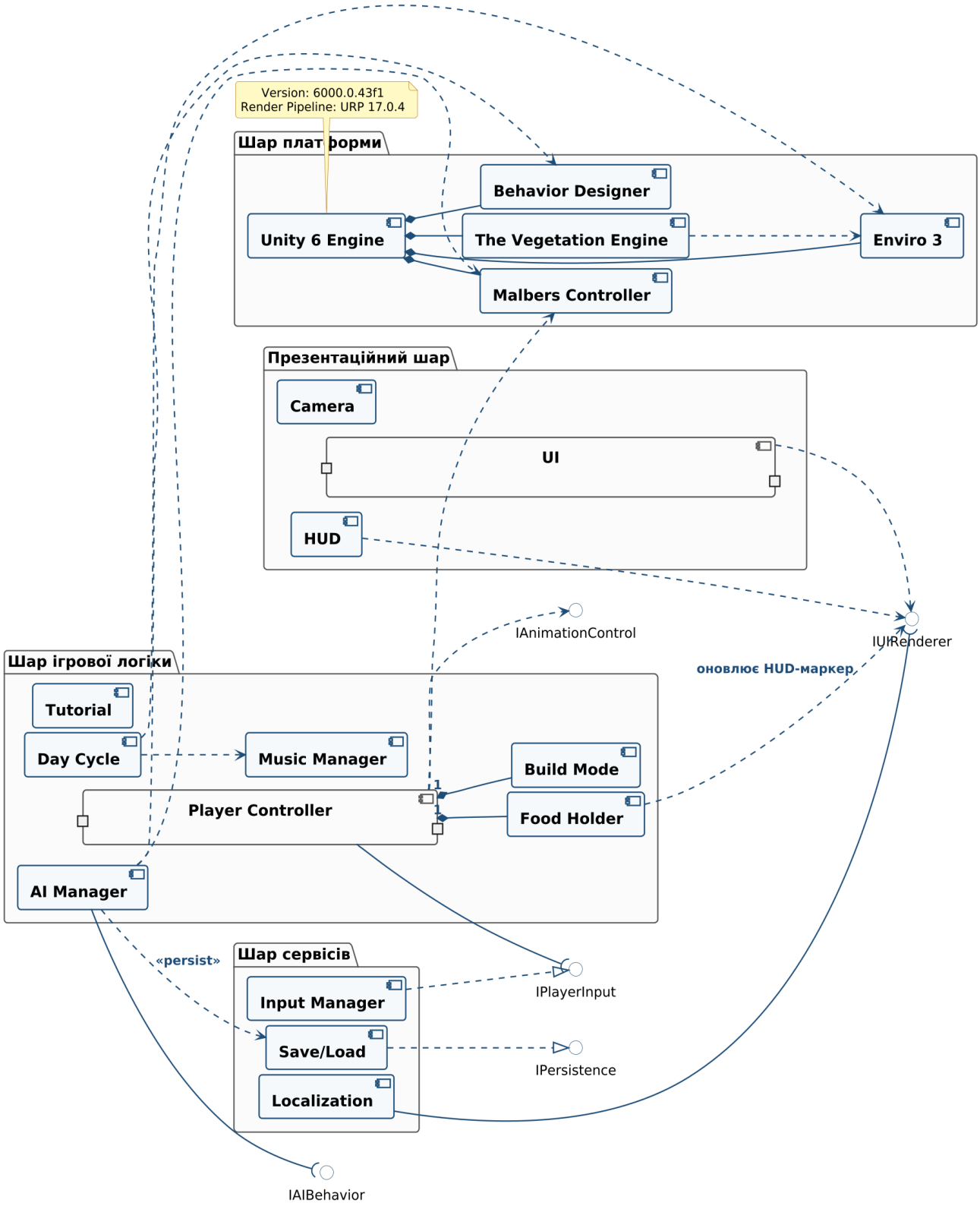


Рис. 2.1 Загальна архітектура застосунку

Такий поділ дозволяє замінювати реалізацію в кожному шарі незалежно від інших. Збереження стану у форматі JSON можна замінити на бінарний формат без впливу на ігрову логіку. Behavior Designer теоретично можна замінити на власну реалізацію FSM без зміни контрактів ігрових об'єктів. Архітектурно у репозиторії також залишено заготовку мережного шару на базі Mirror Networking [12] - вона не задіяна у поточній збірці, але може бути активована на майбутньому етапі розширення на кооперативний режим.

Унаслідок особливостей Unity-середовища у ньому немає чіткої межі між класами-сутностями та контролерами. Кожен скрипт є нащадком класу MonoBehaviour і одночасно описує і дані, і поведінку [11]. Тому традиційний шаблон MVC застосовується в адаптованій формі: моделі представлені класами ScriptableObject (наприклад, ResourceData), контролери - MonoBehaviour-скриптами, представлення - компонентами Canvas і UI.

2.2. Опис контролера персонажа

Персонаж-єнот є центральним елементом ігрового досвіду, тому його контролер - одна з найбільш насичених підсистем програмного забезпечення. Структура контролера будувалася навколо трьох вимог: природна анімація руху на нерівному ландшафті, реактивна взаємодія з оточенням і чіткий розподіл відповідальності між модулями.

2.2.1. Шарова організація персонажа

Контролер єнота розкладено на чотири логічні шари. Кожен шар реалізовано окремим компонентом, що дозволяє замінювати чи розширювати його незалежно від інших.

- шар фізичного руху - компонент MAnimal з пакета Malbers Animal Controller [24]. Відповідає за переміщення, обертання, інерцію, реакцію на нахили рельєфу. Підв'язаний до Rigidbody і CapsuleCollider;

- шар анімації - стандартний Animator Controller Unity з готовою деревом станів від Malbers. Перемикається тригерами (Jump, Eat, Attack) і параметрами (Speed, Vertical, Horizontal);

- шар інверсної кінематики - компонент IKController на основі BioIK. Налаштовує положення лап персонажа під час руху по нерівностях, повертає голову у напрямку об'єкта взаємодії, активує очі за допомогою RaccoonEyesController;

- шар ігрової логіки - власний скрипт Raccoon, що зберігає посилання на нижні шари і реалізує специфічні для гри події (підбирання предмета, споживання їжі, отримання урону, респавн).

2.2.2. Обробка вводу

Ввід обробляється через InputManager - обгортку над Unity Input System. Її задача - ізолювати ігровий код від конкретного джерела вводу (клавіатура, геймпад, сенсорний екран у майбутньому). InputManager публікує події руху, повороту камери, дій (pickup, атаки, відкриття меню налаштувань), на які підписуються Raccoon і UIManager.

Окремо опрацьовано перемикання режимів керування: коли активний режим будівництва, обмежується керування камерою; коли відкрите меню налаштувань - блокується ввід руху. Цю логіку інкапсульовано в стек активних контекстів InputContext, що дозволяє коректно повертатися до попереднього стану після закриття меню.

2.2.3. Взаємодія з оточенням

Виявлення інтерактивних об'єктів реалізовано через комбінацію двох механізмів:

1) сфера-тригер фіксованого радіуса навколо персонажа постійно збирає список об'єктів з компонентом IInteractable;

2) при натисканні клавіші взаємодії (E) виконується додатковий raycast з камери в напрямку центру екрана - пріоритет отримує об'єкт, на який гравець дивиться.

Така комбінація дає інтуїтивну поведінку: предмети під ногами підбираються автоматично, а вибіркові дії (наприклад, активація куща з ягодами серед декількох поруч) керуються прицілом.

Послідовність обробки одного інтерактивного жесту наведена на рисунку 2.2:

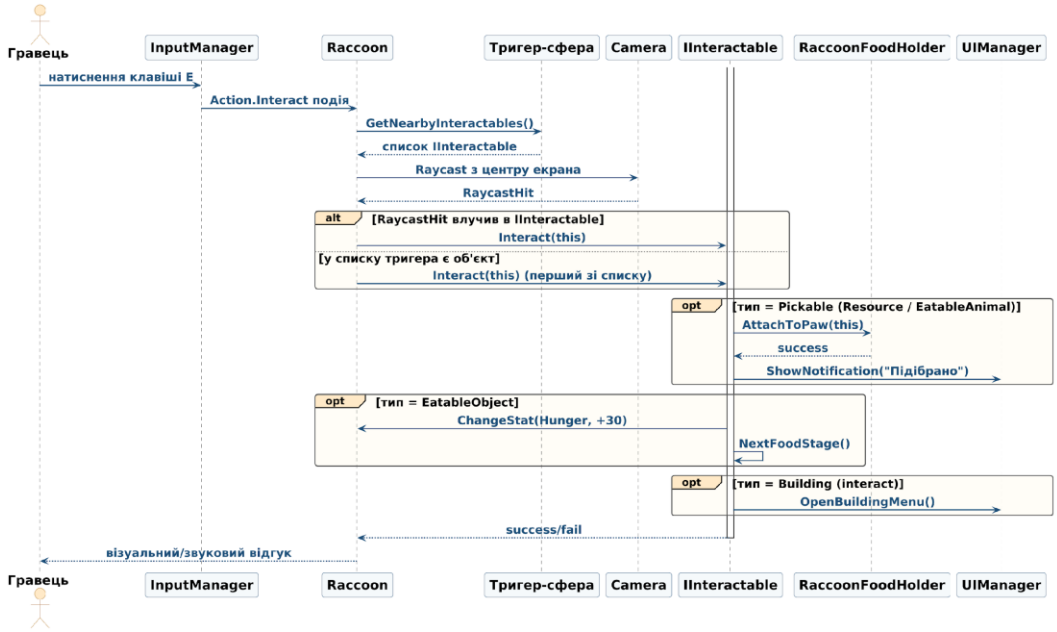


Рис. 2.2 Послідовність обробки взаємодії гравця з оточенням

Перевагою такого підходу є те, що додавання нового типу взаємодії не вимагає змін у Raccoon: достатньо створити новий клас, що реалізує IInteractable. Це відповідає принципу відкритості/закритості SOLID [22].

2.3. Модель ігрового світу

Ігровий світ представлений однією локацією - ділянкою європейського лісу площею близько одного квадратного кілометра. У ньому виділено такі сутності:

- гравець (Raccoon) - керована тварина-єнот;

- дикі тварини (Wildlife) - некеровані тварини, що населяють ліс. Поділяються на два класи: хижаки (вовк, ведмідь, лисиця) та здобич (заєць, жаба, єнот-NPC). Підхід до моделювання їхньої поведінки відповідає рекомендаціям, наведеним у [2];

- ресурси (Resources) - об'єкти, що збираються гравцем. Існує три типи: їжа (гриби, м'ясо), будівельні матеріали (деревина, каміння), вода. Можуть утворювати скупчення (стеки);

- будівлі (Structures) - об'єкти, які гравець може створити з зібраних ресурсів. Розміщуються на терені та зберігаються між сесіями;

- природне середовище (Environment) - ландшафт, рослинність, погодні умови, час доби. Створюється на етапі підготовки сцени, не змінюється під час гри (за винятком освітлення).

UML-діаграма класів моделі світу подана на рисунку 2.3:

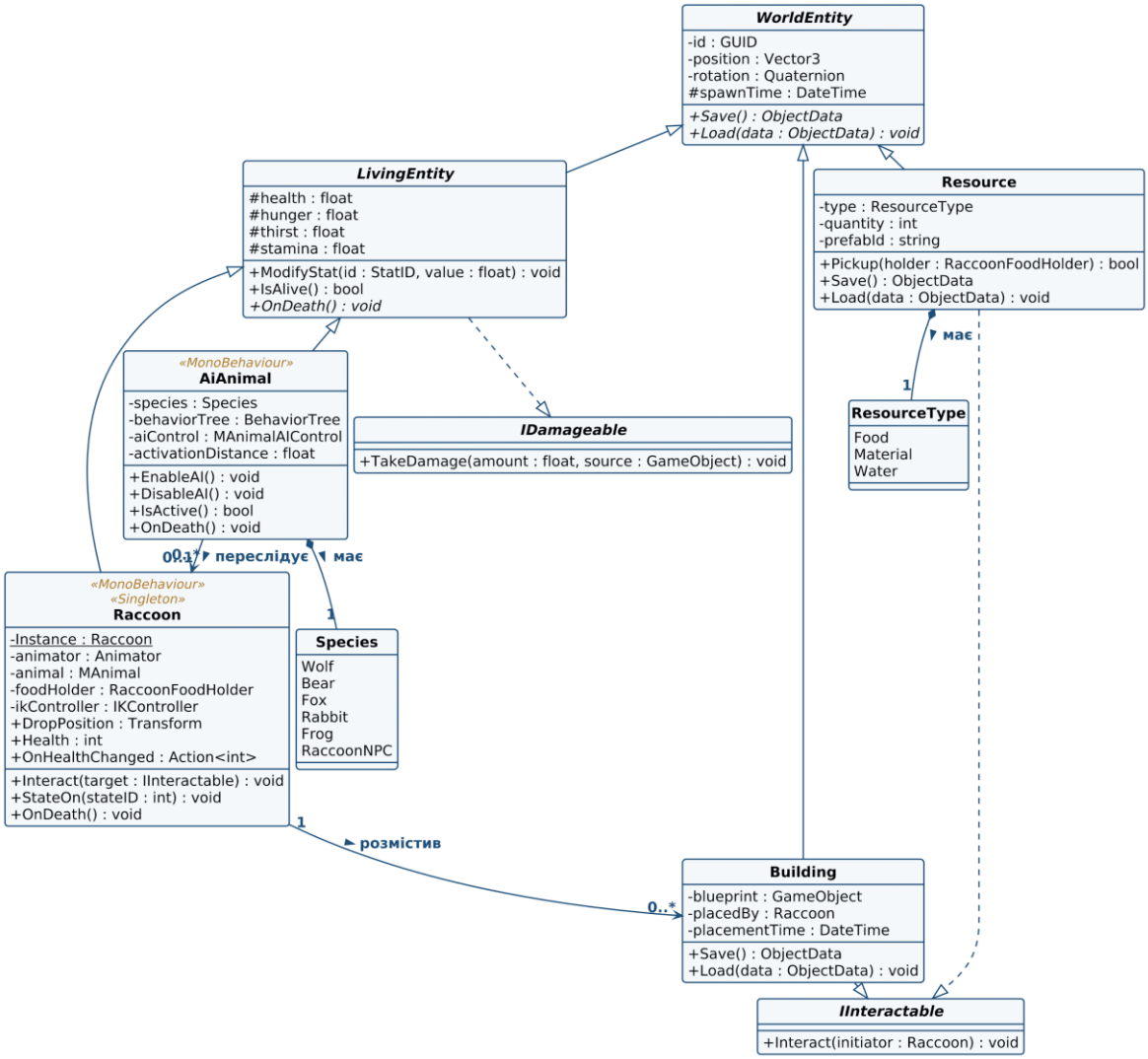


Рис. 2.3 UML-діаграма класів моделі ігрового світу

Базовий клас WorldEntity представляє будь-який об'єкт, що існує у світі. Від нього успадковуються три головні гілки:

- LivingEntity - має параметри життєдіяльності і ділиться на гравця та керовану штучним інтелектом тварину;

- Resource - описує предмет для збирання;

- Building - описує побудовану гравцем споруду.

