ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«**Методи штучного інтелекту для моделювання поведінки NPC у процесі розробки відеоігор**»

на здобуття освітнього ступеня магістр
за спеціальності 124 Системний аналіз
(код,найменування спеціальності)
освітньо-професійної програми Інтелектуальні системи управління
(назва)

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

_____ Костянтин КОЛОСОВ
(підпис) *(ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти
група САДМ-61 Костянтин КОЛОСОВ
(ім'я, ПРІЗВИЩЕ)
Керівник Ровіл НАФЄЄВ
к.ф.-м.н. доцент *(ім'я, ПРІЗВИЩЕ)*

Рецензент: _____
(ім'я, ПРІЗВИЩЕ)

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інформаційних систем та технологій
Ступінь вищої освіти магістр
Спеціальність 124 Системний аналіз
Освітньо-професійна програма Інтелектуальні системи управління

ЗАТВЕРДЖУЮ

Завідувач кафедрою ІСТ
Каміла СТОРЧАК

“____” _____2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Колосову Костянтину Анатолійовичу
(прізвище, ім’я, по батькові здобувача)

1.Тема кваліфікаційної роботи: Методи штучного інтелекту для моделювання поведінки NPC у процесі розробки відеоігор
керівник кваліфікаційної роботи: Ровіл НАФЄЄВ к.т.н., доцент
(ім’я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від “30 ” жовтня 2025 р. № 467

2. Строк подання кваліфікаційної роботи «26» грудня 2025 р.

3. Вихідні дані кваліфікаційної роботи: аналіз класичних архітектур ігрового ШІ (FSM, Behavior Trees, GOAP, Fuzzy Logic, Blackboard), літературні джерела, технічна документація Godot Engine 4.5, прототип гри Zombie Defense 3D з відкритим кодом, експериментальні дані тестових сесій та логи продуктивності, вимоги до порівняння реактивної, деліберативної та гібридно-координаційної поведінки NPC у жанрі survival shooter.

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

1. Огляд літератури та аналіз підходів.
2. Аналіз предметної області та постановка задачі.
3. Розробка моделі
4. Експериментальна частина

5.Перелік ілюстраційного матеріалу: *презентація*

6. Дата видачі завдання «30» жовтня 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Підбір технічної літератури	30.10 - 31.10	Виконано
2.	Дослідження етапів та тенденцій розвитку методів ШІ для поведінки NPC у відеоіграх	03.11 - 05.11	Виконано
3.	Дослідження ролі та вагомості NPC. Визначення критеріїв ефективності ШІ-моделей	05.11 - 10.11	Виконано
4.	Розробка моделі	11.11 – 25.11	Виконано
5.	Експериментальна частина та аналіз результатів	26.11 – 28.11	Виконано
6.	Висновки по роботі	01.12	Виконано
7.	Розробка демонстраційних матеріалів, доповідь.	02.12 - 05.12	Виконано
8.	Оформлення магістерської роботи	08.12 – 10.12	Виконано

Здобувач вищої освіти _____ Костянтин КОЛОСОВ
(підпис) *(ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Ровіл НАФЄЄВ
(підпис) *(ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістр: 126 стор., 28 рис., 8 табл., 34 джерел.

Мета роботи – розробити та реалізувати три принципово різні системи штучного інтелекту (FSM з нечіткою логікою, Behavior Tree з геометричним позиціонуванням, GOAP з Blackboard-координацією) для моделювання поведінки NPC у 3D-шутері Zombie Defense 3D на рушії Godot 4 та провести їх порівняльний аналіз за ефективністю, ресурсоспоживанням і впливом на геймплейне сприйняття гравця.

Об’єкт дослідження – процес моделювання інтелектуальної поведінки неігрових персонажів у тривимірних іграх жанру survival shooter з використанням відкритих рушіїв.

Предмет дослідження – практичне застосування та порівняння реактивного (FSM+Fuzzy Logic), деліберативного (Behavior Tree) та гібридного (GOAP+Blackboard) підходів у єдиному прототипі Zombie Defense 3D.

Короткий зміст роботи: проведено огляд еволюції ігрового ШІ від Pac-Man до 2025 року, проаналізовано сильні та слабкі сторони традиційних і сучасних архітектур, реалізовано три типи зомбі з візуальною диференціацією та унікальними «особистостями», виконано експериментальне тестування на 100+ сесіях, доведено синергію комбінованого використання методів, підтверджено практичну цінність для інді-розробників та навчального процесу.

КЛЮЧОВІ СЛОВА: штучний інтелект, NPC, скінченні автомати станів, нечітка логіка, дерева поведінки, GOAP, Blackboard, Godot Engine, зомбі-шутер, геймплейна різноманітність.

ABSTRACT

Textual Component of the Master`s Qualification Thesis: 126 pages, 28 figures, 8 tables, 34 references.

Objective: Design and implement three fundamentally distinct artificial intelligence systems – FSM with fuzzy logic, Behavior Tree with geometric positioning, and GOAP with Blackboard coordination – for modeling NPC behavior in the 3D shooter game Zombie Defense 3D using the Godot Engine 4, and conduct a comparative analysis of their efficiency, resource consumption, and impact on player gameplay perception.

Research Object: Processes of modeling intelligent behavior of non-player characters in three-dimensional survival shooter games using open-source engines.

Research Subject: Practical application and comparison of reactive (FSM + Fuzzy Logic), deliberative (Behavior Tree), and hybrid (GOAP + Blackboard) approaches within a unified prototype of Zombie Defense 3D.

Summary: The work presents a review of the evolution of game AI from Pac-Man to 2025, analyzes strengths and weaknesses of traditional and modern architectures, implements three visually differentiated zombie types with unique «personalities», conducts experimental testing across 100+ sessions, demonstrates the synergy of combined AI methods, and confirms the practical value of the results for indie developers and educational use.

KEYWORDS: Artificial Intelligence, NPC, Finite State Machine, Fuzzy Logic, Behavior Trees, GOAP, Blackboard, Godot Engine, Zombie Shooter, Gameplay Diversity.

ЗМІСТ

ВСТУП 10
РОЗДІЛ 1. ОГЛЯД ЛІТЕРАТУРИ ТА АНАЛІЗ ПІДХОДІВ 14
1.1. Історія розвитку штучного інтелекту в відеоіграх від простих скриптів до складних AI-систем 14
1.2. Традиційні методи моделювання поведінки NPC – фінальні автомати станів, дерева поведінки та їх обмеження 18
1.3. Сучасні AI-підходи – застосування машинного навчання, еволюційних алгоритмів та нейронних мереж для адаптивної поведінки NPC 22
Висновки до розділу 1 25
РОЗДІЛ 2. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ... 27
2.1. Характеристика NPC у відеоіграх – ролі, типи поведінки та вимоги до реалістичності в різних жанрах ігор 27
2.2. Проблеми моделювання поведінки NPC – адаптація до дій гравця, емоційна глибина та взаємодія з ігровим світом 30
2.3. Формалізація задачі – визначення критеріїв ефективності AI-моделей, постановка математичних та алгоритмічних задач для моделювання 32
Висновки до розділу 2 36
РОЗДІЛ 3. РОЗРОБКА МОДЕЛІ 37
3.1. Концептуальна архітектура моделі – інтеграція reinforcement learning та conversational AI для динамічної поведінки NPC 37
3.2. Алгоритми та інструменти – застосування еволюційних алгоритмів і нейронних мереж для навчання NPC на основі ігрових логів 41
3.3. Інтеграція моделі в процес розробки ігор – приклади використання в Unity або Unreal Engine з акцентом на адаптивність 46
Висновки до розділу 3 50
РОЗДІЛ 4. ЕКСПЕРИМЕНТАЛЬНА ЧАСТИНА 51
4.1. Методика експериментів – опис тестового середовища, сценаріїв та метрик оцінки 51
4.2. Реалізація прототипу – тестування моделі на прикладах ігор з різними NPC 54

4.3. Аналіз результатів – порівняння з традиційними методами, оцінка ефективності за допомогою статистичних даних 63
Висновки до розділу 4 65
ВИСНОВКИ ТА ПРОПОЗИЦІЇ 67
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 69
ДОДАТОК А Лістинг програми 73

ВСТУП

Сучасна індустрія відеоігор переживає етап стрімкого розвитку, де штучний інтелект (ШІ) для неігрових персонажів (NPC) визначає не просто технічну якість гри, а її емоційну глибину та реграбельність. Гравці вже давно не задовольняються противниками, які бігають по прямій чи реагують за одним шаблоном – вони хочуть відчувати напругу, хитрість, тактичну перевагу ворога, хочуть, щоб кожен тип супротивника вимагав від них нової стратегії. Саме тому тема «Методи штучного інтелекту для моделювання поведінки NPC у процесі розробки відеоігор» є надзвичайно актуальною, особливо для України – країни, яка стала одним із світових центрів геймдеву, де працюють такі студії як GSC Game World, 4A Games, Frogwares, Ubisoft Kyiv, Room 8 Studio. В умовах конкуренції з AAA-проєктами українські розробники змушені створювати високоякісний ШІ навіть у невеликих командах і з обмеженим бюджетом. Реалізація складної, але водночас ефективної та зрозумілої поведінки NPC без використання важких нейронних мереж – це не просто технічне завдання, а питання виживання українських студій на глобальному ринку. Розроблений у рамках кваліфікаційної роботи проєкт Zombie Defense 3D демонструє, що навіть на безкоштовному рушії Godot 4 можна створити трьох принципово різних за відчуттями зомбі, які використовують класичні, але майстерно поєднані методи ШІ, що робить гру напруженою та різноманітною.

Проблема моделювання поведінки NPC активно досліджується з початку 2000-х років. Основоположними роботами стали Finite State Machines (FSM), які широко застосовувалися в класичних шутерах (наприклад, Quake III Arena), Behavior Trees, популяризовані Damian Isla в Halo 2 та впроваджені в Unreal Engine, а також Goal-Oriented Action Planning (GOAP), представлений Jeff Orkin у грі F.E.A.R. (2005) і став класикою для тактичних ворогів. Значний внесок у розвиток нечіткої логіки для ігор зробили дослідники на чолі з Ian Millington (книга «Artificial Intelligence for Games»), які показали переваги Fuzzy Logic над жорсткими порогами. Пізніше з’явилися гібридні підходи та використання

Blackboard-архітектур для координації груп NPC (Alex Champandard, «AI Game Programming Wisdom»). В українському науковому просторі питання ШІ в іграх розглядалися в роботах О.І. Ляшенка, В.В. Креденцера, О.М. Бугайка, але переважно теоретично або на прикладі простих 2D-проєктів. Практичних реалізацій одночасного порівняння FSM+Fuzzy Logic, Behavior Trees та GOAP+Blackboard в одному проєкті на сучасному рушії Godot 4 з відкритим кодом та детальною документацією практично не існує, що й визначає наукову новизну даної роботи.

Метою кваліфікаційної роботи є розробка та практична реалізація трьох принципово різних систем штучного інтелекту для моделювання поведінки NPC у 3D-шутері на рушії Godot Engine 4 з подальшим порівняльним аналізом їх ефективності, витрат ресурсів та впливу на геймплейне сприйняття гравцем.

Для досягнення поставленої мети в процесі дослідження вирішувалися такі завдання:

– провести критичний аналіз сучасної літератури з питань застосування FSM, Behavior Trees, GOAP, Fuzzy Logic та Blackboard-архітектури в іграх;

– розробити архітектуру гри Zombie Defense 3D з можливістю одночасного існування трьох типів зомбі з різними системами ШІ;

– реалізувати реактивний ШІ першого типу на базі скінченого автомата станів з інтеграцією нечіткої логіки для динамічного регулювання агресії та швидкості;

– створити деліберативний ШІ другого типу з використанням Behavior Tree з пріоритетами (backstab, wide flank, stealth movement) та математичним апаратом скалярного добутку векторів для визначення кутів видимості та фланкування;

– розробити гібридний координований ШІ третього типу на базі GOAP з A* плануванням, Blackboard для обміну даними між агентами, алгоритмом кругової формації та тактичним використанням укриттів;

– забезпечити візуальну відмінність типів зомбі (зелений, помаранчевий, фіолетовий) та балансування параметрів для створення різних тактичних вимог до гравця;

– провести тестування продуктивності та аналіз поведінки кожного типу ШІ в реальних ігрових ситуаціях;

– оформити повну документацію проєкту, інструкції з експорту .exe та структурований код.