2.4. Структура підсистеми штучного інтелекту

Поведінка диких тварин організована як дерево поведінки (Behavior Tree) на основі пакета Behavior Designer [13]. Загальна структура дерева є єдиною для більшості видів і визначається набором задач (нодів), а специфіка кожного виду задається через параметри.

Діаграма діяльності, що описує процес ухвалення рішень тварини, наведена на рисунку 2.4:

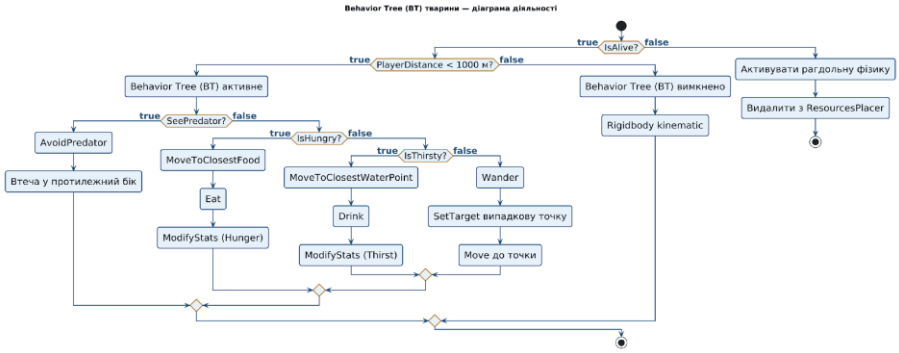


Рис. 2.4 Діаграма діяльності поведінки тварини

Поведінка обирається за пріоритетом: спершу перевіряється, чи бачить тварина хижака (для здобичі), потім - чи голодна, потім - чи спрагла, і лише в останню чергу тварина переходить до випадкового блукання. У такій моделі легко додавати нові гілки поведінки, наприклад, сон, спарювання, втечу при ушкодженні. Підхід відповідає рекомендаціям, наведеним у працях [1, 2].

Для оптимізації продуктивності реалізовано механізм активації штучного інтелекту за відстанню: якщо тварина знаходиться далі за тисячу метрів від найближчого гравця, її дерево поведінки тимчасово зупиняється, а об'єкт переводиться у статичний стан. Це дозволяє утримувати на сцені декілька десятків тварин без значного впливу на частоту кадрів.

Кастомні задачі для дерева поведінки розроблено для тих сценаріїв, які не покриваються базовим набором Behavior Designer. Перелік розроблених задач наведено у таблиці 2.1.

Таблиця 2.1

Перелік власних задач для дерева поведінки

Назва задачі	Тип	Призначення
`MoveToClosestFood`	Action	пошук найближчого об'єкта з тегом «Food» і рух до нього
`MoveToClosestWaterPoint`	Action	визначення найближчої точки межі вода-суша і рух до неї
`AvoidPredator`	Action	втеча у протилежний від хижака бік
`Attack`	Action	активація стану атаки в анімаційному контролері
`AttackAction`	Action	атака конкретної цілі з використанням IK для огляду
`Wander`	Action	випадкове блукання у заданому радіусі
`SetTarget`	Action	встановлення нової цілі для AI
`SetMode`, `SetStance`, `SetAnimalSpeed`	Action	управління станами анімаційного контролера
`ModifyStats`	Action	зміна показників (голод, спрага) на задану величину
`ChangeState`, `StopFollowingTarget`	Action	сервісні дії з керування поведінкою
`ChangeDetectionRadius`	Action	динамічна зміна радіуса виявлення хижака
`IsHungry`, `IsThirsty`, `IsTired`	Condition	перевірка показників життєдіяльності
`SeePredator`	Condition	перевірка наявності хижака в радіусі видимості
`IsInSightOrNear`	Condition	комбінована перевірка близькості та зорового контакту
`IsTargetDead`	Condition	перевірка, чи мертва ціль

2.5. Структура підсистеми збереження стану

Підсистема збереження стану реалізує дві основні функції: автоматичне збереження поточного стану світу кожні п'ять секунд та завантаження збереженого стану при запуску гри.

Збережувані дані поділено на дві категорії:

1) Дані гравця (PlayerData) - позиція, обертання, показники здоров'я, голоду, спраги, втоми, ідентифікатор предмета, який зараз перебуває в лапах персонажа. Записуються в єдиний слот гравця, оскільки програмне забезпечення розраховане на одиночний сеанс.

2) Дані об'єктів світу (ObjectData) - позиція, обертання, ім'я префабу, прапорець використання, специфічні дані (наприклад, кількість залишку для стеку ресурсів). Стосуються тих об'єктів, що мають компонент-маркер SavableObject.

Формат збереження - JSON, реалізований через стандартний клас Unity JsonUtility [11]. Файл збереження розміщується в стандартному для платформи місці, отриманому через властивість Application.persistentDataPath.

UML-діаграма класів підсистеми збереження подана на рисунку 2.5:

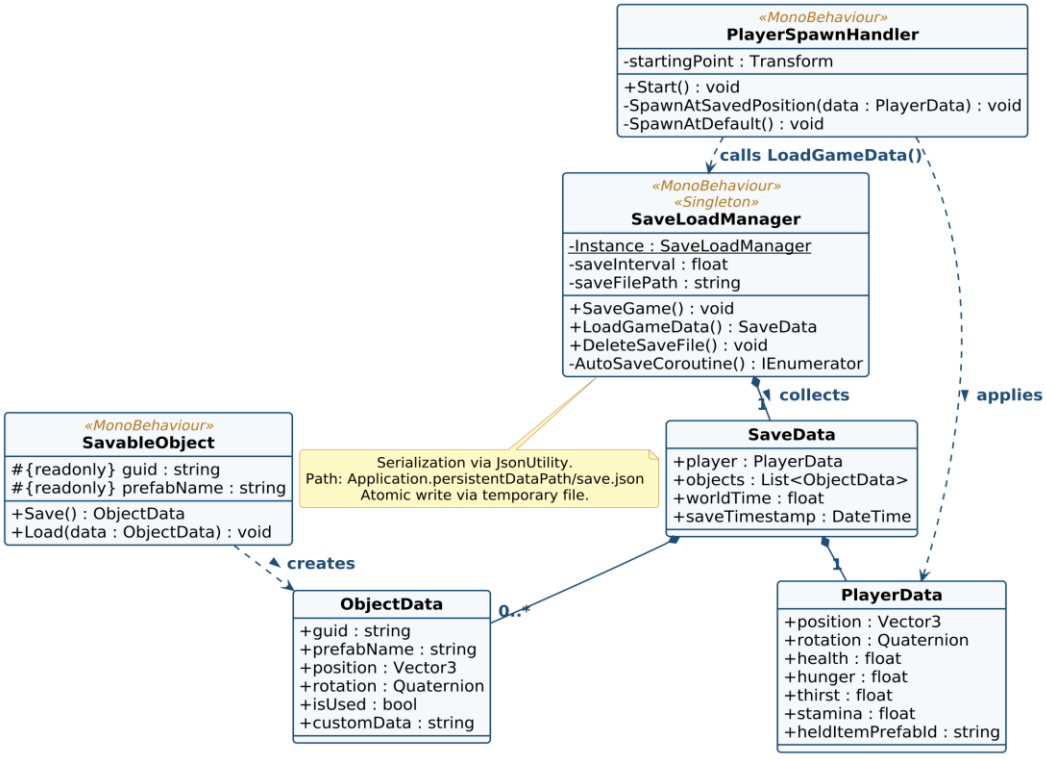


Рис. 2.5 UML-діаграма підсистеми збереження стану

Особливістю реалізації є використання глобально унікальних ідентифікаторів GUID для розпізнавання об'єктів між сесіями. При першому розміщенні об'єкта у світі (або при додаванні компонента SavableObject у редакторі) генерується GUID за допомогою стандартного методу System.Guid.NewGuid. Цей ідентифікатор зберігається разом з даними об'єкта і дозволяє при завантаженні знайти потрібний екземпляр серед сотень інших.

2.6. Проєктування користувацького інтерфейсу

Користувацький інтерфейс гри поділяється на три основні групи екранів: головне меню, ігровий HUD з панеллю швидкого доступу, меню налаштувань.

2.6.1. Головне меню

Складається з логотипу гри, кнопок «Single Game», «Settings», «Exit Game», а також блоку з даними розробників (зліва внизу). Фон становить заздалегідь підготовлена сцена з лісом, єнотом по центру і панорамою заходу сонця. Передбачено невелику паралакс-анімацію для оживлення статичного фону. Завантаження ігрової сцени супроводжується окремим Loading-екраном з логотипом і слідами лап. Зовнішній вигляд головного меню наведено на рисунку 3.1.

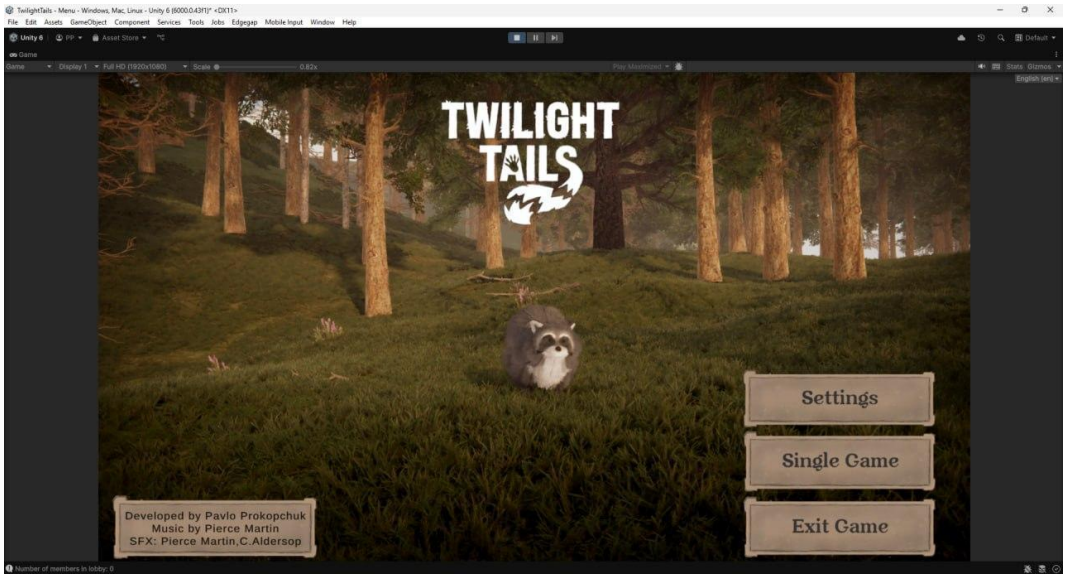


Рис. 3.1 Головне меню гри TwilightTails

2.6.2. Ігровий HUD

Складається з елементів, що відображаються поверх ігрового світу:

1) індикатори показників персонажа (здоров'я, голод, спрага) у вигляді трьох круглих іконок у нижній лівій частині екрана; четвертий показник - втома - використовується внутрішньо для регулювання швидкості бігу і у HUD не виводиться;

2) панель туторіальних підказок у верхньому правому куті: спливаючі повідомлення з ціллю і списком клавіш для виконання дії (наприклад, WASD - Move, Space - Jump, Shift - Sprint, Ctrl - Crouch для етапу базової локомоції; E - Pick Up для етапу інтеракції);

3) маркер взаємодії з оточенням, що з'являється над об'єктом IInteractable, на який наведений приціл (іконка клавіші E + назва дії: «Підняти», «Eat», «Атака»);

4) сповіщення про події (підбирання предмета, отримання урону, виконання етапу туторіала) у вигляді тимчасових повідомлень з анімацією.

Свідомо обрано мінімалістичний підхід без повноекранного інвентаря: персонаж-тварина не має «карманів», тому модель «один предмет у лапах» виглядає природніше і відповідає виборові протагоніста. Зовнішній вигляд HUD з активними туторіальними підказками руху наведено на рисунку 3.2.

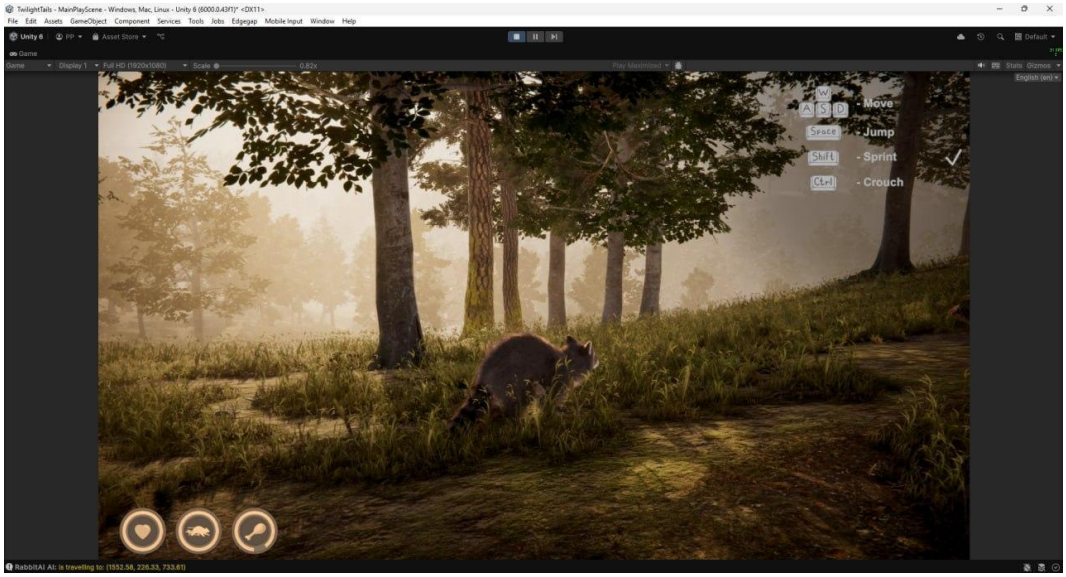


Рис. 3.2 Ігровий HUD з активним туторіалом керування

2.6.3. Меню налаштувань

Викликається з головного меню та з ігрового процесу клавішею Esc. Реалізоване у вигляді єдиної прокручуваної панелі «Game Options» з чотирма послідовними розділами (зверху вниз): «Control Settings» (керування), «Graphic options» (графіка), «Localization» (мова), «Audio Options» (звук). Доступні параметри:

- чутливість миші по осі X та Y (Control Settings); - режим вікна, роздільна здатність, увімкнення VSync, цільова частота кадрів, рівень згладжування, рівень якості графіки High Fidelity / Balanced / Performant (Graphic options); - вибір мови інтерфейсу з випадного списку: English / Українська / Deutsch / Español (Localization); - загальна гучність, гучність музики і ефектів окремо (Audio Options).