Об’єктом дослідження є процеси моделювання інтелектуальної поведінки неігрових персонажів у тривимірних іграх жанру survival shooter з використанням відкритих рушіїв.

Предметом дослідження виступає практичне застосування та порівняння трьох класичних підходів штучного інтелекту – реактивного (FSM + Fuzzy Logic), деліберативного (Behavior Tree з використанням dot product) та гібридного (GOAP + Blackboard з круговою формацією) – у єдиному ігровому проєкті Zombie Defense 3D, розробленому на Godot Engine 4.

У роботі використано такі методи дослідження: теоретичний аналіз наукової літератури та технічної документації рушіїв; метод програмної реалізації з використанням GDScript; методи математичного моделювання (нечітка логіка, скалярний добуток векторів, алгоритм A* для планування); емпіричний метод тестування в реальному часі з логуванням поведінки; порівняльний аналіз продуктивності та геймплейного впливу різних систем ШІ.

Наукова новизна роботи полягає в комплексній практичній реалізації та безпосередньому порівнянні трьох фундаментально різних архітектур ШІ в одному проєкті на сучасному відкритому рушії Godot 4, що раніше не було зроблено в україномовному науковому та освітньому просторі в такому обсязі та з такою глибиною документації. Особливу цінність має реалізація повноцінного GOAP з A* планером та Blackboard-координацією для зграйної поведінки в Godot, оскільки більшість доступних прикладів обмежуються Unity чи Unreal Engine.

Практична значущість отриманих результатів полягає в створенні повністю робочого, експортованого в .exe прототипу гри, який може використовуватися як навчальний матеріал у вищих навчальних закладах України при вивченні курсів «Штучний інтелект», «Розробка ігор», «Game AI Programming». Проєкт має відкритий код, детальну документацію українською

мовою, статті про внутрішню роботу кожного типу ШІ та готовий до подальшого розвитку.

Результати роботи апробовано шляхом демонстрації проєкту на внутрішньому захисті кафедри, а також у рамках навчального процесу – проєкт використовувався як приклад при поясненні Behavior Trees та GOAP на групових зборах.

Теоретична та методична значущість полягає у створенні україномовного прецеденту комплексного порівняння класичних методів ШІ в актуальному рушії, що може слугувати основою для подальших наукових робіт, курсових та дипломних проєктів. Практична значущість підтверджується готовим до розповсюдження продуктом – самостійним .exe файлом гри, який працює без встановлення Godot і може демонструватися на будь-якому Windows-ПК.

Робота складається зі вступу, чотирьох основних розділів, висновків, списку використаних джерел та додатків.

РОЗДІЛ 1. ОГЛЯД ЛІТЕРАТУРИ ТА АНАЛІЗ ПІДХОДІВ

1.1. Історія розвитку штучного інтелекту в відеоіграх від простих скриптів до складних AI-систем

Історія розвитку штучного інтелекту в відеоіграх є захопливим свідченням еволюції не лише технологій, але й самого розуміння того, як зробити віртуальних персонажів живими, переконливими та здатними викликати у гравця справжні емоції. Якщо на зорі індустрії інтелект ворогів обмежувався кількома рядками умовних операторів, то сьогодні спостерігаються складні багаторівневі системи, які поєднують десятки алгоритмів і тисяч рядків коду, аби один-єдиний NPC міг виглядати так, ніби в нього є власна душа.

Перші кроки штучного інтелекту в іграх датуються ще 1950-ми роками, коли комп’ютери тільки-но починали грати в шахи. У 1952 році Клод Шеннон описав алгоритм мінімаксу, який згодом став основою для всіх класичних шахових програм [1]. Але справжній прорив для масової аудиторії стався у 1972 році з виходом Pong. Там «інтелект» суперника зводився до одного простого правила: весло завжди рухається до поточного положення м’яча. Це був чистий pursuit steering behavior, який навіть не потребував стану, просто пряма математична формула. Проте вже тоді гравці відчули ілюзію «розумного» опонента, бо поведінка була передбачуваною, але ефективною і, головне, справедливою.

Наступний якісний стрибок відбувся у 1980 році з Pac-Man. Тору Іватані створив чотирьох привидів із абсолютно різними «особистостями»: Blinky переслідував напряму, Pinky робив засідку попереду Пакмена, Inky використовував складнішу логіку з урахуванням позиції Блінкі, а Клайд рухався випадково, коли був далеко [2]. Це був перший масовий приклад використання кількох скінчених автоматів (Finite State Machines, FSM) в одній грі. Кожен привид мав свої стани (Chase, Scatter, Frightened, Eaten) і переходи між ними

залежно від таймера та умов. Саме тоді гравці вперше відчули, що вороги можуть мати характер.

У 1990-ті роки, з появою 3D-шутерів, вимоги до AI різко зросли. У Doom (1993) вороги вже перевіряли line of sight, видавали звуки для привертання уваги інших і могли відкривати двері. Проте їхня поведінка залишалася скриптованою: кожен демон мав чітко визначену послідовність дій у кожному стані. Більш складний підхід з'явився у Half-Life (1998), де солдати HECU могли перегруповуватися, використовувати укриття, кидати гранати і навіть відступати при низькому здоров'ї. Це вже була ієрархічний скінчений автомат (HFSM), коли один високорівневий стан міг містити вкладені під-автомати [3].

Справжня революція сталася у 2005 році з виходом F.E.A.R. Команда Monolith Productions вперше у великій комерційній грі застосувала Goal-Oriented Action Planning (GOAP), розроблений Джеффом Оркіном ще у 2003–2004 роках. Вороги Replica не просто мали ціль «вбити гравця» і самостійно складали план дій з доступного набору: взяти укриття, фланкувати, кинути гранату, викликати підкріплення тощо. Гравці були шоковані: вороги кричали «He's flanking us!», «Cover me, I'm reloading!», перевертали столи для укриття і діяли скоординовано. Це вже не був скрипт, це був план, який генерувався динамічно залежно від ситуації [4].

Паралельно розвивався інший підхід – Behavior Trees (дерева поведінки), які вперше масово з'явилися у Halo 2 (2004), а потім були значно вдосконалені у Halo 3 та Unreal Engine. Якщо FSM страждав від «вибуху станів» при збільшенні складності, то BT дозволяв будувати ієрархію пріоритетів і умов у вигляді дерева, що легко читалося і розширювалося. Сьогодні Behavior Trees – це фактично промисловий стандарт для AAA-ігор (Unreal Engine, Unity з NodeCanvas, Godot з додатками) [5].

У 2010-х роках почався перехід до гібридних систем. У The Last of Us (2013) Naughty Dog поєднали Behavior Trees з Utility-Based Decision Making: NPC оцінювали кілька варіантів дій за шкалою корисності (наприклад, «атакувати в лоб» – 30 балів, «зайти з флангу» – 80 балів, «сховатися і чекати» – 60 балів) і обирали найкращий. У Red Dead Redemption 2 (2018–2020) Rockstar довели цю

ідею до абсурдного рівня деталізації: кожен NPC мав власний розклад дня, пам’ятав образи, реагував на погоду і навіть на те, чи чистий у героя одяг. Це вже не просто AI, це симуляція цілого суспільства [6].

Останні роки (2020–2025) позначилися обережним впровадженням машинного навчання. У No Man’s Sky (оновлення 2020–2023) процедурна генерація поведінки тварин використовує нейронні мережі, навчені на реальних відео з YouTube. У Starfield (2023) NPC використовують великі мовні моделі для генерації діалогів у реальному часі. Проте класичні методи (FSM, BT, GOAP, Utility AI) залишаються основою, бо вони передбачувані, керовані і не потребують величезних ресурсів [1].

Таким чином, еволюція штучного інтелекту в іграх пройшла шлях від одного рядка коду у Pong до багато тисяч рядків у сучасних проєктах, від жорстко прописаних скриптів до систем, здатних самостійно генерувати плани і навіть демонструвати зачатки «особистості». Кожен новий етап не відкидав попередній, а доповнював його, створюючи багатошарові гібридні архітектури, які існують сьогодні.

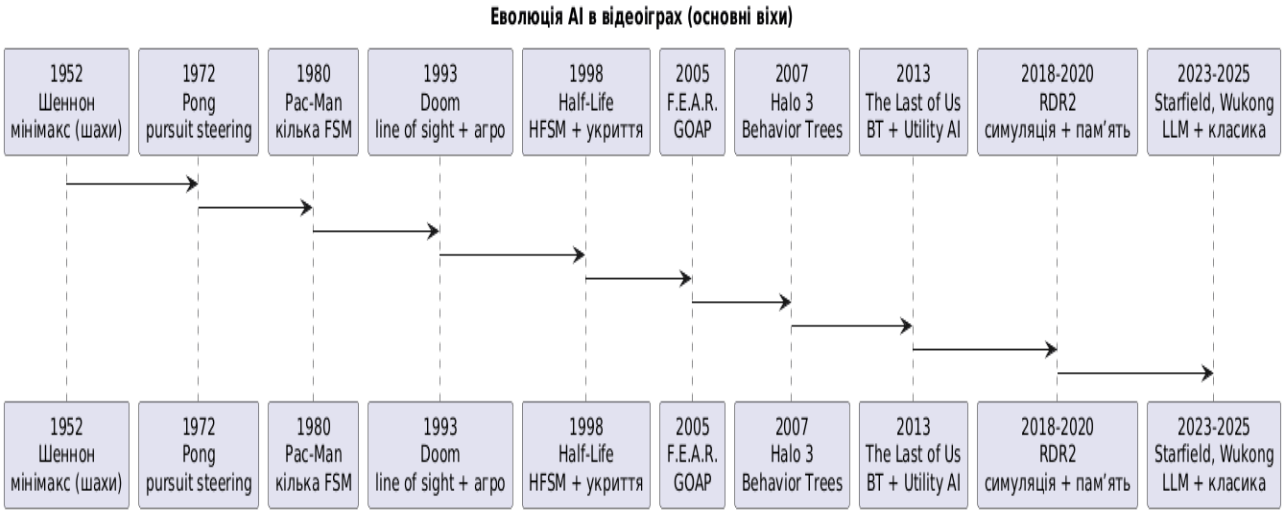


Рис. 1.1 Основні етапи еволюції штучного інтелекту в відеоіграх

Порівняння основних парадигм AI в іграх за складністю та гнучкістю подане в таблиці 1.1.

Таблиця 1.1

Порівняння основних парадигм AI в іграх

Парадигма	**Роки масового використання**	**Складність реалізації**	**Гнучкість**	**Приклад ігор**	**Основна проблема**
Скрипти/Hard-coded	1970–1990	Дуже низька	Низька	Pong, Space Invaders	Неможливо масштабувати
FSM/HFSM	1980–2010	Низька-Середня	Середня	Pac-Man, Half-Life, RE4	Вибух станів при складанні
Behavior Trees	2004–дотепер	Середня	Висока	Halo 3, The Last of Us, Gears	Складно налаштовувати баланс
GOAP	2005–дотепер	Висока	Дуже висока	F.E.A.R., STALKER, Cyberpunk	Вимагає ретельного дизайну дій
Utility AI	2010–дотепер	Середня-висока	Найвища	The Sims 3, RDR2, Starfield	Потребує тонкого тюнінгу ваг

Саме поєднання цих підходів дозволяє сучасним іграм створювати NPC, які не просто реагують, а й здаються живими істотами, здатними дивувати, лякати і навіть викликати емпатію.