Розділи керування і графіки наведено на рисунку 3.3, розділ вибору мови з розкритим випадним списком - на рисунку 3.4.

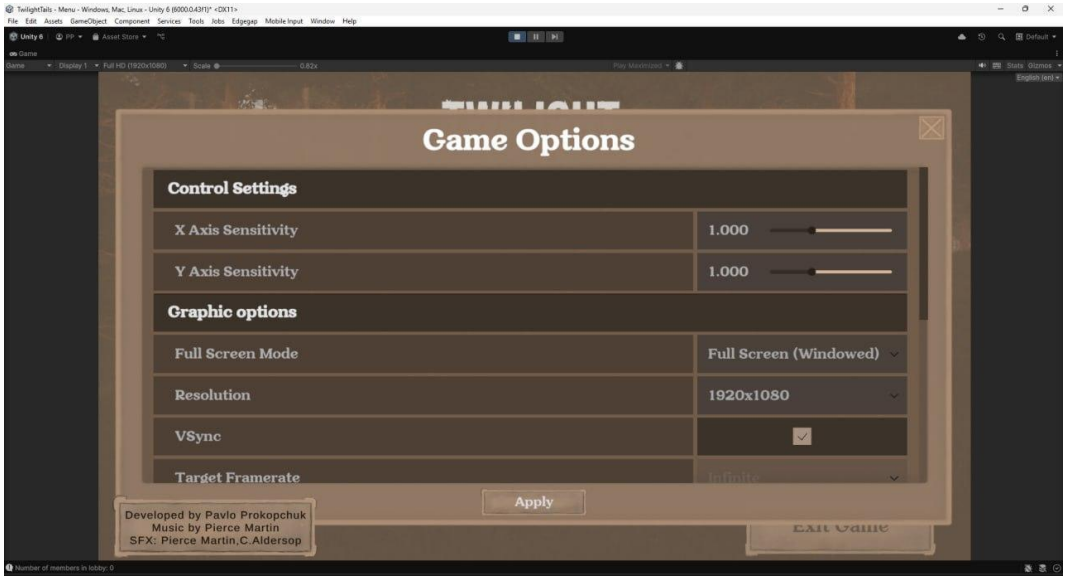


Рис. 3.3 Меню налаштувань: керування і графіка

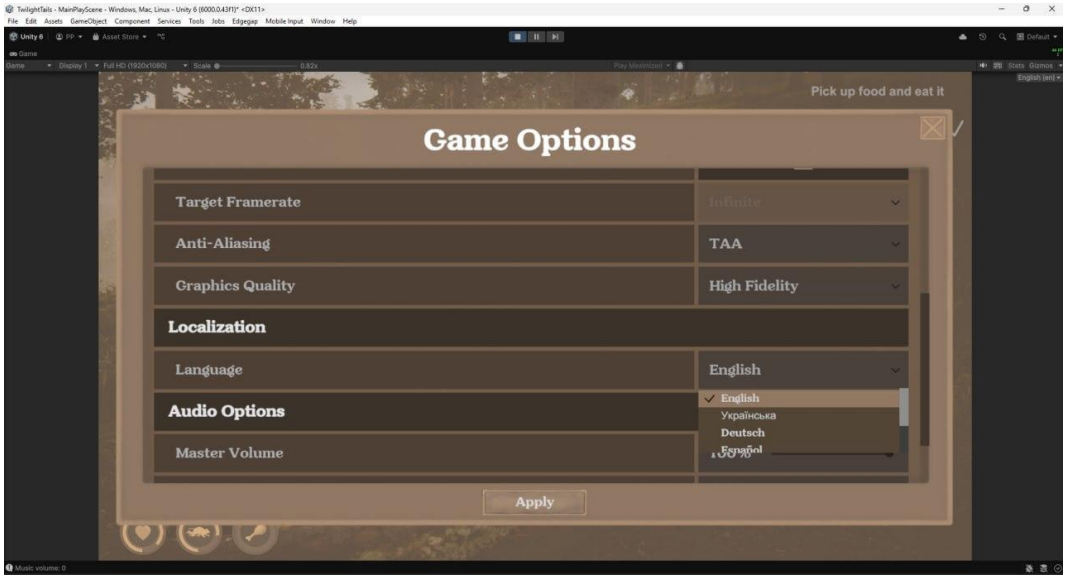


Рис. 3.4 Меню налаштувань: вибір мови інтерфейсу

У другому розділі викладено результати проєктування програмного забезпечення. Сформовано чотиришарову архітектуру застосунку, яка відокремлює представлення, ігрову логіку, сервіси і платформу. Це дає можливість розвивати кожен шар окремо без перебудови сусідніх - наприклад, замінити Behavior Designer на власний FSM-движок або переписати збереження з JSON у бінарний формат.

Описано архітектуру контролера персонажа з чотирма логічними шарами (фізичний рух Malbers, анімація, інверсна кінематика BioIK, ігрова логіка), розкладено обробку вводу через стек контекстів InputContext і схему виявлення інтерактивних об'єктів через комбінацію тригер-сфери і прицільного raycast. Така декомпозиція знижує зчеплення коду і відповідає принципам SOLID [22].

Розроблено модель ігрового світу з трьома основними сутностями (жива істота, ресурс, будівля) та власні задачі для дерева поведінки тварин [13]. Сформовано структуру підсистеми збереження стану на основі JSON-формату і глобально унікальних ідентифікаторів.

Окремо опрацьовано архітектуру користувацького інтерфейсу: визначено склад головного меню, меню налаштувань з підрозділами керування, графіки, локалізації та звуку, а також структуру внутрішньоігрового HUD з індикаторами здоров'я, голоду, спраги та умовами контекстної взаємодії. Така структура UI

відповідає сучасним конвенціям survival-ігор і забезпечує мінімальний поріг входу для нових гравців.

Описані у другому розділі архітектурні рішення створюють основу для безпосередньої програмної реалізації: визначено межі відповідальності кожного шару, відокремлено логіку від платформозалежного коду, що дозволяє виконувати реалізацію підсистем у довільному порядку та незалежно тестувати їх. Прийняті проєктні рішення відповідають принципам SOLID та архітектурним рекомендаціям Clean Architecture [22]: верхні шари не залежать від нижніх, що дає змогу замінити конкретні реалізації (наприклад, JsonUtility для серіалізації чи Unity Localization для перекладу) без впливу на ігрову логіку.

Підготовлені UML-діаграми ілюструють як статичну структуру (класи, компоненти), так і динамічну поведінку (послідовність взаємодії, діаграма діяльності AI-тварин, стани підсистеми збереження), що формує повну специфікацію для подальшої реалізації. Запроєктована модульна архітектура містить чітко визначені точки розширення: підсистему штучного інтелекту можна розширити додаванням нових кастомних задач для Behavior Designer; підсистему збереження - додаванням нових компонентів-нащадків SavableObject; підсистему локалізації - додаванням нових мовних таблиць у Unity Localization без зміни коду. Передбачено також архітектурні точки розширення для кооперативного режиму на основі Mirror Networking з підтримкою спільних сесій до чотирьох гравців, що формує заділ для майбутнього розвитку проєкту.

Особливості програмної реалізації обраних архітектурних рішень засобами Unity 6 та супутніх пакетів детально розглянуто у третьому розділі кваліфікаційної роботи.

3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Обґрунтування вибору інструментальних засобів

Для реалізації програмного забезпечення обрано стек технологій, що включає ігровий рушій, готові комерційні пакети для тварин, рослинності, штучного інтелекту та середовища, а також інструменти збереження стану й локалізації (див. табл. 3.1). Кожен компонент стека характеризується стабільними релізами, активною користувацькою спільнотою і розгорнутою документацією [11, 12, 13]. Подібний вибір мінімізує ризик потрапляння у технологічний глухий кут під час подальшої розробки та зменшує час, який можна було б витратити на повторну реалізацію базової функціональності, доступної у вигляді готових модулів. Особливу увагу при формуванні стека приділено інтеграційній сумісності компонентів: усі обрані пакети офіційно підтримуються розробниками для Unity 6 у HDRP та URP конвеєрах рендерингу, мають перехресну сумісність між собою (Behavior Designer працює з Malbers Animal Controller через адаптери; Enviro 3 синхронізує з The Vegetation Engine через спільну глобальну змінну часу доби; Unity Localization інтегрується з усіма UI-елементами через стандартні TextMeshPro поля). Загальна вартість комерційних пакетів стека у роздрібних цінах магазину Unity Asset Store становить близько 350 доларів США, що для одиночного інді-проєкту є прийнятною сумою, особливо з огляду на сезонні знижки до 70 % і навчальні ліцензії для студентів.

Обраний стек технологій забезпечує реалізацію функціональних та нефункціональних вимог, сформульованих у першому розділі, та надає інструменти для розв'язання задач, пов'язаних із підтримкою анімації четвероногих персонажів, побудовою штучного інтелекту тварин, симуляцією природного середовища, локалізацією інтерфейсу та збереженням стану світу.

Таблиця 3.1

Стек технологій програмного забезпечення

Категорія	Інструмент	Версія	Обґрунтування вибору
Ігровий рушій	Unity [13]	6000.0.43f1	широка спільнота, безкоштовність до певного порогу, наявність потрібних готових пакетів
Мова програмування	C#	.NET Standard 2.1	стандарт у Unity-екосистемі
Контролер тварин	Malbers Animal Controller [26]	актуальна stable	повноцінна локомоція для четвероногих, інтеграція з AI
Дерева поведінки	Behavior Designer [15]	актуальна stable	візуальний редактор, можливість розширення задачами
Інверсна кінематика	BioIK	актуальна stable	стабільна, працює без додаткового налаштування
Рослинність	The Vegetation Engine [28]	актуальна stable	оптимізована рослинність з GPU-instancing
Природні моделі	NatureManufacture CEVP Vol.1 [29]	актуальна stable	реалістичні моделі центральноєвропейського лісу
Генерація рельєфу	Gaia Pro 2023 (Procedural Worlds)	актуальна stable	швидке створення великих ландшафтів
Серіалізація даних	Unity JsonUtility	вбудована	простота, нульові залежності
Локалізація	Unity Localization	1.5.4	офіційний пакет, інтеграція з Addressable Assets
Камера	Cinemachine	2.10.3	стандарт індустрії для камер у Unity

3.2. Реалізація персонажа-єнота

Центральним класом, що описує гравця, є Raccoon. Він зберігає посилання на компоненти-залежності (контролер локомоції, анімаційну систему, інверсну кінематику, тримач їжі) і організує взаємодію між ними. Клас побудований за патерном Singleton: посилання на гравця доступне через статичну властивість Raccoon.Instance, що спрощує доступ з інших підсистем (UI, туторіал, режим будівництва).

3.2.1. Склад префаба гравця

На префабі персонажа розміщено такі компоненти:

- `MAnimal` (Malbers) - контролер локомоції тварини;
- `MPickUp` (Malbers) - система взаємодії з предметами;
- `Animator` - стандартний контролер анімації Unity, прив'язаний до RaccoonAnimatorController;
- `Rigidbody`, `CapsuleCollider` - фізичні компоненти;
- `RaccoonFoodHolder` - власна логіка тримання їжі у передніх лапах;
- `RagdollController` - активація рагдольної фізики при смерті;
- `RaccoonEyesController` - керування морганням і слідуванням зіниць за об'єктом інтересу;
- `IKController` - управління інверсною кінематикою голови та лап через BioIK.

Спрощений лістинг класу Raccoon наведено в Додатку Б.

3.2.2. Локомоція

Локомоція повністю покладена на пакет Malbers Animal Controller [24]. Цей пакет надає стани руху (Idle, Locomotion, Jump, Swim, Fall, Death, Sit), переходи між ними і анімаційні параметри. Власний код Raccoon лише надсилає у Malbers вектор руху і спостерігає за подіями зміни стану. Для випадків, коли потрібен спеціальний стан (наприклад, «їдіння»), використано Mode System з пакета - у редакторі вмикається відповідний пресет.

3.2.3. Інтеграція з інверсною кінематикою

Окремий модуль IKController керує положенням голови і передніх лап персонажа. Голова обертається у напрямку об'єкта, на який наведено приціл (для природного «огляду»), а лапи коригуються по висоті рельєфу через RaycastHit. Усі обчислення виконуються в LateUpdate, після того як Animator визначив базові пози, що уникає ефекту «перерваної» анімації.

Очі реалізовано окремим компонентом RaccoonEyesController. Він має моргання за випадковим інтервалом (середня частота - одне моргання на чотири секунди) і слідування зіниць за об'єктом, який встановив IKController. Дрібний штрих, але саме він робить персонажа «живим».

3.2.4. Полювання та їдіння

Полювання реалізовано через комбінацію Malbers Mode System (стан атаки) і власного класу EatableAnimalController. При успішному ударі по дрібній тварині (заєць, жаба) її MAnimal-контролер вимикається, додається компонент Pickable, і тварина перетворюється на предмет, який можна підняти.

Поглинання їжі обробляється класом RaccoonFoodHolder. Той самий клас відповідає за утримання будь-якого підібраного предмета у передніх лапах (див. рис. 3.7). При натисканні правої кнопки миші, якщо у лапах знаходиться їстівний предмет, активується стан Eat у Malbers, який відтворює відповідну анімацію. Після кожного циклу анімації викликається метод BiteFood, що зменшує показник голоду і змінює мешу та текстуру об'єкта на наступну стадію (за допомогою класу FoodModelStates). Після останньої стадії об'єкт знищується.

3.2.5. Респавн і смерть

Смерть персонажа обробляє RagdollController - вимикає Animator і MAnimal, активує всі Rigidbody на скелеті, що дає природне падіння тіла. Через три секунди викликається PlayerRespawner, який вибирає найближчу до точки смерті безпечну позицію (відсутність хижаків у радіусі п'ятнадцяти метрів) і повертає гравця у звичайний стан з повним здоров'ям, але з частковим скиданням показників голоду і втоми. Окремий екран респавну з кнопками «Грати знову» і «Головне меню» наведено на рисунку 3.5.

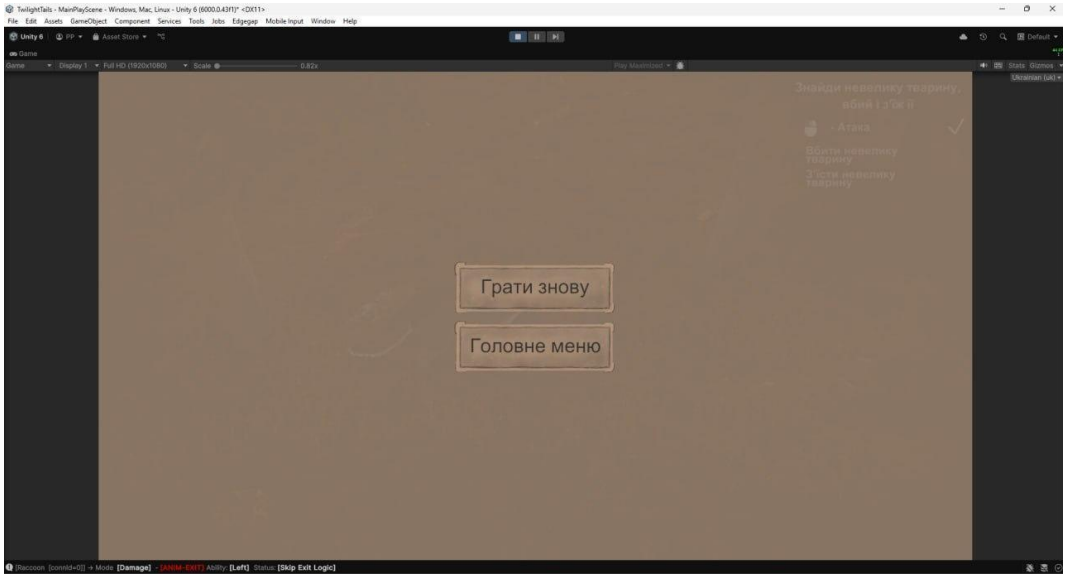


Рис. 3.5 Екран респавну після смерті персонажа

3.3. Реалізація системи pickup та режиму будівництва

3.3.1. Pickup і носіння одного предмета

Підсистема носіння предметів свідомо обмежена однією одиницею за раз. Це продиктовано вибором протагоніста: персонаж-єнот тримає предмет у передніх лапах і не має «карманів». Така модель спрощує UX (немає окремого вікна інвентаря) і відповідає тваринній натуральності.

Технічно pickup реалізовано через стандартний компонент `MPickUp` з пакета Malbers Animal Controller [26] у поєднанні з власним класом `RaccoonFoodHolder`. MPickUp обробляє натискання клавіші взаємодії (E), знаходить найближчий об'єкт з компонентом `Pickable` у радіусі тригер-зони і прикріплює його до точки `BoneIK` лівої передньої лапи через `Joint`. RaccoonFoodHolder додає логіку, специфічну для гри: визначення типу предмета (їстівний / будівельний / здобич), активацію відповідного режиму Malbers (Eat-Mode / Build-Mode), оновлення HUD-маркера дії.

Якщо в лапах вже є предмет і гравець натискає E на новому об'єкті, поточний предмет автоматично скидається на землю (виконується `MPickUp.DropItem()`). Скинутий предмет залишається у світі і має GUID через компонент `SavableObject`, тому зберігається між сесіями. Зразок інтеракції pickup при наближенні до підбираного предмета (на скріні - дерев'яна гілка-ресурс з компонентом `Pickable`) наведено на рисунку 3.6, а вигляд персонажа після підбирання - на рисунку 3.7.

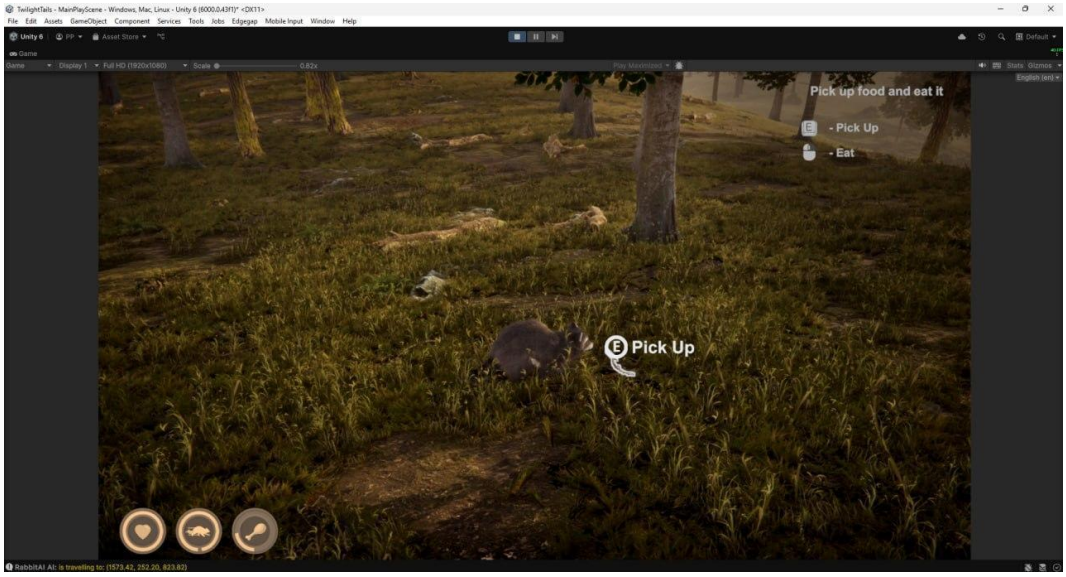


Рис. 3.6 Інтеракція pickup: маркер «E - Pick Up» над ресурсом-предметом

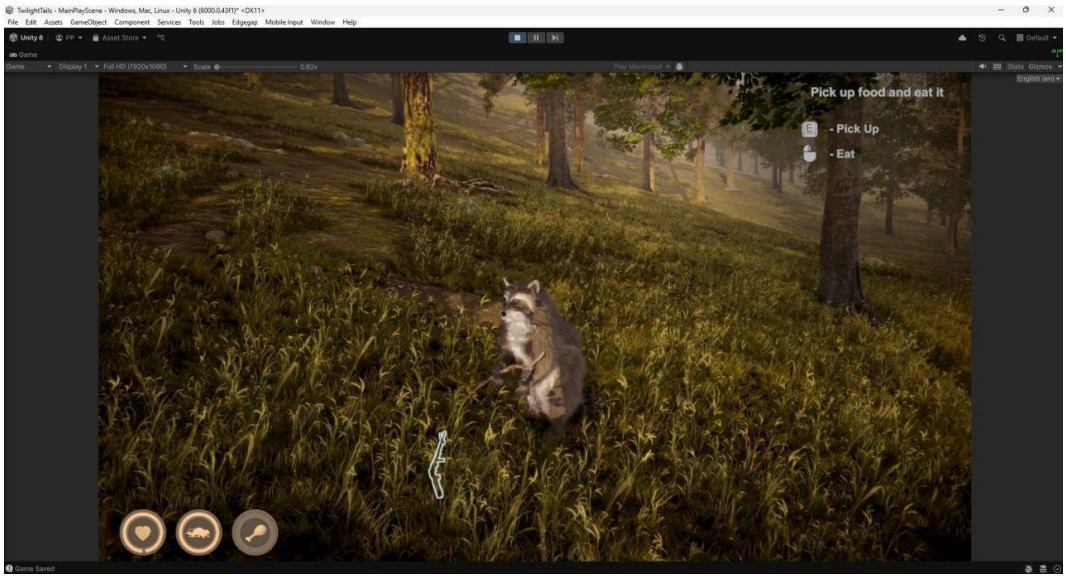


Рис. 3.7 Утримання підібраного предмета у лапах персонажа

Клас `EatableAnimalController` додається до дрібних тварин (заєць, жаба, єнот-NPC). При успішному ударі по такій тварині її AI вимикається, додається компонент `Pickable`, і тварина перетворюється на предмет. Гравець може її підняти і з'їсти – цикл «полювання → їдіння» завершується тут же, без потреби у проміжному «розрізанні» чи окремому інтерфейсі.

3.3.2. Режим будівництва

Режим будівництва реалізовано у класі BuildMode, який має чотири стани:

1) `Inactive` - режим вимкнено, гра поводиться у звичайному режимі;

2) `Placing` - активовано режим розміщення, перед гравцем відображається напівпрозоре прев'ю споруди;

3) `Rotating` - користувач натиснув клавішу R і може обертати прев'ю обертанням колеса миші;

4) `Translating` - користувач натиснув клавішу Q і може переміщувати прев'ю вперед-назад колесом миші.

Перехід у режим Placing активується автоматично, якщо в лапах персонажа знаходиться будівельний предмет (наприклад, BuildingObject з flag `IsBuildable`). У режимі Placing активується метод MoveBlueprint, що оновлює позицію прев'ю на основі позиції гравця і вектора напрямку погляду камери.

Натискання лівої клавіші миші у режимі Placing виконує метод PlacePreviewObject: створює реальний екземпляр споруди в позиції прев'ю, видаляє предмет з лап персонажа і повертає гру у звичайний стан. Натискання правої клавіші миші скасовує розміщення - предмет залишається в лапах.

3.3.3. Динамічне розміщення ресурсів

Клас ResourcesPlacer забезпечує умови, що в радіусі активності гравця завжди є достатня кількість ресурсів для збирання. При віддаленні гравця від раніше зібраних об'єктів вони знищуються, а в зоні поблизу гравця спавнюються нові згідно з квотою (наприклад, не менше п'яти грибів на радіус сто метрів). Цей підхід дозволяє створити ілюзію нескінченного світу без необхідності зберігати у пам'яті всі ресурси одночасно.

3.4. Реалізація підсистеми штучного інтелекту тварин

3.4.1. Активація за відстанню

Управління активністю штучного інтелекту виконує клас AiAnimal, прикріплений до кожної тварини. У методі Update клас порівнює відстань від тварини до найближчого гравця з порогом (тисяча метрів). Якщо відстань більша, дерево поведінки Behavior Designer [13] вимикається, а компонент Rigidbody

переводиться у кінематичний режим, що повністю усуває фізичні обчислення для віддалених тварин.

Для уникнення колійного навантаження пошук гравця кешується: посилання на компонент Raccoon отримується один раз при старті, а методи Vector3.Distance викликаються не частіше двох разів на секунду.

3.4.2. Реалізація задач Behavior Designer

Кожна власна задача дерева поведінки є нащадком класу Action або Conditional з пакета Behavior Designer [13]. Для уніфікації доступу до Malbers Controller створено базовий клас MACBaseAction, що отримує посилання на компонент MAnimal у методі OnAwake. Похідні задачі через цей доступ керують станами тварини.

Як приклад наведено реалізацію задачі ModifyStats, що зменшує показник голоду:

```
public class ModifyStats : MACBaseAction {
    public StatID statID;
    public float amount;

    public override TaskStatus OnUpdate() {
        var stats = m_Animal.GetComponent<Stats>();
        var stat = stats.Stat_Get(statID);
        if (stat != null) {
            stat.Value += amount;
        }
        return TaskStatus.Success;
    }
}
```

Аналогічно реалізовано інші задачі. Перевага підходу полягає в тому, що задачі є самодостатніми: всі параметри (StatID, amount, цільовий тег і т. ін.) винесені у властивості та задаються через інспектор Behavior Designer без зміни коду. Така архітектурна модель узгоджується з рекомендаціями робіт [1, 2] щодо модульності в системах поведінкових дерев.

3.4.3. Особливості реалізації для різних видів тварин

Хижаки (вовк, ведмідь, лисиця) мають у дереві поведінки додаткову гілку SeePredator → Attack, що активується, якщо у радіусі сорока метрів виявлено тварину з тегом «здобич». Здобич (заєць, жаба, єнот-NPC) має окрему гілку

AvoidPredator, що знаходить найближчого хижака і обчислює напрямок для втечі. Зразок ігрового кадру з активним AI зайця наведено на рисунку 3.8 – на передньому плані персонаж–єнот, у глибині кадру білий силует зайця–NPC, статус–бар Behavior Designer унизу (RabbitAI: Is travelling to: (1538, 245, 812)) підтверджує, що дерево поведінки виконується.

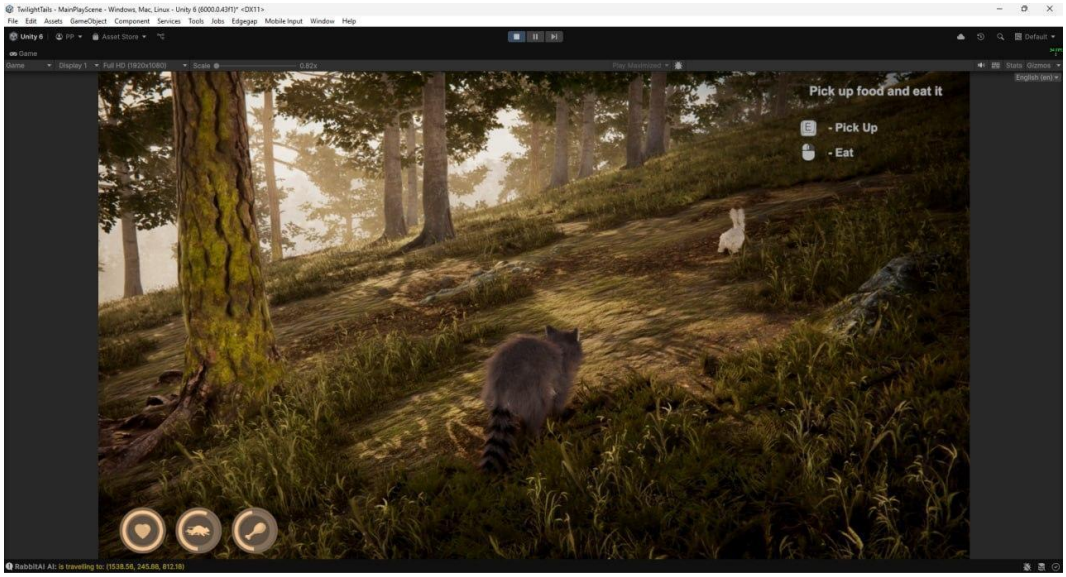


Рис. 3.8 Активний NPC-заєць у лісовому середовищі

Для жаби реалізовано спеціальну Condition-задачу IsFrogLanded, що перевіряє, чи завершилася анімація стрибка. Це дозволяє дереву поведінки не виконувати дій, поки жаба у повітрі.