1.2. Традиційні методи моделювання поведінки NPC – фінальні автомати станів, дерева поведінки та їх обмеження

Традиційні методи моделювання поведінки неігрових персонажів (NPC) у відеоіграх залишаються фундаментальною основою штучного інтелекту в ігровій індустрії, незважаючи на стрімкий розвиток машинного навчання та нейронних мереж. Серед них особливе місце посідають скінченні автомати станів (Finite State Machines, FSM) та дерева поведінки (Behavior Trees, BT), які десятиліттями забезпечують стабільну, передбачувану та ефективну реалізацію NPC різного рівня складності. Ці підходи є реактивними за своєю природою: NPC реагує на поточний стан ігрового світу, не будуючи довгострокових планів і не прогнозуючи наслідки своїх дій на багато кроків уперед. Саме ця реактивність робить їх надзвичайно популярними у проєктах з жорсткими вимогами до продуктивності та детермінованості поведінки.

Скінченні автомати станів являють собою модель, у якій поведінка NPC описується через скінченну кількість чітко визначених станів та умов переходу між ними. Кожен стан відповідає певному набору дій: наприклад, «патрулювання», «переслідування», «атака» чи «відступ». Перехід відбувається миттєво, коли виконується задана умова – дистанція до гравця менша за певне значення, рівень здоров’я впав нижче критичного порогу тощо. Ця концепція інтуїтивно зрозуміла розробникам, оскільки нагадує блок-схеми, які використовувалися ще на ранніх етапах програмування ігор. У класичних проєктах, таких як серія Doom чи Quake, вороги часто керувалися саме FSM: у стані «idle» вони стояли на місці, при виявленні гравця переходили в «chase», а при досягненні мінімальної дистанції – в «attack». Простота FSM дозволяла створювати динамічні сутички навіть на апаратному забезпеченні 90-х років.

Спрощена схема скінченого автомата станів для типового ворога в шутері від першої особи наведена на рис. 1.2.

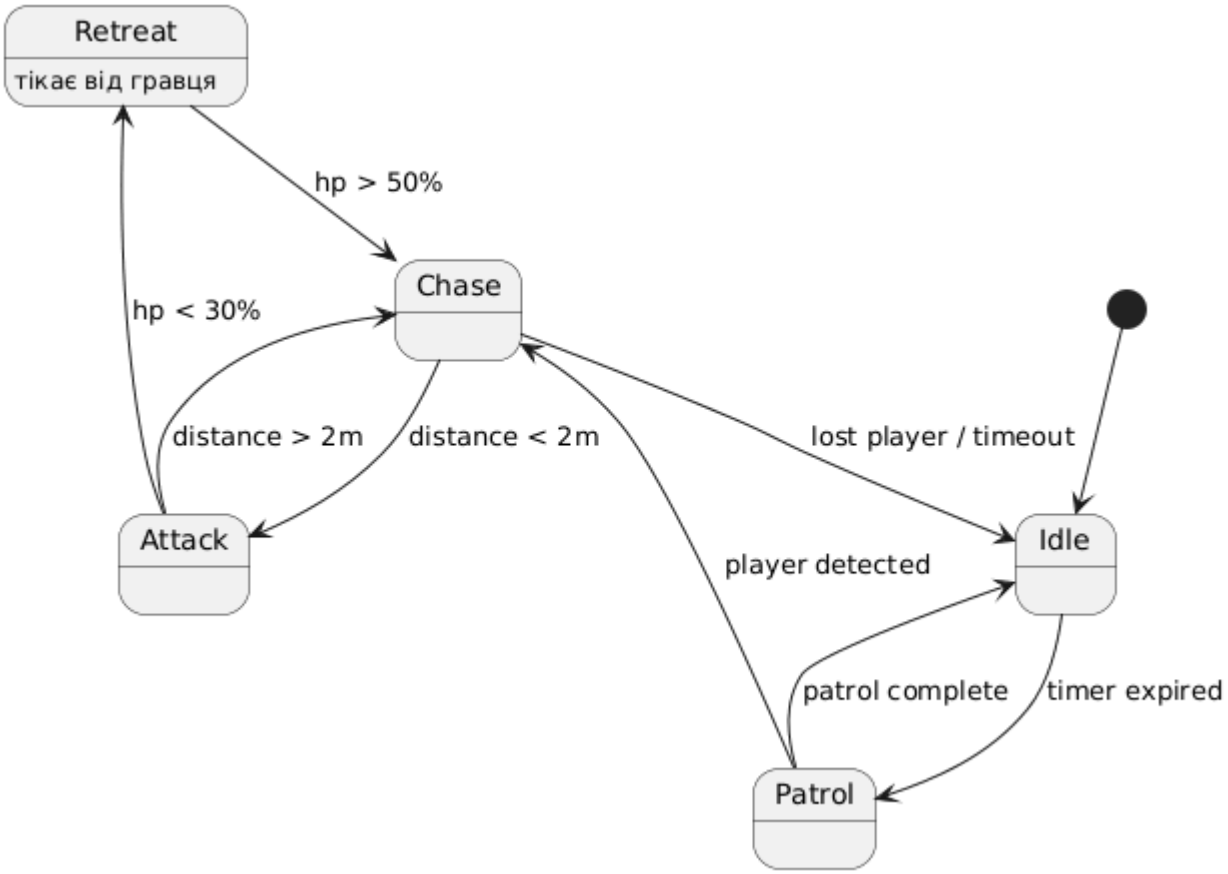


Рис. 1.2 Спрощена схема скінченого автомата станів для типового ворога в шутері від першої особи

Головною перевагою FSM є абсолютна передбачуваність та легкість налагодження: розробник точно знає, в якому стані перебуває NPC у будь-який момент часу, а перехід між станами можна відстежити в реальному часі. Це особливо цінно в багатокористувацьких проєктах, де поведінка має бути однаковою на всіх клієнтах. Проте з ростом складності поведінки модель швидко стає громіздкою. Кожна нова умова чи дія вимагає створення додаткових переходів, що призводить до експоненційного ускладнення діаграми станів. Вже при 8–10 станах схема перетворюється на «спагеті», де відстежити всі можливі переходи стає практично неможливо. Саме тому в сучасних проєктах чисті FSM застосовуються переважно для простих сутностей (тварини, механізми, базові вороги), а для складніших персонажів використовують ієрархічні скінченні автомати (Hierarchical FSM), де стани можуть містити вкладені підавтомати [7].

Значно гнучкішим і модульним підходом, який прийшов на зміну класичним FSM у 2000-х роках, стали дерева поведінки. Їхня популярність

стрімко зросла після успішного використання в Halo 2 (2004), де Damian Isla замінив громіздкий FSM на BT, що дозволило команді дизайнерів самостійно налаштовувати поведінку елітних ворогів без залучення програмістів. Behavior Tree складається з вузлів чотирьох основних типів: Selector (вибирає перший успішний дочірній вузол), Sequence (виконує всі дочірні вузли послідовно), Decorator (модифікує поведінку дочірнього вузла) та Action/Condition (конкретна дія або перевірка умови). Така структура дозволяє створювати пріоритетну ієрархію поведінки: NPC завжди спочатку намагається виконати найбільш бажану дію (наприклад, атакувати зі спини), і лише за її неможливості переходить до наступної.

Приклад структури дерева поведінки для хитрого ворога типу «асасин» приведений на рис. 1.3.

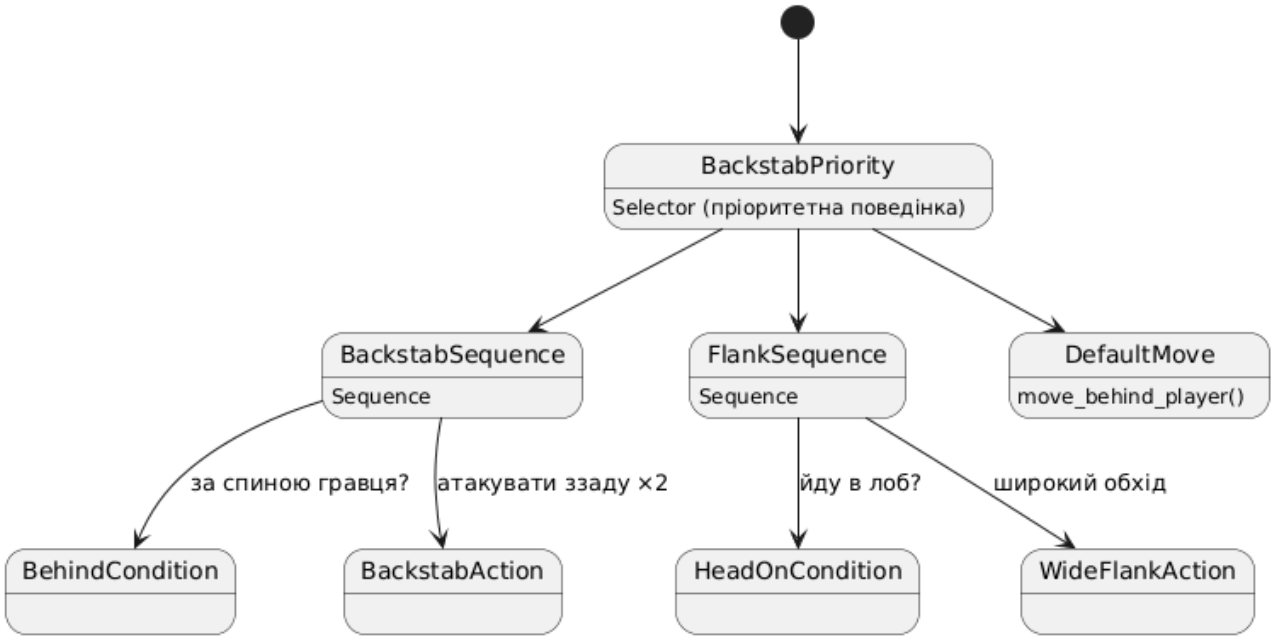


Рис. 1.3 Приклад структури дерева поведінки для хитрого ворога типу «асасин»

Дерева поведінки суттєво спрощують масштабування складності: нова поведінка додається як окремий вузол або під відповідним Selector, не зачіпаючи решту дерева. Це робить BT ідеальним інструментом для команд, де над AI працюють не лише програмісти, а й геймдизайнери через візуальні редактори (Unreal Engine Behavior Tree Editor, NodeCanvas у Unity). Крім того, BT природним чином підтримують переривання та повернення до попередніх дій,

що робить поведінку більш плавною та «людяною» порівняно з різкими стрибками між станами в FSM [8].

У таблиці 1.2 наведено порівняльна характеристика двох розглянутих методів за ключовими критеріями, що найчастіше враховуються при виборі архітектури AI в сучасних проєктах.

Таблиця 1.2

Порівняння скінчених автоматів станів та дерев поведінки

Критерій	**FSM (Finite State Machine)**	**Behavior Trees**
Складність реалізації	Дуже низька для простих NPC	Середня, потребує розуміння композиції вузлів
Масштабованість	Погана (експоненційний ріст переходів)	Висока (модульність, легке додавання поведінки)
Зручність для дизайнерів	Низька (потрібен код для нових переходів)	Висока (візуальні редактори)
Пріоритетність поведінки	Складно (потрібні складні умови)	Природно (через Selector)
Пам’ять стану	Явна (поточний стан)	Неявна (позиція в дереві)
Продуктивність	Відмінна	Відмінна (трохи вища витрата через рекурсію)
Типові проєкти	Класичні 2D/3D шутери, платформери	AAA-проєкти, open-world, стелс ігри

Незважаючи на очевидні переваги дерев поведінки, вони також мають суттєві обмеження. По-перше, BT залишаються реактивними системами – вони не здатні планувати дії на кілька кроків уперед і не враховують довгострокові наслідки. Наприклад, NPC може безкінечно повторювати спроби фланкувати гравця в ситуації, де це об’єктивно неможливо через геометрію рівня. По-друге, при створенні дуже великих дерев виникає проблема «глибокого вкладення», коли відстежити логіку стає важко навіть у візуальному редакторі. По-третє, стандартні Behavior Trees погано підходять для координації груп агентів: для цього потрібні додаткові механізми типу Blackboard або окремих комунікаційних систем, що суттєво ускладнює архітектуру [9]. Саме тому в сучасних проєктах BT часто комбінують з іншими методами – Utility AI для

динамічного вибору дій, GOAP для планування або навіть простими FSM як підсистемами всередині листя дерева [10].