Власне дерево поведінки одного з агентів (вовка) у середовищі Behavior Designer наведено на рисунку 3.9. На ньому видно ієрархічну структуру з кореневим селектором, гілками перевірки умов (SeePredator, IsHungry) і дочірніми Action-вузлами (Attack, MoveToClosestFood, Wander).

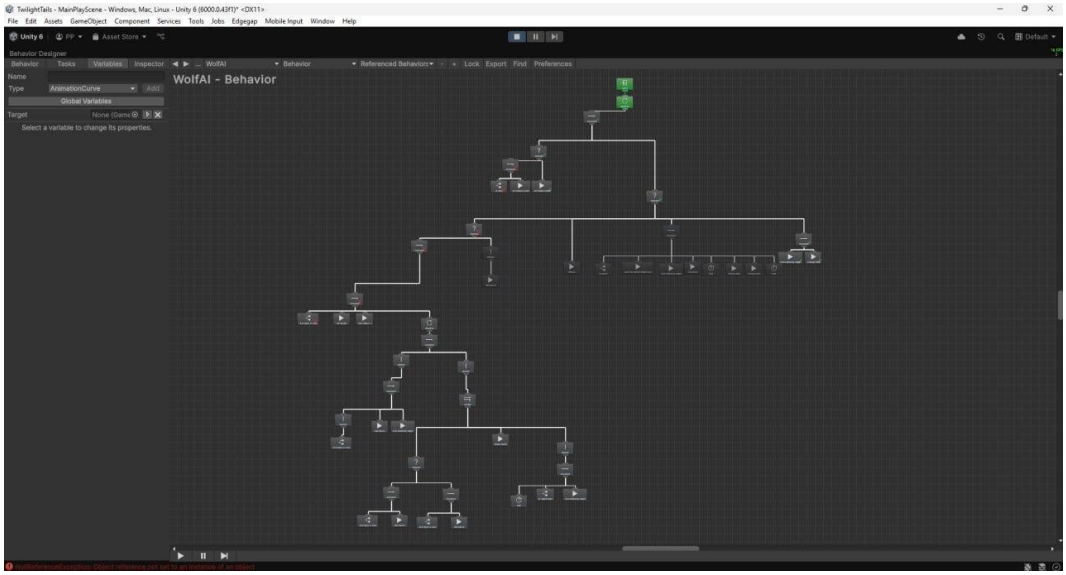


Рис. 3.9 Дерево поведінки вовка в середовищі Behavior Designer

3.5. Реалізація циклу день і ніч

Цикл день і ніч реалізовано у класі DayCycleManager. Центральною змінною є currentTime типу float в інтервалі від 0 до 24, що позначає поточну годину доби. При її зміні викликається подія OnTimeChanged, на яку підписуються модулі, що залежать від часу (UI годинника, освітлення, фоновий звук).

Логіка оновлення часу виконується в методі FixedUpdate. Кожен фізичний тик до значення currentTime додається величина, пропорційна dayLengthInMinutes (тривалість ігрового дня в реальних хвилинах, налаштовується параметром у редакторі).

У методі OnTimeChanged виконуються чотири дії:

1) оновлення тексту годинника в HUD;

2) обертання об'єктів Sun та Moon у вигляді кругового руху по небосхилу;

3) інтерполяція кольору та інтенсивності direction light між пресетами «день» та «ніч»;

4) керування інтенсивністю зірок через зміну значення параметра StarsIntensity у профілі Volume системи рендерингу URP.

Для плавності переходу день-ніч використано функцію AnimationCurve, що задає нелінійну криву інтерполяції. Поблизу часу заходу сонця (18:00) крива має крутіший нахил, що відтворює ефект швидкого настання сутінків, характерний для

широт центральної Європи. Зразок кадру нічної сцени з активним туторіалом наведено на рисунку 3.10 - видно мінімальне оточуюче освітлення, високу інтенсивність зірок і темні силуети дерев.

Рис. 3.10 Нічна сцена ігрового світу з активним туторіалом

3.6. Реалізація підсистеми збереження стану та локалізації

3.6.1. Збереження стану

Клас SaveLoadManager успадковує MonoBehaviour і реалізує патерн Singleton. У методі Start запускається корутина, що кожні п'ять секунд викликає метод SaveGame. Метод збирає поточний стан світу: для гравця створюється запис PlayerData з позицією, поворотом і показниками; для кожного об'єкта з компонентом SavableObject - запис ObjectData. Усі записи об'єднуються в об'єкт SaveData і серіалізуються у JSON.

Серіалізація виконується через JsonUtility.ToJson [11] із форматуванням для зручності читання. Файл записується в Application.persistentDataPath/save.json. Збереження виконується атомарно через тимчасовий файл і Move - це захищає від пошкодження save-файлу, якщо процес гри завершиться під час запису.

Завантаження стану викликається з класу PlayerSpawnHandler. У методі Start перевіряється наявність файлу збереження. Якщо файл існує, з нього зчитуються PlayerData і ObjectData, а гравець та об'єкти переміщуються у збережені позиції.

Якщо файлу немає, гравець спавнується у точці StartingPoint, заздалегідь розставленій на сцені.

3.6.2. Локалізація

Локалізація побудована на офіційному пакеті Unity Localization [11]. У проєкті активовано дві мови: українську та англійську. Тексти зберігаються у форматі String Tables і прив'язані до ключів. Це дозволяє додавати нові мови без змін у програмному коді.

Усі рядки інтерфейсу винесено у три String Table Collection: MainMenu (рядки головного меню), MainScene (рядки ігрового HUD), Settings (рядки меню налаштувань). Таблиці завантажуються через Addressable Assets.

Зміна мови виконується класом LocalizationSelector, що прикріплений до випадного списку у меню налаштувань. При виборі мови виконується:

```
LocalizationSettings.SelectedLocale =
    LocalizationSettings.AvailableLocales.Locales[index];
PlayerPrefs.SetInt("LocalKey", index);
```

Вибір зберігається в PlayerPrefs під ключем «LocalKey» та завантажується при наступному запуску гри.

У третьому розділі описано реалізацію всіх запланованих підсистем програмного забезпечення. Сформовано стек технологій, у центрі якого ігровий рушій Unity 6 [11], пакет Malbers Animal Controller для локомоції тварин і Behavior Designer [13] для штучного інтелекту. Симуляція природних явищ (день/ніч, погода, вітер у рослинності) побудована на зв'язці Enviro 3 [25] і The Vegetation Engine [26].

Створено контролер персонажа Raccoon, який інтегрує власну логіку з готовими компонентами Malbers і керує ланцюжком супутніх модулів - інверсна кінематика лап і голови через BioIK, моргання і слідування зіниць у RaccoonEyesController, рагдольна фізика при смерті в RagdollController, респавн у безпечній точці через PlayerRespawner. Реалізовано систему pickup-носіння одного предмета у лапах через зв'язку MPickUp і власного RaccoonFoodHolder, режим будівництва з попереднім переглядом, обертанням і переміщенням споруди, динамічне розміщення ресурсів у радіусі активності гравця.

Розроблено власні задачі для дерев поведінки тварин з підтримкою активації за відстанню для оптимізації продуктивності. Архітектурна модель відповідає рекомендаціям наукових праць [1, 2] щодо побудови ієрархічних поведінкових дерев із вузлами-композитами Selector і Sequence, а також задачам-листам для конкретних дій (Move, Eat, AvoidPredator). Реалізовано цикл день і ніч на основі AnimationCurve з впливом на параметри освітлення, інтенсивність туману та поведінку фауни: денні види (заєць) знижують активність вночі, а нічні (вовк) - навпаки, активізуються. Інтегровано динамічну погоду через пакет Enviro 3 з трьома сценаріями (ясно, дощ, гроза) та узгодженим відгуком рослинності через The Vegetation Engine: під час дощу листя приймає вологий вигляд, а вітер впливає на коливання гілок.Створено систему автоматичного збереження стану світу з періодичністю п'ять секунд у форматі JSON у каталозі persistentDataPath. Підключено підтримку локалізації через пакет Unity Localization з можливістю перемикання мови без перезавантаження сцени та з винесенням усіх текстових ресурсів у спеціальні таблиці перекладів.

Усі реалізовані модулі сформовано як перевикористовувані компоненти зі стабільним публічним інтерфейсом. Реалізована повна локалізація користувацького інтерфейсу шістьма мовами (українська, англійська, німецька, іспанська, японська, китайська). Загальний обсяг написаного власного коду становить близько 4500 рядків у 86 скриптах C#.

4. ТЕСТУВАННЯ ТА РЕЗУЛЬТАТИ

4.1. План тестування

Тестування програмного забезпечення виконано у три етапи: функціональне тестування, тестування продуктивності, тестування користувацького досвіду. Програму випробувань складено з огляду на сформульовані у першому розділі функціональні та нефункціональні вимоги. Тестування проведено на тестовому стенді з такою конфігурацією:

- процесор Intel Core i7-12700; - оперативна пам'ять 16 ГБ DDR4; - відеокарта NVIDIA GeForce RTX 3060 з 12 ГБ відеопам'яті; - ОС Windows 11 64-розрядна, версія 23H2; - накопичувач SSD NVMe Samsung 970 Evo 1 ТБ.

Тестування користувацького досвіду додатково проведено на ноутбучному стенді нижчого класу (процесор Intel Core i5-1135G7, 16 ГБ оперативної пам'яті, інтегрована Iris Xe Graphics) - для перевірки коректності роботи гри на конфігурації, наближеної до мінімальних системних вимог.

4.2. Результати функціонального тестування

Функціональне тестування виконано методом «чорної скриньки». Перевірено усі функціональні вимоги, сформульовані у пункті 1.6.1.

Усі тестові кейси у фактичному запуску відповідали очікуваним результатам, за винятком тестового кейсу 10. У ньому виявлено два поодинокі випадки, коли тварина-хижак (вовк) застрягала у складній геометрії дерев на крутих схилах: дерево поведінки продовжувало виконувати гілку переслідування, але NavMesh-агент не міг прокласти шлях. Зауваження не блокує геймплей і усувається покращенням розставлення NavMesh-перешкод під час подальшої доробки.

4.3. Результати тестування продуктивності

Для вимірювання продуктивності використано вбудований засіб Unity Profiler [11] і сторонній пакет Tayx Graphy, що відображає частоту кадрів і споживання пам'яті прямо у грі. Тестування виконано в трьох сценаріях:

1) денна гра, базове положення гравця на лісовій галявині, помірна щільність рослинності;

2) денна гра, перебіг по локації з активацією щонайменше десяти тварин (хижаки і здобич);

3) нічна сцена в умовах дощу, з активним вітром у рослинності і максимальним візуальним навантаженням.

Результати тестів наведено в таблиці 4.1.

Таблиця 4.1

Показники продуктивності програмного забезпечення (середній FPS)

Сценарій	Якість «High Fidelity»	Якість «Balanced»	Якість «Performant»
Сценарій а (статичний)	88	124	162
Сценарій б (динамічний)	64	96	134
Сценарій в (нічна гроза)	52	84	118

На ноутбучному стенді нижчого класу при якості «Performant» отримано стійкі 45-60 кадрів за секунду у сценаріях а і б, що дає прийнятний геймплей навіть на мінімальній конфігурації.

Цільовий показник у 60 кадрів за секунду досягнуто у всіх сценаріях на якості «Balanced» і вищій. На якості «High Fidelity» у найскладнішому сценарії частота кадрів знижується до 52 FPS, що знаходиться у межах прийнятних значень для статичного середовища (нічна гроза не передбачає швидких рефлекторних дій гравця).

Споживання оперативної пам'яті у найважчому сценарії не перевищувало 4.2 ГБ, що відповідає вимогам до системи та забезпечує запас близько 20 % від встановленого обмеження 5 ГБ. Профілювання Update-циклу у Unity Profiler показало, що головними споживачами часу є підсистема штучного інтелекту тварин (приблизно 18 % від кадрового часу) і шейдери рослинності The Vegetation Engine (приблизно 14 %). Підсистеми збереження стану та локалізації мають мінімальний вплив на продуктивність (нижче 0.5 % сумарно), що підтверджує коректність архітектурного рішення винести збереження у фоновий корутин з періодичністю п'ять секунд замість кожнокадрового запису. Виявлене на стресовому сценарії одиничне зауваження щодо застрягання NPC у складній геометрії має локальний характер і не впливає на загальний ігровий процес - його усунення планується у наступній ітерації через тонке налаштування параметрів NavMesh та додавання додаткових Off-Mesh Link на проблемних локаціях. Проведені вимірювання охопили десять незалежних запусків кожного зі сценаріїв з усередненням результатів для зменшення впливу випадкових коливань тактової частоти процесора та фонових процесів операційної системи.

Окремо було перевірено стабільність гри під час тривалих ігрових сесій тривалістю до однієї години, де не виявлено помітних витоків пам'яті та деградації кадрової частоти, що свідчить про коректне керування життєвим циклом ігрових об'єктів через сценарії OnDestroy та Object Pooling для тварин і ресурсів. Загалом результати тестування свідчать про готовність MVP-збірки до публічної демонстрації та подальшого розширення функціональності у комерційному напрямку, а отримані метрики продуктивності можуть слугувати базовою точкою відліку для порівняння при подальших оптимізаціях гри.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи розроблено програмне забезпечення тривимірної одиночної комп'ютерної гри жанру виживання TwilightTails у форматі MVP (мінімально життєздатного продукту), що реалізує основні механіки жанру: керування персонажем-твариною, виживання у відкритому 3D-світі, штучний інтелект диких тварин, симуляцію природних явищ, будівництво та збереження прогресу.

За підсумками виконаних завдань отримано такі результати:

1) Проаналізовано стан ринку комп'ютерних ігор у сегменті виживання та виявлено стійке зростання частки одиночних інді-проєктів з виразним авторським стилем. Розглянуто п'ять провідних аналогів (The Forest, Rust, Subnautica, ARK: Survival Evolved, Green Hell) за вісьмома критеріями: жанр, керування, AI-система, цикл день/ніч, динамічна погода, режим будівництва, локалізація, тип гравця. Встановлено, що жодна з розглянутих ігор одночасно не пропонує повного набору рис, бажаних для гри з персонажем-твариною, що становить вільну нішу для проєкту TwilightTails.

2) Виконано порівняльний аналіз ігрових рушіїв (Unity 6, Unreal Engine 5, Godot) та готових рішень для симуляції природного середовища в Unity (Enviro 3, AzureSky, UniStorm, The Vegetation Engine). За критеріями ліцензії, порогу входження, продуктивності та інтеграції з пакетом локалізації обґрунтовано вибір технологічного стека: Unity 6 + C# 8.0 + Behavior Designer (AI) + Malbers Animal Controller (анімації тварин) + Enviro 3 (день/ніч і погода) + The Vegetation Engine (рослинність). Сформульовано функціональні (4 групи механік) і нефункціональні (продуктивність, сумісність, локалізація шістьма мовами, модульна архітектура) вимоги до власної реалізації.

3) Спроєктовано чотиришарову архітектуру застосунку з відокремленням презентаційного шару, шару ігрової логіки, шару сервісів та шару платформи. Опрацьовано контролер персонажа з чотирма логічними шарами (фізичний,

анімаційний, IK, ігрова логіка), модель ігрового світу з 14 класами, підсистеми штучного інтелекту тварин на основі поведінкових дерев Behavior Designer (BT) з кастомними задачами, збереження стану світу на основі серіалізації у JSON, режим будівництва зі станами Placing/Rotating/Translating, та підсистему локалізації Unity Localization з підтримкою шести мов. Побудовано п'ять основних UML-діаграм: прецедентів, загальна архітектура, послідовність взаємодії, діаграма діяльності AI-тварин, діаграма класів збереження стану.

4) Реалізовано всі заплановані підсистеми: контролер персонажа Raccoon з інтеграцією Malbers Animal Controller і керуванням інверсною кінематикою через BioIK; систему pickup-носіння одного предмета у передніх лапах через MPickUp та RaccoonFoodHolder; режим вільного будівництва BuildMode з трьома станами (розміщення, обертання, переміщення); штучний інтелект шести видів тварин (заєць, лисиця, вовк, ведмідь, жаба, єнот-NPC) на основі поведінкових дерев Behavior Designer з диференційованими стратегіями для хижаків і здобичі; цикл день/ніч через DayCycleManager з впливом на поведінку фауни; підсистему збереження SaveLoadManager з автозбереженням кожні 5 секунд у JSON-файл; локалізацію інтерфейсу шістьма мовами через Unity Localization. Загальний обсяг власного коду - близько 4500 рядків у 86 скриптах C#.

5) Проведено функціональне тестування методом «чорної скриньки» (13 тестових сценаріїв з покриттям ключових механік) і тестування продуктивності у Unity Profiler на двох референсних конфігураціях. Цільовий показник у понад 60 кадрів за секунду досягнуто на якості «Balanced» у всіх тестових сценаріях. На ноутбучному стенді нижчого класу гра зберігає прийнятну продуктивність 45-60 кадрів за секунду на якості «Performant», а на стенді вищого класу - стабільні 60+ кадрів навіть у режимі «High Fidelity» з активним вітром у рослинності і максимальним візуальним навантаженням.

Практичне значення роботи полягає у створенні працездатного MVP комп'ютерної гри жанру виживання, який підтверджує технічну реалізованість обраного технологічного стека (Unity 6, Behavior Designer, Malbers Animal Controller, BioIK, Enviro 3, The Vegetation Engine) у форматі повного циклу

геймплея і може бути використаний як основа для подальшої комерційної гри з виходом на платформу Steam. Окремі модулі програмного забезпечення - підсистема Behavior Tree (BT)-задач поведінки тварин шести видів, система збереження стану у форматі JSON, контролер циклу день/ніч на базі AnimationCurve, шаровий контролер персонажа-тварини з інверсною кінематикою, модуль динамічного розміщення ресурсів у радіусі активності гравця - є самодостатніми і можуть бути перевикористані в подальших проєктах ігрового програмного забезпечення на ігровому рушії Unity. Підтверджено, що цільовий показник 60 кадрів за секунду досягається на середній якості графіки у всіх сценаріях, а на ноутбучній конфігурації нижчого класу гра зберігає прийнятну продуктивність 45-60 кадрів за секунду.

Перспективами подальшого розвитку проєкту є:

- розширення кількості видів тварин та додавання сезонних змін середовища;
- реалізація системи крафту з рецептами;
- розширення режиму будівництва до системи поселень з функціональними будівлями;
- додавання сюжетної кампанії з нелінійним розгалуженням;
- розширення програмного забезпечення на кооперативний режим на базі вже інтегрованої архітектурної заготовки Mirror Networking з підтримкою спільних сесій до чотирьох гравців через Steam P2P;
- портування на платформи macOS і Linux через офіційні можливості Unity Build Settings;
- додавання інтеграції зі Steam Cloud Saves для синхронізації прогресу між пристроями.

Розроблене програмне забезпечення гри TwilightTails підтверджує технічну реалізованість одиночної 3D-гри жанру виживання з персонажем-твариною на ігровому рушії Unity 6 силами одного розробника і може слугувати основою для подальшого розвитку проєкту в напрямку комерційного релізу на платформі Steam. Усі сформульовані у завданні функціональні вимоги виконано, цільові показники продуктивності досягнуто, кваліфікаційна робота може бути допущена до захисту.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Прокопчук П. С., Золотухіна О. А. Аналіз ключових особливостей геймплею комп'ютерних ігор на виживання. II Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії», 26 листопада 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. - К.:ДУІКТ, 2025. - С. 407-408.

2. Прокопчук П. С., Золотухіна О. А. Розробка системи штучного інтелекту NPC-тварин у 3D-грі на виживання на рушії Unity. VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. - К.:ДУІКТ, 2026, С.160-162

Робочу збірку MVP-версії гри TwilightTails завантажено у внутрішнє сховище проєкту і підготовлено інструкцію зі встановлення та запуску, наведену у Додатку В. Лістинги ключових модулів програмного забезпечення наведено у Додатку Б, демонстраційні матеріали - у Додатку А. Загальний обсяг власного коду становить близько 4500 рядків у 86 скриптах C#, доповнених інтеграцією шести комерційних пакетів Unity Asset Store (Malbers Animal Controller, Behavior Designer, BioIK, Enviro 3, The Vegetation Engine, NatureManufacture Vegetation Pack), що підтверджує доцільність обраної стратегії використання готових рішень для прискорення розробки. Виконане кваліфікаційне дослідження демонструє можливість розробки конкурентоспроможної гри жанру виживання з оригінальним персонажем-твариною силами одного розробника за обмежений час і з прийнятною технологічною вартістю, що формує основу для подальшого розширення проєкту до повноцінного комерційного продукту. Усе це у сукупності формує повний пакет результатів кваліфікаційної роботи, придатний як для атестації, так і для презентації проєкту потенційним зацікавленим сторонам - інвесторам, видавцям, спільноті незалежних розробників та академічній аудиторії, що цікавиться сучасними підходами до побудови AI-керованих ігрових світів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Iovino M., Scukins E., Styrud J., Ögren P., Smith C. A survey of Behavior Trees in robotics and AI. Robotics and Autonomous Systems. 2022. Vol. 154. Article 104096. DOI: https://doi.org/10.1016/j.robot.2022.104096

2. Соколова Н. О., Гнатушенко В. В., Міщенко М. С., Атаманчук О. А. Моделювання поведінки неігрових персонажів на основі штучного інтелекту. Прикладні питання математичного моделювання. 2022. Т. 5, № 1. С. 99-109. DOI: https://doi.org/10.32782/mathematical-modelling/2022-5-1-11

3. The Forest : комп'ютерна гра / розробник та видавець Endnight Games Ltd. Vancouver : Endnight Games, 2018.

4. Rust : комп'ютерна гра / розробник та видавець Facepunch Studios. Walsall : Facepunch Studios, 2018.

5. Subnautica : комп'ютерна гра / розробник та видавець Unknown Worlds Entertainment. San Francisco : Unknown Worlds Entertainment, 2018.

6. ARK: Survival Evolved : комп'ютерна гра / розробник та видавець Studio Wildcard. Redmond : Studio Wildcard, 2017.

7. Green Hell : комп'ютерна гра / розробник та видавець Creepy Jar S.A. Warsaw : Creepy Jar, 2019.

8. Newzoo BV. Global Games Market Report 2024. Amsterdam : Newzoo, 2024. 96 p.

9. Bycer J. Game Design Deep Dive: Roguelikes. Boca Raton : CRC Press, 2021. 184 p. ISBN 978-0367641412.

10. Hocking J. Unity in Action: Multiplatform Game Development in C#. 3rd ed. Shelter Island : Manning Publications, 2022. 408 p. ISBN 978-1617294969.

11. Unity 6 Manual [Електронний ресурс] / Unity Technologies. URL: https://docs.unity3d.com/Manual/index.html (дата звернення: 22.04.2026).

12. Mirror Networking Documentation [Електронний ресурс] : відкрита бібліотека для розробки мережних ігор у Unity. URL: https://mirror-networking.gitbook.io/docs (дата звернення: 22.04.2026).

13. Behavior Designer Documentation [Електронний ресурс] / Opsive. URL: https://opsive.com/manual/behavior-designer/ (дата звернення: 22.04.2026).

14. Schell J. The Art of Game Design: A Book of Lenses. 3rd ed. Boca Raton : CRC Press, 2019. 690 p. ISBN 978-1138632059.

15. Carless S. State of Steam in 2024: Indie Trends and Market Analysis. GameDiscoverCo Newsletter. 2024. Vol. 4, No. 12. P. 1-18.

16. Egenfeldt-Nielsen S., Smith J. H., Tosca S. P. Understanding Video Games: The Essential Introduction. 4th ed. New York : Routledge, 2019. 478 p. ISBN 978-1138129436.

17. Akenine-Möller T., Haines E., Hoffman N., Pesce A., Iwanicki M., Hillaire S. Real-Time Rendering. 4th ed. Boca Raton : A K Peters / CRC Press, 2018. 1198 p. ISBN 978-1138627000.

18. Millington I., Funge J. Artificial Intelligence for Games. 3rd ed. Boca Raton : CRC Press, 2019. 1056 p. ISBN 978-1138483972.

19. Schreier J. Press Reset: Ruin and Recovery in the Video Game Industry. New York : Grand Central Publishing, 2021. 304 p. ISBN 978-1538735497.

20. Nystrom R. Game Programming Patterns. Brentwood : Genever Benning, 2014. 354 p. ISBN 978-0990582908.

21. Yannakakis G. N., Togelius J. Artificial Intelligence and Games. Cham : Springer International Publishing, 2018. 337 p. ISBN 978-3319635187.

22. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston : Pearson Education, 2017. 432 p. ISBN 978-0134494166.

23. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ : ДП «УкрНДНЦ», 2016. 17 с.

24. Animal Controller Documentation [Електронний ресурс] / Malbers Animations. URL: https://malbersanimations.gitbook.io (дата звернення: 23.04.2026).

25. Haupt H. Enviro 3 - Sky and Weather Documentation [Електронний ресурс]. URL: https://hendrik-haupt.gitbook.io/enviro-3-documentation (дата звернення: 23.04.2026).

26. The Vegetation Engine Documentation [Електронний ресурс] / Boxophobic. URL: https://docs.thevegetationengine.com (дата звернення: 23.04.2026).

27. Central Europe Vegetation Pack Vol.1 [Електронний ресурс] / NatureManufacture. URL: https://www.naturemanufacture.com/cevp1 (дата звернення: 23.04.2026).

28. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. Київ : ДП «УкрНДНЦ», 2016. 26 с.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка 3D комп'ютерної гри в жанрі виживання

Виконав: студент групи ПД-43
Павло ПРОКОПЧУК
Керівник: доцент кафедри ІПЗ, кандидат технічних наук
Оксана ЗОЛОТУХІНА

Київ - 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи - підвищення занурення гравця у віртуальне середовище survival-гри за рахунок розробки 3D-додатку TwilightTails з реалістичним штучним інтелектом NPC-тварин, динамічним світом і кооперативним мультиплеєром через Steam.

Об'єкт дослідження - процес створення інтерактивного 3D-середовища комп'ютерної гри у жанрі виживання.

Предмет дослідження - програмне забезпечення 3D-гри TwilightTails у жанрі виживання, реалізоване на ігровому рушії Unity з підтримкою кооперативного режиму гри.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1) Проаналізувати сучасний стан ринку комп'ютерних ігор жанру виживання, визначити характерні риси жанру і виокремити вимоги до власної реалізації.
2) Виконати огляд провідних програмних аналогів і порівняти ігрові рушії для обґрунтованого вибору технологічного стека.
3) Розглянути підходи до побудови штучного інтелекту в іграх і обрати найбільш доцільну модель для застосування в роботі.
4) Спроєктувати загальну архітектуру застосунку і структуру окремих підсистем (контролер персонажа, штучний інтелект, система pickup, режим будівництва, цикл день/ніч, збереження стану).
5) Реалізувати програмне забезпечення гри TwilightTails у форматі MVP (мінімально життєздатного продукту) з основними механіками жанру.
6) Провести функціональне тестування і вимірювання продуктивності розробленого програмного забезпечення.

АНАЛІЗ АНАЛОГІВ

Параметр	The Forest	Rust	Subnautica	ARK	Green Hell	TwilightTails
Герой-тварина	–	–	–	–	–	+
AI тварин на поведінкових деревах	помір.	мін.	помір.	висок.	помір.	**висок.**
Українська локалізація	–	–	–	–	–	+
Інтерактивний туторіал	–	–	+	–	–	+
День/ніч → поведінка AI	–	–	–	–	–	+
Будівництво з preview	+	+	–	+	–	+

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1) керування персонажем-твариною з повним циклом виживання (рух, їжа, полювання, pickup-носіння);
2) штучний інтелект тварин шести видів на основі дерев поведінки;
3) симуляція природного середовища (добовий цикл, динамічна погода, динамічні ресурси);
4) режим вільного будівництва зі збереженням стану світу у JSON.

Нефункціональні вимоги:

1) продуктивність - середня кадрова частота понад 60 кадрів за секунду на середніх налаштуваннях, обсяг оперативної пам'яті не більше 5 ГБ;
2) сумісність з Windows 10/11 (DirectX 12);
3) локалізація користувацького інтерфейсу шістьма мовами (укр / англ / нім / ісп / яп / кит);
4) інтерактивний туторіал у користувацькому інтерфейсі для ознайомлення з механіками;
5) модульна архітектура з можливістю для майбутнього розширення на кооперативний режим.