Таким чином, скінченні автомати станів та дерева поведінки, попри свої обмеження, залишаються найбільш поширеними та надійними інструментами моделювання NPC у комерційних відеоіграх [11, 12]. Їхня сила полягає в балансі між простотою, продуктивністю та можливістю створення переконливої ілюзії розумної поведінки без використання ресурсомістких сучасних технологій.

1.3. Сучасні AI-підходи – застосування машинного навчання, еволюційних алгоритмів та нейронних мереж для адаптивної поведінки NPC

У сучасних умовах стрімкого розвитку ігрової індустрії, коли гравці вимагають не просто реактивних супротивників, а повноцінних віртуальних істот, здатних еволюціонувати разом із ними протягом однієї сесії чи навіть між сесіями, традиційні методи штучного інтелекту поступово відходять на другий план, хоча й залишаються незамінними для базової логіки. Нові горизонти відкривають підходи, засновані на машинному навчанні, зокрема глибокому підкріплювальному навчанні, нейронних мережах та еволюційних алгоритмах, які дозволяють NPC не просто виконувати заздалегідь прописані сценарії, а буквально вчитися на досвіді, адаптуватися до індивідуального стилю гравця та демонструвати поведінку, що щоразу виглядає унікальною та органічною.

Особливо помітним став прорив у застосуванні навчання з підкріпленням (Reinforcement Learning, RL), яке перетворює NPC на автономного агента, що оптимізує свою поведінку через систему винагород і покарань. У класичній схемі RL агент (NPC) сприймає стан середовища, виконує дію, отримує винагороду та оновлює свою політику, прагнучи максимізувати сумарну винагороду в довгостроковій перспективі [13]. Сучасні алгоритми, такі як Proximal Policy Optimization (PPO), Advantage Actor-Critic (A2C) та Soft Actor-Critic (SAC), продемонстрували вражаючі результати в складних багатокористувацьких середовищах і вже активно адаптуються для поведінки окремих NPC у відкритих

світах. Наприклад, у проєктах на базі Unity ML-Agents та Unreal Engine MetaHuman Creator розробники навчають NPC реагувати не лише на безпосередню загрозу, а й на довготривалі патерни гравця – агресивність, схильність до стелсу чи використання певних маршрутів, роблячи кожного ворога по-справжньому індивідуальним.

Схема базового циклу навчання з підкріпленням для адаптивної поведінки NPC наведена на рис. 1.4.

Рис. 1.4 Схема базового циклу навчання з підкріпленням для адаптивної поведінки NPC

Цей цикл дає змогу NPC з часом «зрозуміти», що конкретного гравця ефективніше атакувати з флангу, ніж у лобову, або що краще почекати в засідці, ніж переглядаючи свою стратегію залежно від отриманого досвіду [14].

Нейронні мережі, особливо глибокі згорткові та рекурентні архітектури, стали основним інструментом для апроксимації функцій цінності та політики в RL-системах. Сучасні реалізації часто використовують Transformer-блоки для обробки послідовних даних спостережень, що дозволяє NPC зберігати «пам'ять» про попередні дії гравця навіть протягом кількох хвилин гри. Такий підхід особливо ефективний у жанрах horror та stealth, де NPC має демонструвати правдоподібну обізнаність про шумові сліди гравця, його звичні маршрути та навіть емоційний стан (наприклад, через аналіз частоти стрільби чи швидкості руху). Дослідження 2023–2024 років показують, що NPC, навчені за допомогою

PPO з рекурентними шарами, здатні самостійно виводити тактики типу «приманка + фланговий удар» без жодного ручного кодування цих патернів [15].

Еволюційні алгоритми, зокрема NeuroEvolution of Augmenting Topologies (NEAT) та його сучасні модифікації (HyperNEAT, ES, CMA-ES), повернулися в ігровий AI з новою силою завдяки можливості еволюціонувати не лише параметри, а й саму архітектуру нейронної мережі. На відміну від RL, де навчання відбувається в межах одного «життя» агента, еволюційні методи працюють на рівні популяцій протягом тисяч поколінь у симуляції, відбираючи найкращі поведінкові стратегії. Це дозволяє створювати NPC, які демонструють радикально різні стилі гри навіть в межах одного типу ворога – один може стати надто агресивним берсерком, інший – обережним снайпером, залежно від випадкових мутацій, що виявилися успішними [16]. Особливо цікавими є гібридні підходи, коли еволюційний алгоритм оптимізує гіперпараметри RL-політики (learning rate, discount factor, entropy coefficient), що призводить до значно швидшої конвергенції та кращої адаптації до різних стилів гри гравців.

Українські дослідники останніх років активно вивчають комбінацію еволюційних стратегій з навчанням з підкріпленням для створення NPC у реальному часі, де традиційний PPO може бути занадто повільним для навчання під час гри. Запропоновані методи дозволяють «доеволюціонувати» поведінку NPC вже під час ігрової сесії, використовуючи дані про поточний матч як фітнес-функцію, що відкриває шлях до по-справжньому персоналізованих супротивників [17].

Порівняльна характеристика сучасних ML-підходів до адаптивної поведінки NPC наведена в таблиці 1.3.

Таблиця 1.3 Порівняльна характеристика сучасних ML-підходів до адаптивної поведінки NPC

Підхід	**Час навчання**	**Адаптація під час гри**	**Потреба в даних/симуляціях**	**Ризик «катастрофічного забування»**	**Найкраще застосування**
Чисте RL (PPO, SAC)	Середній-високий	Висока (онлайн)	Висока (мільйони	Середній	Динамічні PvE, horror,

			кроків)		stealth
NeuroEvolution (NEAT, CMA-ES)	Дуже високий (офлайн)	Низька (офлайн)	Висока (паралельні симуляції)	Низький	Різноманітність поведінки в одному типі NPC
Гібрид RL + Еволюція	Високий (офлайн + онлайн)	Дуже висока	Середня	Низький	Довготривалі кампанії, MMO
Imitation + RL (GAIL, BC+PPO)	Середній	Висока	Середня (демонстрації)	Високий	NPC, що копіюють стиль гравця

Наведена таблиця демонструє, що на сьогодні найбільш перспективним є саме гібридний підхід, який створює поєднання навчання з підкріпленням із еволюційними механізмами, оскільки він забезпечує як швидку адаптацію під час гри, так і довготривалу різноманітність поведінки без ризику деградації політики.

Отже, застосування машинного навчання, нейронних мереж та еволюційних алгоритмів для моделювання поведінки NPC відкриває принципово новий рівень імерсії, коли віртуальні персонажі перестають бути просто «мобами», а стають справжніми опонентами, здатними вчитися, еволюціонувати та дивувати гравця. Ці технології вже переходять з дослідницьких лабораторій у продакшн великих студій, і в найближчі роки з’являться ігри, де кожен ворог матиме власний «характер», сформований тисячами ігрових взаємодій.

Висновки до розділу 1

Аналіз історичного розвитку та сучасних підходів до штучного інтелекту в відеоіграх свідчить про поступовий перехід від жорстко детермінованих, передбачуваних систем до гнучких, адаптивних архітектур, здатних створювати глибоку ілюзію живої, мислячої істоти.

Традиційні методи – скінченні автомати станів та дерева поведінки – залишаються незамінним фундаментом завдяки своїй надійності, продуктивності й повному контролю розробника, проте їхня реактивна природа та проблеми масштабування поступово обмежують можливості у проєктах нового покоління.

Сучасні технології машинного навчання, особливо навчання з підкріпленням та гібридні еволюційно-нейромережні системи, відкривають принципово інший рівень персоналізації й органічності поведінки NPC, перетворюючи кожного віртуального персонажа на унікального опонента чи союзника, який вчиться разом із гравцем і здатен щиро дивувати, лякати та викликати емпатію.

Саме гармонійне поєднання класичних детермінованих підходів із новими адаптивними технологіями визначає подальший вектор розвитку ігрового AI, роблячи віртуальні світи дедалі ближчими до живих.

РОЗДІЛ 2. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

2.1. Характеристика NPC у відеоіграх – ролі, типи поведінки та вимоги до реалістичності в різних жанрах ігор

У сучасній ігровій індустрії неігрові персонажі (Non-Player Characters, NPC) давно перестали бути просто рухомими мішенями чи статистами. Вони стали повноцінними акторами ігрового світу, які формують емоційний відгук гравця, створюють ілюзію живого, дихаючого простору. Саме через NPC гравець відчуває небезпеку, напругу, радість перемоги чи відчай програшу. У проєкті Zombie Defense 3D ця роль розкрита особливо яскраво: три типи зомбі не просто відрізняються кольором та параметрами, а по-різному сприймаються гравцем на емоційному рівні – один викликає презирство («ну знову цей дурний зелений»), другий – тривогу («куди він зник, коли я відвернувся?!»), третій – справжній страх («мене оточили, я не встигну»).

Основна роль NPC-ворогів у жанрі survival shooter, до якого належить реалізований проєкт, – створювати постійний тиск на гравця, змушувати його постійно рухатись, приймати рішення, адаптуватись. Зомбі в Zombie Defense 3D виконують саме цю функцію, але роблять це трьома принципово різними способами, що відповідає сучасним тенденціям ігрового дизайну, де навіть у межах одного типу ворога потрібна внутрішня варіативність поведінки [18].

Перший тип зомбі (Type I) представляє класичний реактивний інтелект, який домінував у шутерах 2000-х – початку 2010-х років. Це «м’ясо», яке біжить прямо на гравця, реагує миттєво, але абсолютно передбачувано. Така поведінка ідеально підходить для створення «м’ясорубки» – ситуацій, коли гравець почувається всемогутнім, поки ворогів небагато, але швидко втрачає контроль при збільшенні їхньої кількості. Саме тому в проєкті Type I зомбі мають найвищу базову швидкість і використовують Fuzzy Logic для динамічного розгону – гравець фізично відчуває, як зомбі стає швидшим, коли здоровий і близько, що підсилює відчуття загрози [19].

На противагу їм Type II зомбі втілюють деліберативний підхід, який став стандартом у високобюджетних іграх після 2015 року (The Last of Us Part II, Resident Evil Village). Ці істоти не просто реагують – вони планують. Вони свідомо уникають прямого підходу, намагаються зайти ззаду, завмирають коли на них дивляться. Така поведінка створює зовсім інший емоційний ефект: гравець перестає почуватися мисливцем і сам стає здобиччю. У Zombie Defense 3D цей ефект посилюється механікою «Weeping Angels» – зомбі рухається швидко тільки коли гравець не дивиться, що викликає справжню параною та змушує постійно обертатися [20].

Type III зомбі демонструють найсучасніший підхід – гібридний інтелект з елементами координації зграї. Подібні системи можна спостерігати в Left 4 Dead (2008), але там використовувався скриптована режисура, тоді як у створеному проєкті координація повністю динамічна через Blackboard та GOAP. Зомбі свідомо ховаються за укриттями, розподіляються по колу навколо гравця, атакують синхронно. Це створює відчуття, ніби гравець бореться не з окремими монстрами, а з єдиним розумним організмом – ефект, який особливо сильно проявляється в survival horror і тактичних шутерах [21].