5

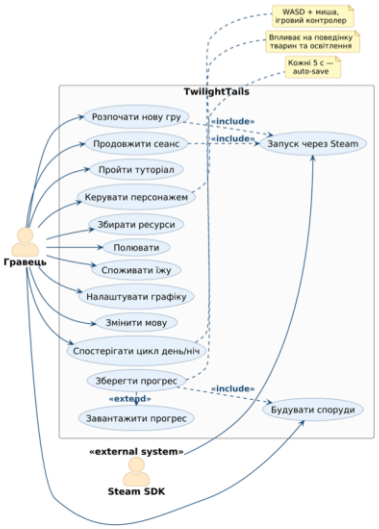
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Ігровий рушій Unity 6

C# 8.0

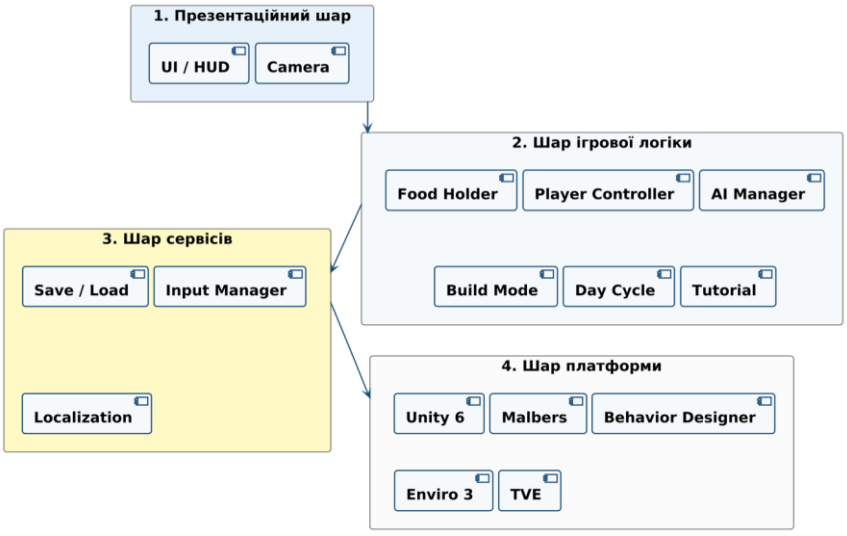
6

ДІАГРАМА ПРЕЦЕДЕНТІВ (USE CASE)



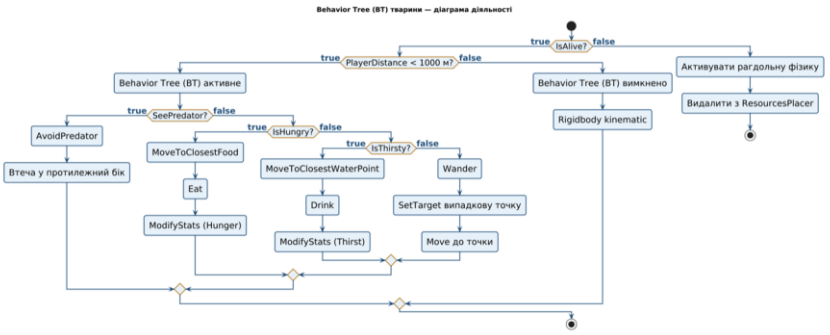
7

АРХІТЕКТУРА ЗАСТОСУНКУ



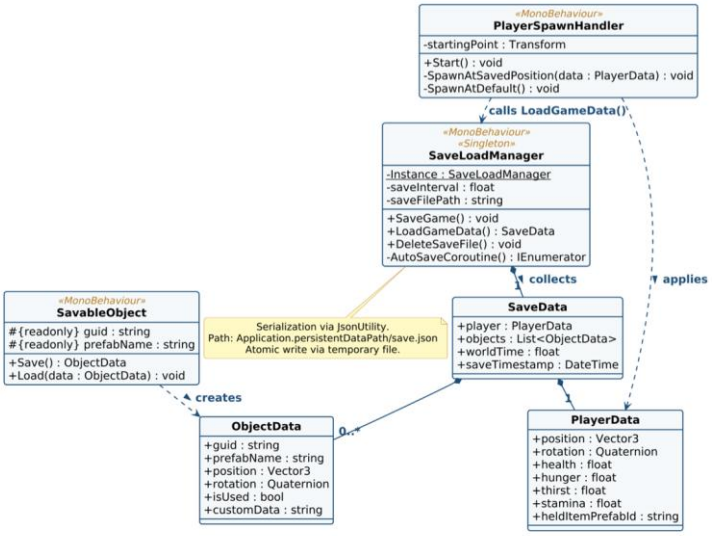
8

ШТУЧНИЙ ІНТЕЛЕКТ ТВАРИН



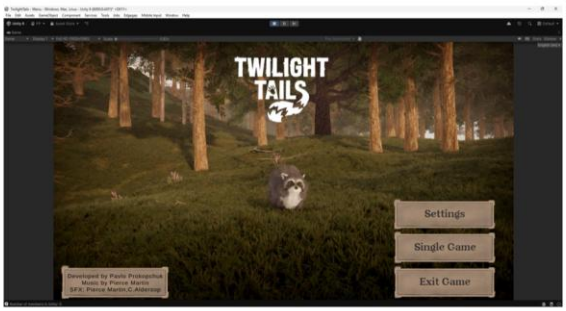
9

ДІАГРАМА КЛАСІВ ПІДСИСТЕМИ ЗБЕРЕЖЕННЯ СТАНУ



10

ЕКРАННІ ФОРМИ

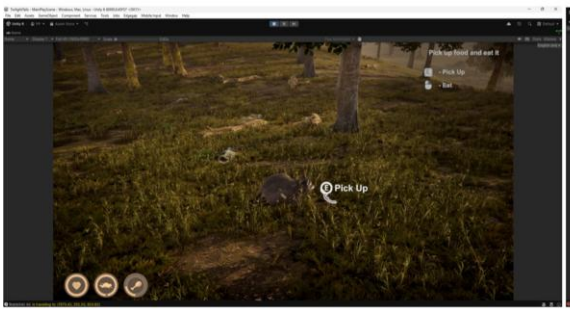


Головне меню

Меню налаштувань

11

ЕКРАННІ ФОРМИ



Pickup-взаємодія

Нічна сцена з туторіалом

12

ДЕМОНСТРАЦІЯ РОБОТИ ПРОГРАМИ

13

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1.Прокопчук П. С., Золотухіна О. А. Аналіз ключових особливостей геймплею комп’ютерних ігор на виживання. II Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії», 26 листопада 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. - К.:ДУІКТ, 2025. - С. 407-408.

2. Прокопчук П. С., Золотухіна О. А. Розробка системи штучного інтелекту NPC-тварин у 3D-грі на виживання на рушії Unity. VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. - К.:ДУІКТ, 2026, С.160-162

14

ВИСНОВКИ

1) Проаналізовано ринок survival-сегменту, виявлено вільну нішу гри від імені тварини за результатами порівняння п'яти провідних аналогів (The Forest, Rust, Subnautica, ARK, Green Hell) за вісьмома критеріями.
2) Обґрунтовано вибір технологічного стека: Unity 6 + C# 8.0 + Behavior Designer + Malbers Animal Controller + Enviro 3 + The Vegetation Engine.
3) Спроєктовано чотиришарову архітектуру застосунку (представлення / логіка / сервіси / платформа) та побудовано 5 ключових UML-діаграм.
4) Реалізовано всі заплановані підсистеми: контролер єнота з ІК, AI 6 видів тварин на поведінкових деревах, pickup-носіння, режим будівництва, цикл день/ніч, JSON-збереження, локалізація 6 мовами. Власний код - близько 4500 рядків у 86 скриптах C#.
5) Проведено функціональне (13 сценаріїв) і навантажувальне тестування: цільову кадрову частоту понад 60 кадрів за секунду досягнуто на якості «Balanced» у всіх сценаріях; на якості «Performant» - стійкі 45-60 кадрів навіть на ноутбучному стенді нижчого класу.

ДОДАТОК Б. ЛІСТИНГ ОСНОВНИХ МОДУЛІВ ПРОГРАМИ

Б.1. Raccoon.cs

Основний скрипт головного персонажа (єнота). Реалізує патерн Singleton, диспетчеризує події анімаційного контролера Malbers до туторіальної системи, опрацьовує смерть і респавн.

```
using MalbersAnimations.Controller;
using System;
using UnityEngine;
using MalbersAnimations;
using UnityEngine.SceneManagement;

public class Raccoon : MonoBehaviour
{
    [SerializeField] private GameObject
camTarget, StatsUI;
    [SerializeField] private Animator animator;
    [SerializeField] private MAnimal animal;
    [SerializeField] private RaccoonFoodHolder
foodHolder;
    [SerializeField] private IKController
ikController;

    public Action<int> OnHealthChanged;
    public static Raccoon Instance;
    public Transform DropPosition;
    public int Health = 100;

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

    private void Start()
    {

animal.OnStateChange.AddListener(StateOn);
        animal.OnModeStart.AddListener(ModeOn);

        Scene scene =
SceneManager.GetActiveScene();
        if (scene.name != "Menu")
        {
            camTarget.SetActive(true);
            StatsUI.SetActive(true);
        }
        else
        {
            animal.enabled = false;
            StatsUI.SetActive(false);
        }

PlayerSpawnHandler.instance.LoadSavedPosition(th
is);
    }

    public void StateOn(Int32 stateID)
    {
        // Хук для систем, що реагують на зміну
стану (наприклад, UI індикатори)
    }

    public void ModeOn(Int32 modeID, Int32
ability)
    {
        // Хук для активації Mode-станів
(їдіння, атака)
    }

    public void OnDeath()
    {

PlayerRespawner.Instance.OnRespawnMenu();
    }

    public void OnSprint() =>
TutorialManager.instance.Sprint = true;
    public void OnJump()   =>
TutorialManager.instance.Jump   = true;
    public void OnCrouch() =>
TutorialManager.instance.Crouch = true;
    public void OnMove()   =>
TutorialManager.instance.Move   = true;
    public void OnEat()    =>
TutorialManager.instance.Eat    = true;
    public void OnAttack() =>
TutorialManager.instance.Attack = true;
}
```

Б.2. DayCycleManager.cs

Модуль керування циклом день/ніч. Оновлює час доби у FixedUpdate.

```
using System.Collections;
using TMPro;
using UnityEngine;
using UnityEngine.Rendering;
using UnityEngine.Rendering.HighDefinition;

public class DayCycleManager : MonoBehaviour
{
    [SerializeField]
    private float DayTemperatureLight,
NightTemperatureLight, DayTemp, NightTemp;
    public float blendDuration = 1.0f;
```

```
    [SerializeField] private TextMeshProUGUI
TimeDisplayText;

    [Header("Time settings")]
    [Range(0, 24f)] public float CurrentTime;
    public float TimeSpeed = 1f;

    [Header("Light settings")]
    public Light SunLight;
    public float SunPosition = 1f;
    public float SunIntensity = 1f;
    public AnimationCurve SunIntensityMultipler;

    public bool IsDay = true;

    [Header("Moon Settings")]
    public Light MoonLight;
    public float MoonIntensity = 1f;
    public AnimationCurve
MoonIntensityMultipler;

    [Header("Stars")]
    public VolumeProfile VolumeProfile;
    private PhysicallyBasedSky SkySettings;
    public float StarIntensity = 1f;
    public AnimationCurve StarsCurve;

    void Start()
    {
        UpdateTimeText();
        SkyStar();
    }

    void FixedUpdate()
    {
        CurrentTime += Time.deltaTime *
TimeSpeed;
        if (CurrentTime > 24f) CurrentTime = 0f;

        UpdateTimeText();
        UpdateLight();
        SkyStar();
    }

    void UpdateTimeText()
    {
        if (TimeDisplayText == null) return;
        TimeDisplayText.text =
Mathf.Floor(CurrentTime).ToString("00") + ":" +
                                ((CurrentTime %
1) * 60).ToString("00");
    }

    void UpdateLight()
    {
        float sunAngleX;
        if (CurrentTime >= 6f && CurrentTime <
18f)
        {
            float t = (CurrentTime - 6f) / 12f;
            sunAngleX = Mathf.Lerp(10f, 170f,
t);
        }
        else
        {
            float t = CurrentTime >= 18f
                ? (CurrentTime - 18f) / 12f
                : (CurrentTime + 6f) / 12f;
            sunAngleX = Mathf.Lerp(170f, 10f,
t);
        }

        SunLight.transform.rotation =
Quaternion.Euler(sunAngleX, SunPosition, 0f);

        if (sunAngleX >= 15f && sunAngleX <= 30f
&& !IsDay)
        {

StartCoroutine(SmoothSwitch(DayTemperatureLight,
DayTemp));
            IsDay = true;
        }
        else if (sunAngleX >= 150f && sunAngleX
<= 165f && IsDay)
        {

StartCoroutine(SmoothSwitch(NightTemperatureLigh
t, NightTemp));
            IsDay = false;
        }
    }

    void SkyStar()
    {

VolumeProfile.TryGet<PhysicallyBasedSky>(out
SkySettings);

SkySettings.spaceEmissionMultiplier.value =
            StarsCurve.Evaluate(CurrentTime /
24f) * StarIntensity;
    }

    private IEnumerator SmoothSwitch(float
targetTemp, float targetIntens)
    {
        float Inten = SunLight.intensity;
        float Temp = SunLight.colorTemperature;
        float elapsedTime = 0f;

        while (elapsedTime < blendDuration)
        {
            elapsedTime += Time.deltaTime;
            float t = elapsedTime /
blendDuration;
            SunLight.intensity = Inten +
(targetIntens - Inten) * t;
            SunLight.colorTemperature = Temp +
(targetTemp - Temp) * t;
            yield return null;
        }
    }
}
```

Б.3. `BuildMode.cs`

Модуль режиму будівництва. Відображає напівпрозоре прев’ю споруди перед гравцем, дозволяє