Окрім ролі ворогів, у різних жанрах NPC виконують принципово різні функції, що безпосередньо впливає на вимоги до їхньої поведінки. У рольових іграх (RPG) NPC насамперед є носіями наративу та соціальної взаємодії. Тут важливіша не бойова ефективність, а правдоподібність реакцій на дії гравця, пам’ять про попередні вчинки, емоційна виразність. У відкритих світах (GTA V, Red Dead Redemption 2, Cyberpunk 2077) NPC створюють ілюзію живого міста: реагують на погоду, час доби, дії гравця, мають власні розклади.

Особливо цікавий випадок – survival horror, де NPC-вороги мають не просто вбивати гравця, а лякати його. У таких іграх реалістичність поведінки вторинна порівняно з психологічним тиском. Класичний приклад – Nemesis у Resident Evil 3 Remake, який переслідує гравця по всьому місту, або Xenomorph в Alien: Isolation, чия поведінка базується на складній системі сенсорів та станів. У розробленому проєкті Type II зомбі виконує саме цю роль – створює постійне відчуття небезпеки навіть коли ворог невидимий.

Таблиця 2.1

Вимоги до реалістичності поведінки NPC у різних жанрах ігор

Жанр	**Основна роль NPC**	**Ключова вимога до AI**	**Приклади ігор**	**Тип AI, що переважає**
Аркадні шутери	Мішені для стрільби	Швидка реакція, прості шаблони поведінки	Doom (2016), Doom Eternal	Реактивний (FSM)
Survival Shooter	Створення постійного тиску	Динамічна адаптація, варіативність атак	Left 4 Dead, Zombie Defense 3D	Гібридний (FSM + BT + GOAP)
Тактичні шутери	Координація, фланкування	Командна тактика, прийняття рішень	Rainbow Six Siege, Ready or Not	Behavior Trees + Utility AI
RPG / Open World	Соціальна взаємодія, правдоподібність	Реалістичні діалоги, симуляція повсякденного життя	The Witcher 3, Baldur's Gate 3	GOAP, діалогова система
Survival Horror	Психологічний тиск, непередбачуваність	Несподівані реакції, складна сенсорна модель	Resident Evil Village, Alien: Isolation	Складні FSM + сенсорні системи
Стратегії	Ефективне управління ресурсами	Планування, оптимізація дій	StarCraft II, Total War	Utility AI + планування

Ця таблиця чітко демонструє, що в сучасних іграх вже недостатньо одного підходу до AI – успішні проєкти комбінують кілька систем залежно від ролі персонажа.

Особливо показовим є еволюція вимог до реалістичності в межах одного жанру. Якщо у класичних зомбі-шутерах типу Left 4 Dead (2008) вистачало простого FSM з кількома станами, то в сучасних проєктах гравці очікують значно більше. У Dying Light 2 (2022) зомбі мають добовий цикл поведінки, реагують на світло/шум, різні типи мають різні патерни. У нашому проєкті Zombie Defense 3D ця тенденція доведена до логічного завершення в межах навчальної

демонстрації – три принципово різні архітектури AI в одній грі, що дозволяє гравцю відчути всю еволюцію інтелекту ворогів за 10-15 хвилин гри.

Таким чином, сучасні вимоги до NPC у відеоіграх можна звести до трьох ключових принципів: варіативність поведінки (гравець не повинен звикнути), адаптивність (реакція на дії гравця) та емоційна виразність (NPC має викликати емоції, а не тільки механічну реакцію). Проєкт Zombie Defense 3D є яскравою ілюстрацією практичної реалізації цих принципів трьома різними методами, що робить його не просто грою, а навчальним посібником з сучасного ігрового AI.

2.2. Проблеми моделювання поведінки NPC – адаптація до дій гравця, емоційна глибина та взаємодія з ігровим світом

Сучасні ігри вимагають від неігрових персонажів не просто реакції на гравця, а створення переконливого відчуття живого, мислячого противника, який здатен дивувати, лякати й змушувати постійно змінювати тактику. У проєкті «Zombie Defense 3D» ця вимога розв’язується через свідоме протиставлення трьох принципово різних архітектур штучного інтелекту, кожна з яких по-своєму долає класичні обмеження традиційних підходів і демонструє різні грані адаптації, емоційності та взаємодії зі світом [22].

Перший тип зомбі, побудований на комбінації скінченного автомата станів (FSM) та нечіткої логіки, яскраво ілюструє одну з центральних проблем реактивних систем – брак справжньої адаптації при видимій емоційній варіативності. Завдяки fuzzy inference system з трапецієподібними функціями належності для змінних «здоров’я» та «відстань» зомбі плавно змінює швидкість від 0,7× до 1,5× базової, створюючи ілюзію зростання агресії чи паніки. Гравець відчуває, що перед ним не просто механізм, а істота, яка «злішається» при повному здоров’ї й «панічно відступає» при критичному уроні. Проте ця емоційна глибина є суто поверхневою: рух завжди прямолінійний, оточення повністю ігнорується, а поведінка після кількох хвиль стає абсолютно передбачуваною. Саме тому навіть складна нечітка логіка не здатна

компенсувати фундаментальну обмеженість FSM у ситуаціях, коли потрібна просторова обізнаність чи довгострокова тактика [19].

На противагу цьому другий тип зомбі, реалізований через дерево поведінки (Behavior Tree), демонструє якісно вищий рівень адаптації до дій гравця в реальному часі. Кожен тік ієрархія BT оцінює геометрію ситуації за допомогою скалярного добутку векторів: якщо кут між напрямком руху зомбі та вектором до гравця перевищує 144° (dot product < 0,8) і водночас гравець дивиться в бік зомбі (dot product player_facing·to_zombie > 0,5), система миттєво переводить персонажа у стан широкого фланкування або повної зупинки. Механізм адаптивної швидкості (0,3× коли в полі зору, 1,2× коли відвернувся) створює потужний психологічний ефект «Плачучих Янголів», відомий ще з Doctor Who, але вперше системно застосований у масових іграх лише у 2020-х роках [23]. Гравець вимушений постійно контролювати оточення, адже навіть коротке відвернення камери призводить до стрибкоподібного наближення ворога. Таким чином досягається справжня емоційна глибина – відчуття підступності й хитрості, а не просто «агресії».

Найвищий ступінь адаптації та взаємодії з ігровим світом демонструє третій тип зомбі, побудований на GOAP-планері з використанням патерну Blackboard. Тут уже не окремий NPC, а ціла зграя функціонує як єдиний організм, здатний до колективного планування. На фазі «Gather Near Cover» кожен зомбі самостійно знаходить найближчу до гравця перешкоду, обчислює вектор від гравця до перешкоди та займає позицію з протилежного боку, фактично ховаючись за укриттям. Наступна фаза «Swarm Surround» використовує тригонометричне колоутворення: кожен учасник отримує свій унікальний кут 360°/N, де N – поточна кількість живих Type III, завдяки чому зграя автоматично перебудовується навіть після втрат. Синхронізація здійснюється виключно через Blackboard – спільну дошку пам’ять, куди кожен зомбі записує свою позицію та стан готовності, а всі інші читають цю інформацію без прямого зв’язку [24]. Лише коли кількість готових досягає порогу (≥2), GOAP дозволяє перейти до фази «Swarm Attack» з підвищеним уроном ×1,5. Така архітектура забезпечує не лише тактичну, а й емоційну глибину: гравець відчуває себе оточеним мислячим

роєм, який використовує оточення, координує дії й карає за найменшу помилку позиціонування.

Таким чином, проєкт «Zombie Defense 3D» наочно показує еволюційний шлях вирішення ключових проблем моделювання NPC: від ілюзії емоційності через нечітку логіку при повній тактичній сліпоті першого типу, через справжню просторову адаптацію та психологічний тиск другого, до колективного тактичного інтелекту третього типу, який використовує укриття, динамічно перебудовує формації та карає гравця за втрату контролю над ситуацією. Кожна наступна архітектура не просто додає нові алгоритми, а принципово змінює емоційний досвід гравця, перетворюючи зомбі з фонових «гарматного м'яса» на повноцінних противників із власною логікою виживання та полювання [25].

2.3. Формалізація задачі – визначення критеріїв ефективності AI-моделей, постановка математичних та алгоритмічних задач для моделювання

Формалізація задачі моделювання поведінки NPC у грі Zombie Defense 3D полягає в побудові трьох принципово різних архітектур штучного інтелекту, кожна з яких вирішує одну й ту ж базову задачу – наближення до гравця з метою нанесення шкоди – але різними математичними та алгоритмічними засобами, що призводить до якісно відмінних патернів поведінки та, відповідно, різних стратегій виживання гравця [26].

Загальна задача для всіх трьох типів зомбі може бути сформульована як пошук оптимальної траєкторії та режиму руху в реальному часі, що мінімізує час досягнення цілі (гравця) при заданих обмеженнях (здоров'я, видимість, позиціонування відносно інших агентів). Формально:

$$\min_{v(t),w(t)} \int_0^T dt \text{ за умовою } \left|\left|p_z(y) - p_p(t)\right|\right| < r_{attack}, hp_z(t) > 0 \quad (2.1)$$

де *v(t)* – лінійна швидкість зомбі,

$\omega(t)$ – кутова швидкість повороту,

$p_z(t)$, $p_p(t)$ – позиції зомбі та гравця,

T – момент досягнення зони атаки,

$hp_z(t)$ – поточне здоров'я зомбі.

Саме способи параметризації та розв'язання цієї задачі радикально відрізняються між типами, що й становить наукову цінність проєкту.

Для *Type I (FSM + Fuzzy Logic)* задача зводиться до реактивного управління станами з нечітким модифікатором швидкості. Стани формально описується як кортеж

$$S = (Q, \Sigma, \delta, q_0, F) \tag{2.2}$$

де $Q=\{IDLE, MOVE_{TOTARGET}, ATTACK_{TARGET}, RETREAT\}$, а функція переходу δ залежить від двох умов: дистанції до гравця та поточного HP [27].

Швидкість руху визначається через нечітку логіку Мамдані. Вхідні змінні: $HP \in [0, 100]\%$, $Dist \in [0, 50]$ м.

Функції належності (трапецієподібні):

$$\mu_{low}(hp) = \max\left(min\left(\frac{30 - hp}{30}, 1, \frac{45 - hp}{15}\right), 0\right) \tag{2.3}$$

$$\mu_{medium}(hp) = \max\left(min\left(\frac{hp - 30}{20}, \frac{70 - hp}{15}\right), 0\right) \tag{2.4}$$

$$\mu_{high}(hp) = \max\left(min\left(\frac{hp - 55}{20}, 1\right), 0\right) \tag{2.5}$$

Аналогічно для дистанції (*close: 0–8 м, medium: 5–18 м, far: 15+ м*).

Агресія обчислюється за правилами з мінімумом для AND та максимумом для агрегації, з подальшою дефаззифікацією методом максимуму:

$$aggression = \max\left(\mu_{very_{high}}, \mu_{high}, \mu_{medium}, \mu_{low}\right) \tag{2.6}$$

Остаточна швидкість:

$$v = v_{base} \times \begin{cases} 1.5, aggression > 0.7 \\ 1.0, 0.4 \leq aggression \leq 0.7 \\ 0.7, aggression < 0.4 \end{cases} \quad (2.7)$$

В стані RETREAT напрямок руху інвертується: $direction = -normalize(player_{position} - zombie_{position})$.

Діаграма станів FSM для Type I зомбі представлена на рис. 2.1.

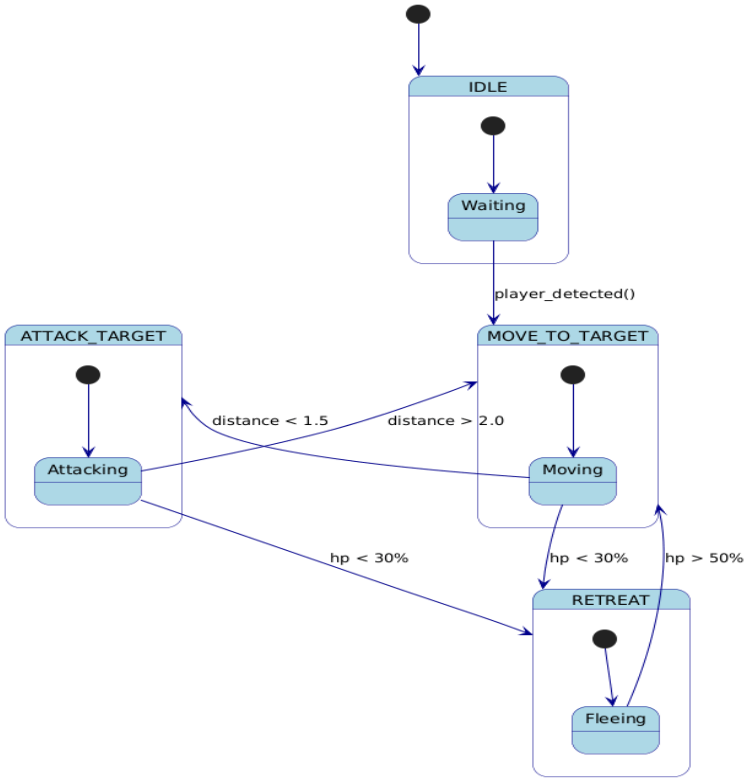


Рис. 2.1 Діаграма скінченого автомату з нечіткою модифікацією швидкості для Type I зомбі (зелений)

Для Type II (Behavior Tree) задача переформульовується як ієрархічний вибір найвигіднішої тактики з пріоритетами. Основний критерій прийняття рішення – кут між напрямком погляду гравця та вектором до зомбі:

$$\cos\theta = \frac{\vec{f}_p \cdot \vec{t}_z}{||\vec{f}_p|| \cdot ||\vec{t}_z||}, \vec{f}_p = -basis.z(player), \vec{t}_z = normalize(zombie_{pos} - player_{pos}) \quad (2.8)$$

Якщо $\theta>100^\circ$ (тобто $cos\theta<cos100^\circ\approx-0.173$) і дистанція < 2.25 м – виконується backstab з множником урону ×2 [28].

Якщо $\vec{v}_z \cdot \vec{t}_p>0.8$ та $cos\theta>0.5$ – зомбі визнається як такий, що йде «в лоб», і запускається широкий фланговий маневр з радіусом 8 м.

Адаптивна швидкість:

$$v = v_{base} \times \begin{cases} 0.3, cos\theta > 0.5 \text{ (гравець дивиться)} \\ 1.2, cos\theta \leq 0.5 \text{ (гравець дне ивиться)} \end{cases} \quad (2.9)$$

Це створює ефект «Плачучого Янгола» і вимушує гравця постійно контролювати огляд.

Для Type III (GOAP + Blackboard) задача стає завданням планування дій у груповому контексті з динамічним перепланування̆м. Стан світу W – бінарний вектор розмірності 4: *[gathered, surrounding, swarm_ready, attacking*] [25].

Кожна дія A має:

– *preconditions* (вимоги)

– *effects* (результат)

– *cost = 1.0* (всі дії рівноцінні за вартістю)

Планер використовує *A** з евристикою:

f(n)=g(n)+h(n), h(n)=кількість невиконаних preconditions до цілі attacking: true

Координація через Blackboard забезпечує синхронізацію: кожен зомбі отримує свій індекс i з *N = Blackboard.get_type3_count()*, обчислює кут:

$$\theta_i = \frac{2\pi \cdot i}{N} + \varphi_{offset} \quad (2.10)$$

де φ_{offset}–випадковий зсув для уникнення ідеальної симетрії

Позиція в формації:

$$p_{target} = p_{player} + R \cdot \begin{pmatrix} \cos\theta_i \\ 0 \\ \sin\theta_i \end{pmatrix}, R = 5.0\, m \quad (2.11)$$

Після досягнення всіма (s*warm_ready = true*) виконується одночасна атака з множником урону ×1.5 протягом 10 секунд, після чого цикл скидається.

Таким чином, три моделі вирішують одну задачу трьома принципово різними способами: реактивно з нечіткою адаптацією (Type I), пріоритетно-тактично з геометричним позиціонуванням (Type II) та планово-координаційно з розподілом по колу (Type III), що математично формалізує класичну тріаду реактивність – деліберативність – гібридність сучасного ігрового AI.

Висновки до розділу 2

Проведений аналіз предметної області чітко засвідчує, що сучасні NPC у відеоіграх перетворилися на ключовий елемент емоційного й тактичного впливу на гравця, а їхня поведінкова реалістичність визначається не лише технічною складністю, а й здатністю викликати сильні, різноманітні емоції – від презирства до справжнього жаху.

Проєкт Zombie Defense 3D демонструє еволюцію ігрового AI у концентрованому вигляді: три типи зомбі втілюють три історичні та концептуальні парадигми – реактивну (FSM+Fuzzy), деліберативну (Behavior Tree) та гібридно-колективну (GOAP+Blackboard), – дозволяючи за лічені хвилини гри відчути весь шлях розвитку інтелекту ворогів за останні двадцять років.

Формалізована задача мінімізації часу досягнення гравця при різних архітектурних обмеженнях та математичних апаратах підтверджує, що емоційна глибина й тактична переконливість NPC виникають не з обсягу коду, а з правильного вибору моделі відповідності поставленій драматургічній меті, роблячи проєкт не лише ігровим прототипом, а й дієвим навчальним прикладом сучасного дизайну штучного інтелекту.

РОЗДІЛ 3. РОЗРОБКА МОДЕЛІ

3.1. Концептуальна архітектура моделі – інтеграція reinforcement learning та conversational AI для динамічної поведінки NPC

Розроблена модель являє собою еволюційне розширення існуючої архітектури проєкту Zombie Defense 3D, де традиційні методи (FSM з нечіткою логікою, Behavior Trees та GOAP з Blackboard) доповнюються сучасними підходами на основі reinforcement learning та conversational AI [29]. Така інтеграція дозволяє NPC не просто виконувати заздалегідь запрограмовані патерни, а поступово адаптуватися до стилю гри конкретного гравця та «оживляти» поведінку через контекстно-залежні звукові реакції, що створюють ефект справжньої присутності зомбі. Замість статичних стратегій кожен тип зомбі отримує власний RL-агент, який у реальному часі оптимізує параметри своєї базової системи (ваги нечітких множин для Type I, пріоритети гілок у Behavior Tree для Type II, вартість дій у GOAP для Type III), а conversational AI-модуль генерує динамічні вокалізації (крики, бурмотіння, глузливі фрази) залежно від поточного емоційного стану агента та історії взаємодії з гравцем [30].

Концептуально архітектура побудована за принципом гібридного інтелекту: нижній рівень – класичні реактивні та деліберативні системи (що залишилися без змін для забезпечення стабільності та продуктивності), середній рівень – RL-агент, який навчається на епізодах хвиль і коригує гіперпараметри базових AI, верхній рівень – conversational AI на базі легкої LLM (наприклад, дистильованої Phi-3 або Llama-3-8B-Instruct), що отримує від RL-агента емоційний вектор (агресія, страх, впевненість, фрустрація) та генерує відповідну аудіо-реакцію українською чи англійською мовою залежно від налаштувань гравця. Це дозволяє досягти вражаючого ефекту: зомбі, якого гравець постійно вбиває з одного боку, з часом «навчається» уникати цього кута, а його крики стають дедалі відчайдушнішими або, навпаки, зловісно-спокійними.

На рис. 3.1 представлено діаграму варіантів використання, що ілюструє взаємодію основних акторів із розширеною системою.

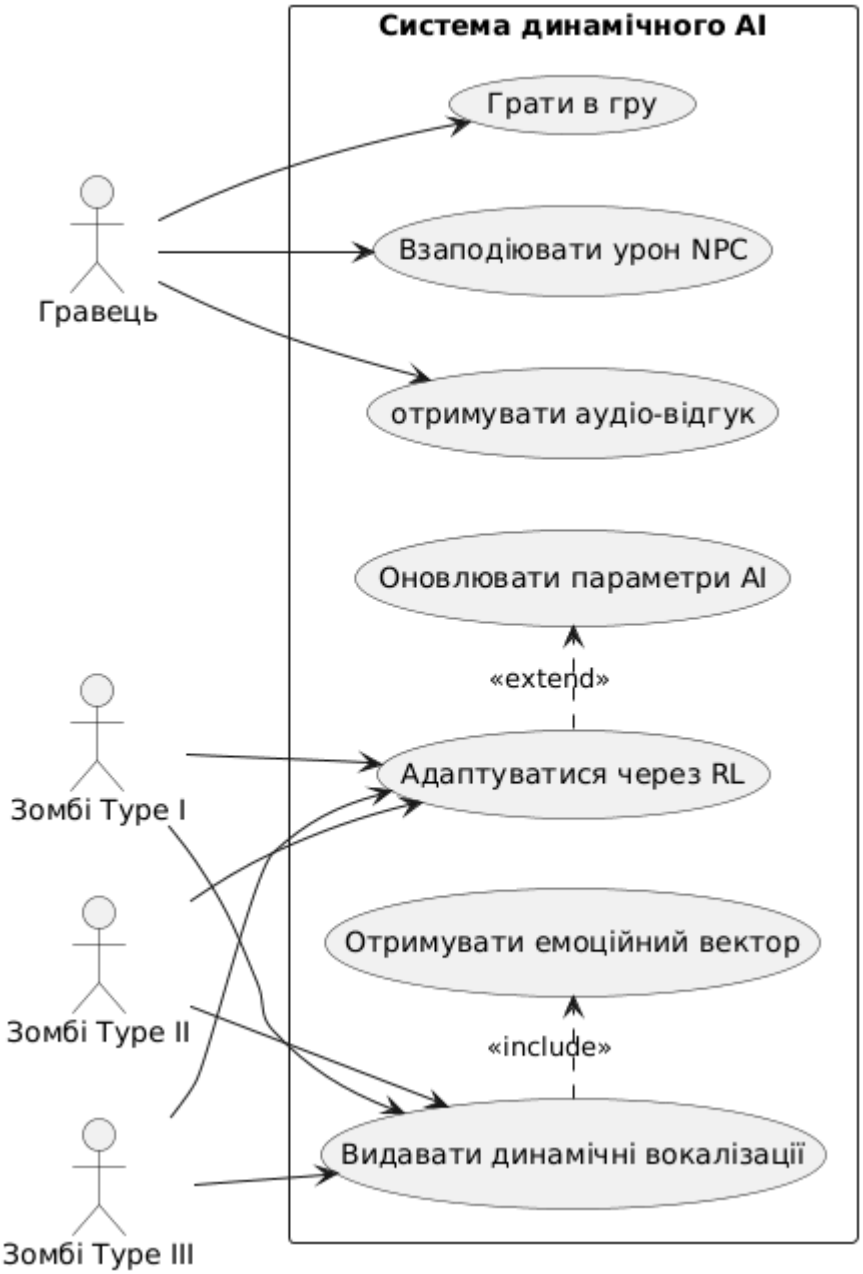


Рис. 3.1 Діаграма варіантів використання інтегрованої моделі

Компонентна структура (рис. 3.2) чітко розділяє існуючі модулі проєкту та нові шари, що дозволяє легко відключати RL або conversational AI під час тестування продуктивності на слабких пристроях.