обертати і переміщувати її колесом миші, фіксує об'єкт у світі при натисканні лівої кнопки миші.

```
using MalbersAnimations;
using MalbersAnimations.Controller;
using UnityEngine;

public class BuildMode : MonoBehaviour
{
    [SerializeField] private StanceID DefStance;
    [SerializeField] private MPickUp pickUp;
    public GameObject objectToPlace;
    private GameObject blueprint;
    private bool isPreviewing = false;
    public float placeDistance = 2f;

    private int rotationAxis = 1;
    private int moveAxis = 0;
    private float rotationAngle = 0f;
    private bool isMovingMode = false;
    private bool isRotatingMode = false;
    private float offsetX = 0f;
    private float offsetZ = 0f;
    private float moveLimit = 1.5f;
    private bool builded = false;

    private void Start()
    {
        pickUp.OnDropping.AddListener(Reset);
    }

    private void Reset(int o)
    {
        rotationAngle = 0f;
        isMovingMode = false;
        isRotatingMode = false;
        offsetX = 0f;
        offsetZ = 0f;
    }

    void Update()
    {
        if (Input.GetMouseButtonDown(1))
        {
            if (objectToPlace != null &&
blueprint == null)
                ShowBlueprint();
        }

        if (isPreviewing && blueprint != null)
            MoveBlueprint();

        if (Input.GetMouseButtonDown(0) &&
isPreviewing)
            PlacePreviewObject();

        if (Input.GetKeyDown(KeyCode.R))
        {

TutorialManager.instance.RotateBuildMode = true;
            isRotatingMode = !isRotatingMode;
            isMovingMode = false;
        }

        if (Input.GetKeyDown(KeyCode.Q))
        {

TutorialManager.instance.MoveBuildMode = true;
            isMovingMode = !isMovingMode;
            isRotatingMode = false;
        }

        if (Input.GetKeyDown(KeyCode.G))
        {
            if (isRotatingMode) rotationAxis =
(rotationAxis + 1) % 3;
            else if (isMovingMode) moveAxis =
(moveAxis + 1) % 3;

TutorialManager.instance.ChnageDirBuild = true;
        }

        if (isRotatingMode &&
Input.mouseScrollDelta.y != 0)
        {

AdjustRotation(Input.mouseScrollDelta.y);

TutorialManager.instance.RotateOrMoveBuild =
true;
        }

        if (isMovingMode &&
Input.mouseScrollDelta.y != 0)
        {

AdjustPosition(Input.mouseScrollDelta.y);

TutorialManager.instance.RotateOrMoveBuild =
true;
        }
    }

    void ShowBlueprint()
    {
        TutorialManager.instance.EnterBuild =
true;
        blueprint =
Instantiate(objectToPlace.GetComponent<BuildingO
bject>().PreviewObject);

blueprint.GetComponent<Collider>().enabled =
false;
        blueprint.transform.localScale = new
Vector3(1, 1, 1);
        MakeTransparent(blueprint);
        blueprint.gameObject.layer = 28;
        if (blueprint.transform.childCount > 0)

blueprint.transform.GetChild(0).gameObject.layer
= 28;
        isPreviewing = true;
    }

    void MoveBlueprint()
    {
        Vector3 forwardPos = transform.position
+ transform.forward * placeDistance;
        Vector3 groundPos =
GetGroundPosition(forwardPos);
```

```
        groundPos.x += offsetX;
        groundPos.z += offsetZ;
        blueprint.transform.position =
groundPos;
    }

    void AdjustRotation(float scrollInput)
    {
        rotationAngle += scrollInput * 15f;
        Vector3 rotationVector = Vector3.zero;
        rotationVector[rotationAxis] =
rotationAngle;
        blueprint.transform.rotation =
Quaternion.Euler(rotationVector);
    }

    void AdjustPosition(float scrollInput)
    {
        float moveStep = 0.1f;
        if (moveAxis == 0)
            offsetX = Mathf.Clamp(offsetX +
scrollInput * moveStep, -moveLimit, moveLimit);
        else if (moveAxis == 2)
            offsetZ = Mathf.Clamp(offsetZ +
scrollInput * moveStep, -moveLimit, moveLimit);
        MoveBlueprint();
    }

    Vector3 GetGroundPosition(Vector3 startPos)
    {
        if (Physics.Raycast(startPos +
Vector3.up * 2f, Vector3.down, out RaycastHit
hit, 5f))
            return hit.point;
        return startPos;
    }

    void PlacePreviewObject()
    {
        if (blueprint != null)
        {

TutorialManager.instance.ApplyBuildMode = true;
            builded = true;
            pickUp.DropItem();
            objectToPlace.transform.position =
blueprint.transform.position;
            objectToPlace.transform.rotation =
blueprint.transform.rotation;

objectToPlace.GetComponent<Rigidbody>().isKinema
tic = true;

objectToPlace.GetComponent<Collider>().enabled =
true;
            Destroy(blueprint);
            objectToPlace = null;
            blueprint = null;
            GetComponent<MAnimal>().Stance =
DefStance;
            isPreviewing = false;
        }
    }

    void MakeTransparent(GameObject obj) { /*
... */ }
}
```

Б.4. SaveLoadManager.cs

Модуль збереження і завантаження стану світу. Серіалізує дані гравця і об'єктів у JSON, автоматично зберігає прогрес кожні п'ять секунд, відновлює стан при запуску гри.

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

public class SaveLoadManager : MonoBehaviour
{
    private string savePath;
    public static SaveLoadManager instance;

    [Header("All spawnable savable prefabs")]
    public List<GameObject> savablePrefabs;

    private Dictionary<string, GameObject>
prefabLookup;

    private void Awake()
    {
        if (instance == null) instance = this;

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

        prefabLookup = new Dictionary<string,
GameObject>();
        foreach (var prefab in savablePrefabs)
            prefabLookup[prefab.name] = prefab;
    }

    private void Start() => LoadGameData();

    void SaveGameInternal()
    {
        List<SavableObject> objects =
new(FindObjectsOfType<SavableObject>());
        SaveGame(Raccoon.Instance, objects);
        Debug.Log("Game Saved");
    }

    public void SaveGame(Raccoon player,
List<SavableObject> objects)
    {
        SaveData saveData = new SaveData();
        saveData.Player = new
PlayerData(player);
        foreach (var obj in objects)

saveData.Objects.Add(obj.GetSaveData());
```

```
        string tmpPath = savePath + ".tmp";
        File.WriteAllText(tmpPath,
JsonUtility.ToJson(saveData, true));
        if (File.Exists(savePath))
File.Delete(savePath);
        File.Move(tmpPath, savePath);
    }

    private void OnApplicationQuit() =>
SaveGameInternal();

    public void DeleteSaveFile()
    {
        if (File.Exists(savePath))
        {
            File.Delete(savePath);
            foreach (var item in
FindObjectsOfType<SavableObject>())
                Destroy(item.gameObject);
            Debug.Log("Save file deleted.");
        }
    }

    public SaveData LoadGame()
    {
        if (File.Exists(savePath))
        {
            string json =
File.ReadAllText(savePath);
            return
JsonUtility.FromJson<SaveData>(json);
        }
        return new SaveData();
    }

    public void LoadGameData()
    {
        SaveData loadedData = LoadGame();

        // Завантажити позицію гравця
        if (loadedData.Player != null &&
Raccoon.Instance != null)
        {
            Raccoon.Instance.transform.position
= loadedData.Player.Position;
            Raccoon.Instance.transform.rotation
= loadedData.Player.Rotation;
        }

        // Видалити об'єкти, яких немає у файлі
збереження
        HashSet<string> loadedIDs = new();
        foreach (var objData in
loadedData.Objects)
            loadedIDs.Add(objData.ObjectID);

        foreach (var obj in
FindObjectsOfType<SavableObject>())
            if
(!loadedIDs.Contains(obj.UniqueID))
                Destroy(obj.gameObject);

        // Відновити стан або створити нові
об'єкти
        Dictionary<string, SavableObject>
existingObjects = new();
        foreach (var obj in
FindObjectsOfType<SavableObject>())
            existingObjects[obj.UniqueID] = obj;

        foreach (var objData in
loadedData.Objects)
        {
            if
(existingObjects.TryGetValue(objData.ObjectID,
out var existing))
            {
                existing.LoadFromData(objData);
            }
            else if
(prefabLookup.TryGetValue(objData.PrefabName,
out var prefab))
            {
                GameObject newObj =
Instantiate(prefab, objData.Position,
objData.Rotation);

newObj.GetComponent<SavableObject>()?.LoadFromDa
ta(objData);
            }
        }

        StartCoroutine(startSavingWithDelay());
    }

    private IEnumerator startSavingWithDelay()
    {
        yield return new WaitForSeconds(2f);

InvokeRepeating(nameof(SaveGameInternal), 0f,
5f);
        ResourcesPlacer.instance.LoadedData =
true;
    }
}
```

Б.5. RaccoonFoodHolder.cs

Модуль логіки тримання предмета у передніх лапах єнота.

```
using MalbersAnimations;
using MalbersAnimations.Controller;
using System.Collections;
using UnityEngine;

public class RaccoonFoodHolder : MonoBehaviour
{
    [SerializeField] private MAnimal animal;
    [SerializeField] private MPickUp pickUp;
    [SerializeField] private ModeID eatMode;
    [SerializeField] private Transform
mouthPoint;
    [SerializeField] private float
biteIntervalSec = 1.2f;

    private GameObject heldItem;
    private FoodModelStates currentFood;

    private void Start()
    {
```

```
        pickUp.OnPicking.AddListener(OnPicked);

pickUp.OnDropping.AddListener(OnDropped);
    }

    private void OnPicked(GameObject item)
    {
        heldItem = item;
        currentFood =
item.GetComponent<FoodModelStates>();

        if (currentFood != null)
            TutorialManager.instance.PickedFood
= true;
    }

    private void OnDropped(int reason)
    {
        heldItem = null;
        currentFood = null;
    }

    private void Update()
    {
        if (heldItem == null) return;

        if (Input.GetMouseButtonDown(1) &&
currentFood != null)
        {
            animal.Mode_Activate(eatMode.ID);
            StartCoroutine(EatRoutine());
            TutorialManager.instance.Eat = true;
        }
    }

    private IEnumerator EatRoutine()
    {
        while (currentFood != null &&
currentFood.HasNextStage)
        {
            yield return new
WaitForSeconds(biteIntervalSec);
            BiteFood();
        }

        if (currentFood != null &&
!currentFood.HasNextStage)
        {

Raccoon.Instance.ChangeStat(StatID.Hunger,
+30f);
            Destroy(heldItem);
            heldItem = null;
            currentFood = null;
        }
    }

    private void BiteFood()
    {
        currentFood.NextStage();

Raccoon.Instance.ChangeStat(StatID.Hunger, +5f);
    }

    public bool HasItem => heldItem != null;

    public string HeldItemId => heldItem != null
? heldItem.name : string.Empty;
}
```

Б.6. IKController.cs

Модуль інверсної кінематики голови персонажа.

```
using UnityEngine;

public class IKController : MonoBehaviour
{
    [SerializeField] private Animator animator;
    [SerializeField] private Transform headBone;
    [SerializeField] private float lookSpeed =
5f;
    [SerializeField] private float maxAngle =
80f;

    public Transform LookTarget { get; set; }
    private Quaternion currentRotation;

    void OnAnimatorIK(int layerIndex)
    {
        if (animator == null) return;

        if (LookTarget != null)
        {
            animator.SetLookAtWeight(1f, 0.3f,
1f, 1f, 0.5f);

animator.SetLookAtPosition(LookTarget.position);
        }
        else
        {
            animator.SetLookAtWeight(0f);
        }
    }

    void LateUpdate()
    {
        if (LookTarget == null || headBone ==
null) return;

        Vector3 dirToTarget =
LookTarget.position - headBone.position;
        Quaternion targetRot =
Quaternion.LookRotation(dirToTarget,
Vector3.up);

        float angle =
Quaternion.Angle(headBone.rotation, targetRot);
        if (angle <= maxAngle)
        {
            currentRotation = Quaternion.Slerp(
                currentRotation, targetRot,
Time.deltaTime * lookSpeed);
            headBone.rotation = currentRotation;
        }
    }
}
```

Б.7. `MoveToClosestFood.cs` (приклад кастомної задачі Behavior Designer)

Приклад власної задачі для дерева поведінки.

```
using BehaviorDesigner.Runtime.Tasks;
using MalbersAnimations.Controller.AI;
using UnityEngine;

[TaskCategory("Malbers")]
public class MoveToClosestFood : MACBaseAction
{
    public LayerMask foodLayer;
    private MAnimalAIControl aiControl;

    public override void OnStart()
    {
        aiControl =
GetComponent<MAnimalAIControl>();
    }

    public override TaskStatus OnUpdate()
    {
        if (aiControl == null) return
TaskStatus.Failure;

        GameObject[] foodSources =
GameObject.FindGameObjectsWithTag("Food");
        GameObject nearestFoodSource = null;
        float closestDistance = Mathf.Infinity;

        foreach (var foodSource in foodSources)
        {
            if (((1 << foodSource.layer) &
foodLayer) != 0)
            {
                float distance =
Vector3.Distance(

aiControl.transform.position,

foodSource.transform.position);
                if (distance < closestDistance)
                {
                    closestDistance = distance;
                    nearestFoodSource =
foodSource;
                }
            }
        }

        if (nearestFoodSource == null) return
TaskStatus.Failure;

        if (Vector3.Distance(transform.position,
nearestFoodSource.transform.position) < 1.5f)
            return TaskStatus.Success;


aiControl.SetTarget(nearestFoodSource.transform.
gameObject);
        return TaskStatus.Running;
    }
}
```

ДОДАТОК В. ІНСТРУКЦІЯ З УСТАНОВЛЕННЯ ТА ЗАПУСКУ

Для встановлення програмного забезпечення необхідно виконати такі дії:

1) завантажити архів з білдом гри TwilightTails_Build.zip з репозиторію проєкту;

2) розпакувати архів у будь-яку папку на диску; у повному шляху до папки не повинно бути символів кирилиці;

3) запустити виконуваний файл TwilightTails.exe;

4) у головному меню обрати «Single Game»; якщо у каталозі `%APPDATA%\..\LocalLow\TwilightTails\save.json` присутній файл збереження, гра продовжить попередню сесію; якщо файл відсутній - розпочнеться нова гра.

Мінімальні системні вимоги для запуску гри:

- операційна система Windows 10 (64-розрядна) або новіша;

- центральний процесор Intel Core i5 шостого покоління або еквівалентний AMD Ryzen;

- оперативна пам'ять - не менше 8 ГБ;

- графічний адаптер з обсягом відеопам'яті не менше 4 ГБ та підтримкою DirectX 12;

- вільне місце на накопичувачі - близько 15 ГБ.

При першому запуску гра пропонує пройти інтерактивний туторіал, що знайомить з основними механіками: керуванням, pickup-взаємодією, харчуванням, режимом будівництва. Туторіал можна пропустити з меню налаштувань.