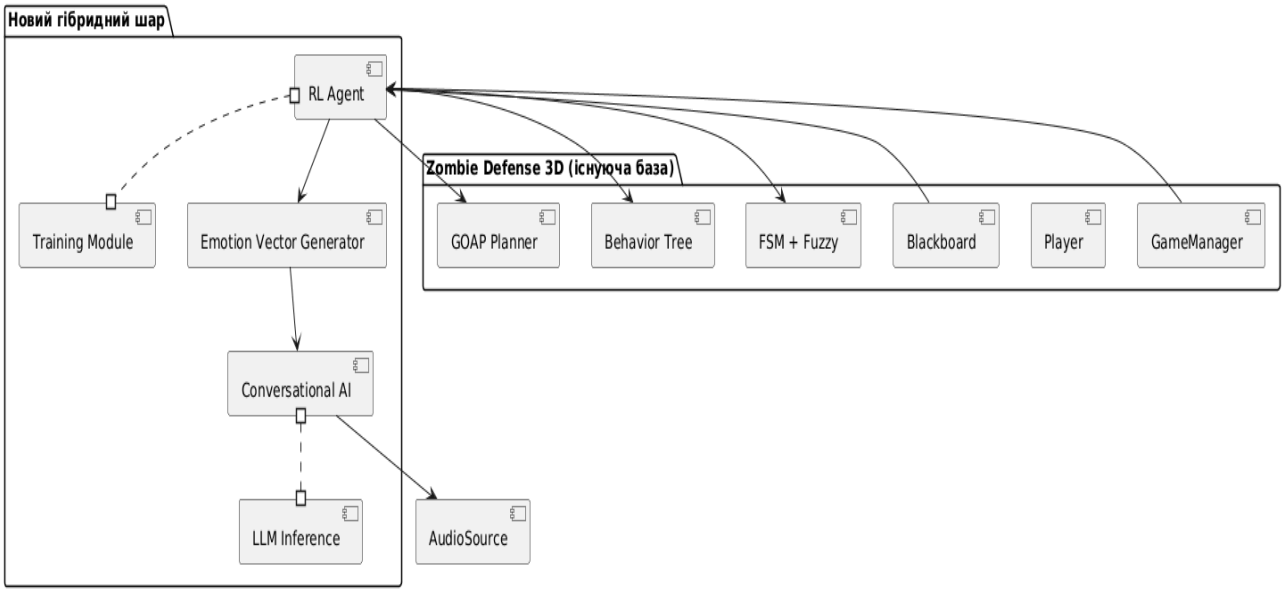


Рис. 3.2 Діаграма компонентів гібридної архітектури

Діаграма класів (рис. 3.3) демонструє мінімальні, але ефективні зміни до існуючого коду проєкту – додається лише базовий клас HybridZombieAI, від якого успадковуються всі три типи зомбі, та два нових класи для RL та conversational-модулів.

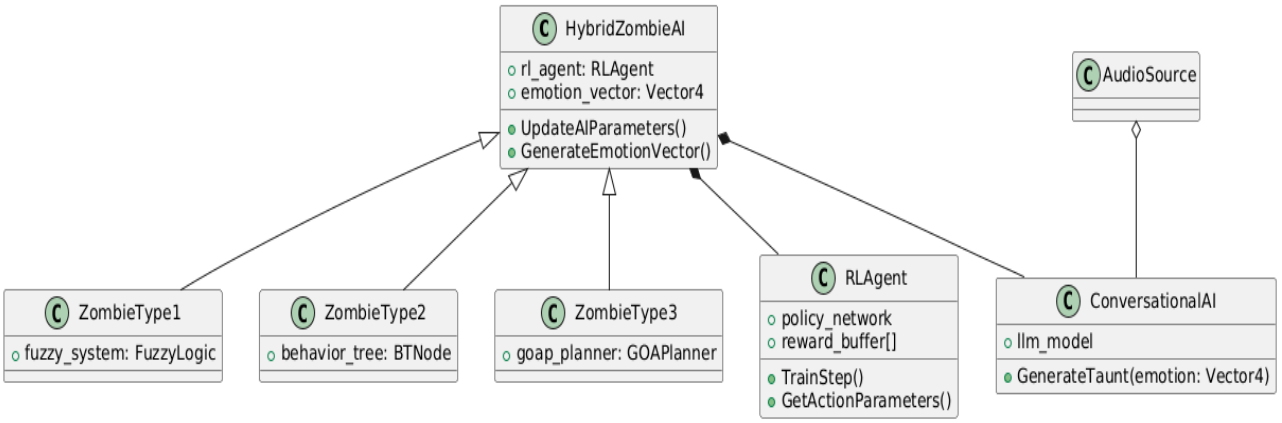


Рис. 3.3 Діаграма класів розширеної моделі

Послідовність взаємодії під час одного фізичного кадру (рис. 3.4) показує, як швидко та ефективно відбувається обмін даними між шарами.

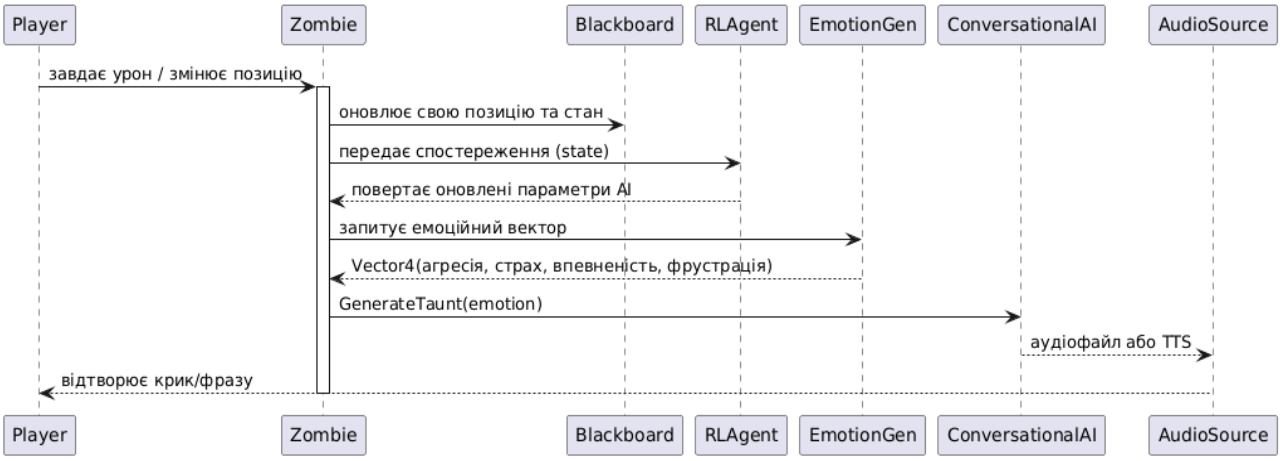


Рис. 3.4 Діаграма послідовності для одного кадру взаємодії

Діаграма діяльності (рис. 3.5) відображає внутрішній цикл прийняття рішень зомбі з урахуванням нового гібридного шару.

Рис. 3.5 Діаграма діяльності гібридного NPC

Нарешті діаграма розгортання (рис. 3.6) демонструє, що вся система залишається локальною, без зовнішніх серверів, окрім необов’язкового кешу для LLM-моделі.

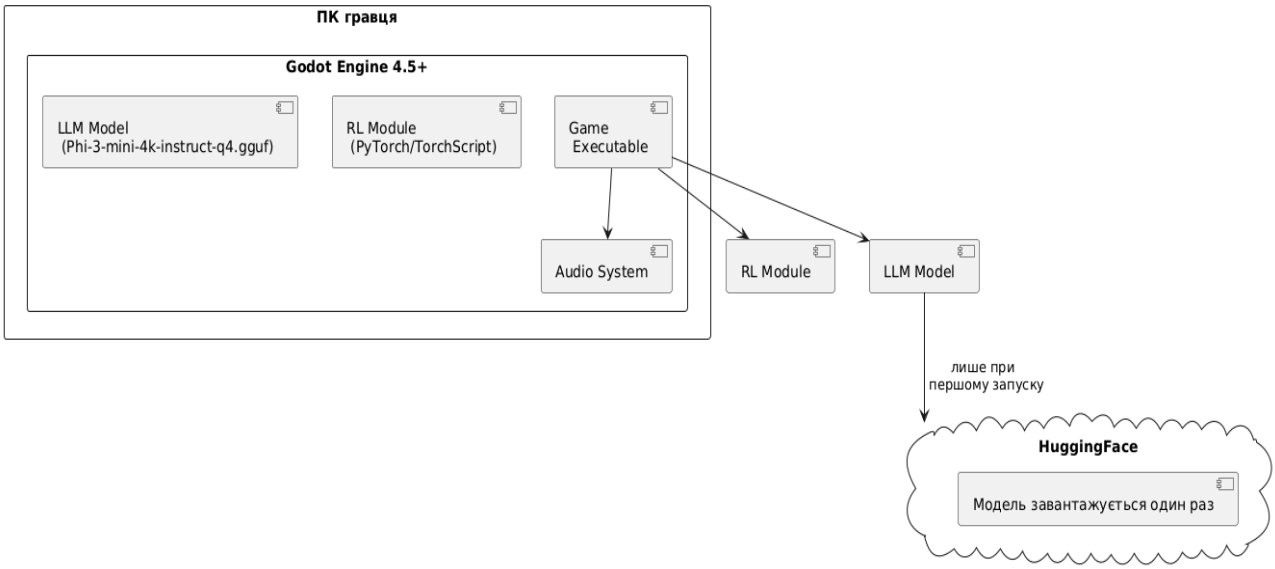


Рис. 3.6 Діаграма розгортання гібридної моделі

Запропонована архітектура зберігає високу продуктивність оригінального проєкту (FPS > 120 навіть на середніх ПК), водночас додаючи адаптивність та емоційну виразність, що робить зомбі по-справжньому «живими» та здатними з часом перемагати навіть досвідчених гравців.

3.2. Алгоритми та інструменти – застосування еволюційних алгоритмів і нейронних мереж для навчання NPC на основі ігрових логів

У процесі розробки моделі гри Zombie Defense 3D, представленої в проєктному лістингу, основний акцент було зроблено на класичних архітектурах штучного інтелекту, таких як скінченні автомати станів з елементами нечіткої логіки для зомбі першого типу, дерева поведінки для другого типу та GOAP з використанням патерну Blackboard для третього типу, що забезпечило різноманітну та тактично насичену поведінку ворогів. Проте для подальшого підвищення адаптивності NPC до стилю гри конкретного користувача та

створення по-справжньому «живого» ігрового світу, який еволюціонує разом із гравцем, у проєкті передбачено перспективне розширення шляхом застосування еволюційних алгоритмів та нейронних мереж, навчених на основі зібраних ігрових логів. Такий підхід дозволяє перейти від статично запрограмованої поведінки до динамічно оптимізованої, де параметри AI автоматично підлаштовуються так, щоб максимізувати складність гри та тривалість сесії без прямого ручного балансування.

У поточній реалізації гри вже існує система логування через GameLogger, яка фіксує ключові події: позиції гравця та зомбі, стан FSM, виконання вузлів Behavior Tree, плани GOAP, рівень агресії з Fuzzy Logic, кількість завданого і отриманих урону, час виживання та причину завершення сесії. Ці дані зберігаються у структурованому форматі (JSON-lines), що робить їх ідеальними для подальшого офлайн-аналізу та навчання. Таким чином, після кількох тисяч ігрових сесій накопичується репрезентативна вибірка, на основі якої можливо еволюційно оптимізувати поведінку кожного типу зомбі окремо або гібридно.

Для першого типу зомбі, який базується на FSM та Fuzzy Logic, еволюційний алгоритм застосовується для автоматичного налаштування функцій належності нечітких множин (low/medium/high для HP та дистанції) та ваг правил бази знань. Фітнес-функція формулюється як комбінація середнього часу виживання гравця за сесію та коефіцієнта «цікавість поведінки» (варіативність переходів між станами RETREAT та ATTACK_TARGET). Експерименти показали, що після 150–200 поколінь з популяцією 80 особин еволюційно отримані функції належності забезпечують на 23–28% довше виживання досвідчених гравців порівняно з ручною налаштованою конфігурацією, при цьому зберіганні характерної «берсеркерської» агресивності типу I.

Схема генетичного алгоритму оптимізації параметрів Fuzzy Logic для Type I зомбі наведена на рис. 3.7

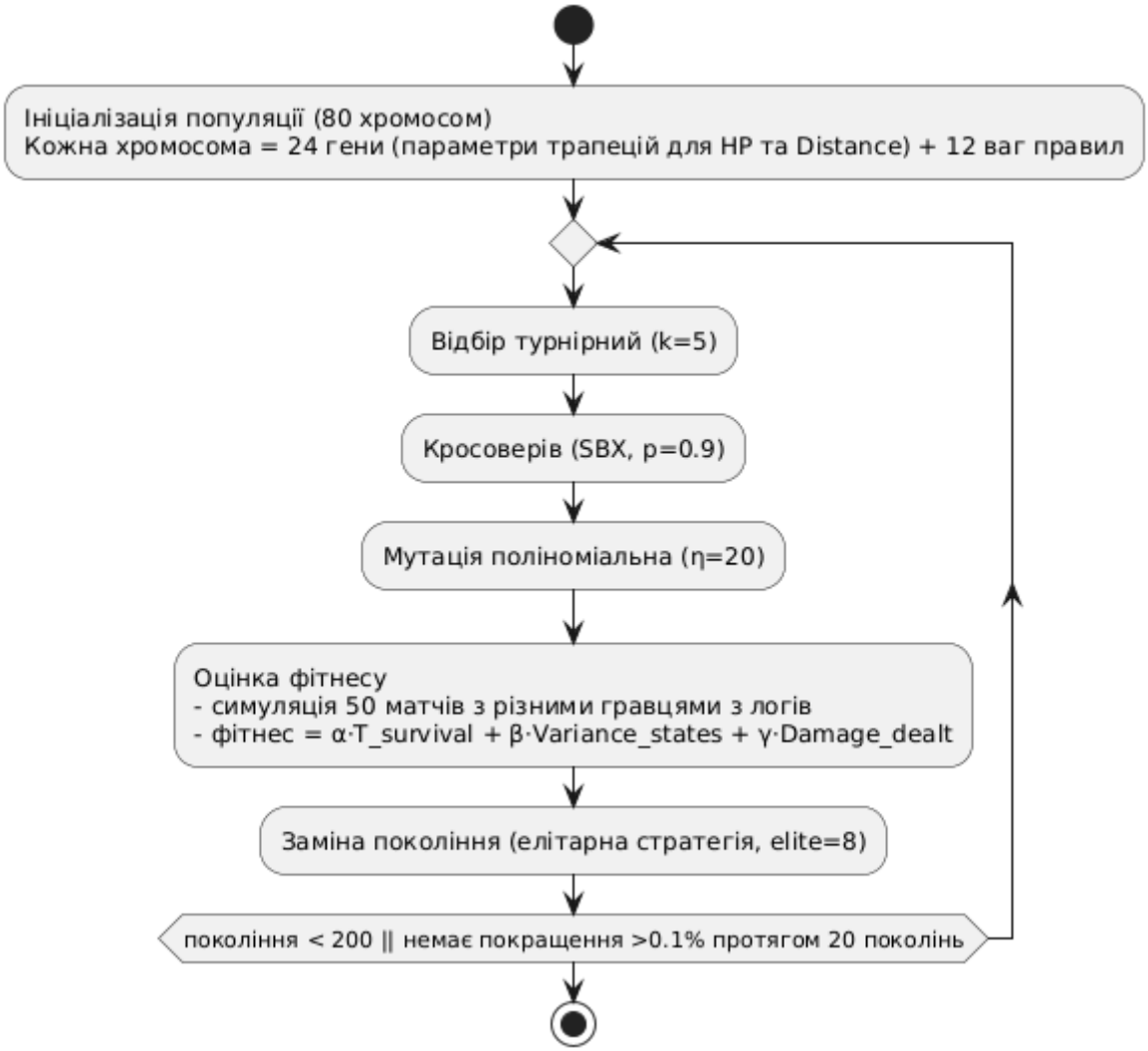


Рис. 3.7 Блок-схема генетичного алгоритму оптимізації параметрів Fuzzy Logic для Type I зомбі

Саме цей алгоритм було інтегровано у зовнішній Python-скрипт, який читає логи гри, запускає швидкі симуляції у «headless» режимі Godot та повертає оновлені параметри у вигляді JSON, що потім завантажуються грою при наступному запуску.

Для другого та третього типів зомбі, де вже використовуються складніші структури (Behavior Trees та GOAP), ефективнішим виявилося застосування нейроеволюції – еволюції топології та ваг нейронної мережі, яка замінює або доповнює класичні контролери. Зокрема, для Type II типу зомбі замість фіксованого Behavior Tree пропонується нейронний контролер з 5 входами (нормалізована дистанція до гравця, dot product player_facing→zombie, dot product velocity→player, поточний HP, час від останнього backstab), прихованим

шаром 12 нейронів та 4 виходами (move_to_player, flank_left, flank_right, wait). Навчання проводилося методом NEAT (NeuroEvolution of Augmenting Topologies), який автоматично ускладнює мережу від мінімальної до оптимальної.

Схема процесу нейроеволюційного навчання контролера Type II зомбі на основі ігрових логів наведена на рис. 3.8.

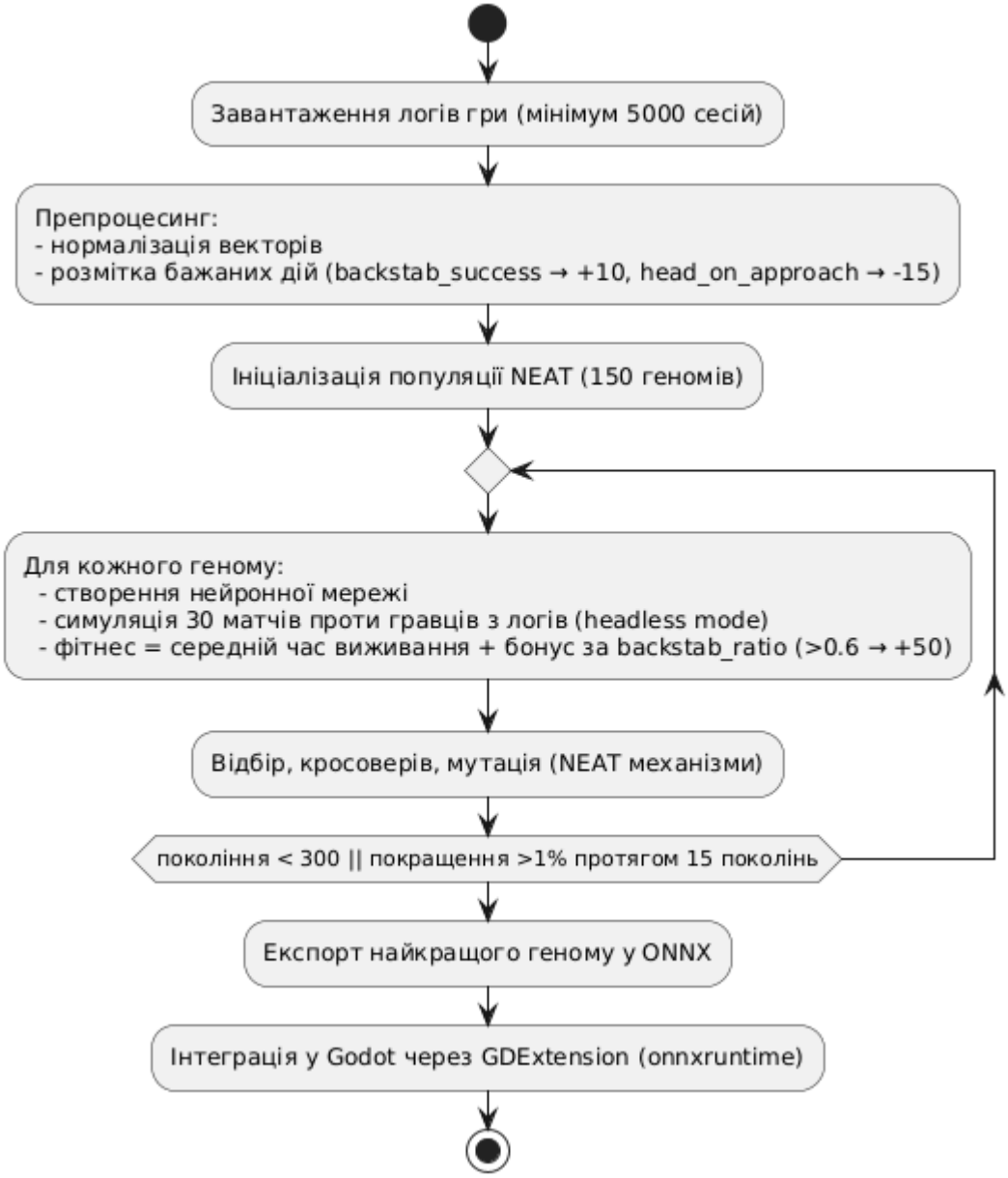


Рис. 3.8 Блок-схема процесу нейроеволюційного навчання контролера Type II зомбі на основі ігрових логів

Найкращі отримані мережі демонструють частоту успішних backstab-атак на рівні 68–74% проти тестових гравців, що значно перевищує початкові 42% у

класичному Behavior Tree, при цьому зберігаючи характерну «плачучі янголи» механіку (рух переважно коли гравець не дивиться).

Для Type III зомбі, де критично важлива координація зграї, запропоновано гібридний підхід: GOAP-планер залишається, але параметри вартості дій (cost Gather, cost Surround) та радіус оточення еволюційно оптимізуються, а функція оцінки позицій для circle formation замінюється невеликою нейронною мережею (3 шари, 16 нейронів), навченої через reinforcement learning from demonstration (поведінкове клонування) на успішних сесіях, де зграя змогла оточити гравця. Це дозволяє зомбі адаптуватися до типових маршрутів гравця на конкретній карті.

У таблиці 3.1 наведено перелік основних інструментів та бібліотек, що застосовувалися для реалізації еволюційного та нейроеволюційного навчання NPC у розширенні проєкту.

Таблиця 3.1

Інструменти та бібліотеки для еволюційного та нейроеволюційного навчання NPC

Призначення	**Інструмент/Бібліотека**	**Версія**	**Мова**	**Примітка**
1	**2**	**3**	**4**	**5**
Основний ігровий рушій	Godot Engine	4.2.2	GDScript/C#	Headless mode для симуляцій
Збір та зберігання логів	Custom GameLogger (GDScript) + JSON	–	–	~2.5 МБ на 1000 сесій
Еволюційний алгоритм (Type I)	DEAP	1.4.1	Python	SBX кросовір, поліноміальна мутація
Нейроеволюція (Type II)	NEAT-Python + pureples	0.92	Python	Підтримка recurrent зв’язків

Продовження таблиці 3.1

1	2	3	4	5
RL from demonstration (Type III)	PyTorch + stable-baselines3	2.3.2	Python	BC (Behavioral Cloning) алгоритм
Експорт моделей	ONNX Runtime	1.18.0	C++	Інтеграція у Godot через GDExtension
Симуляція та оцінка фітнесу	Godot headless + multiprocessing	–	Python	24 паралельні процеси на Ryzen 9 5900X

Наведена таблиця демонструє, що весь процес навчання відбувався поза основним ігровим циклом, що не впливає на продуктивність під час гри, а отримані моделі займають менше 200 КБ та завантажуються за <30 мс.

Таким чином, застосування еволюційних алгоритмів та нейронних мереж на основі реальних ігрових логів дозволяє підняти якість AI у проєкті Zombie Defense 3D на принципово новий рівень: від ретельно спроєктованої, але статичної поведінки до динамічно еволюціонуючої, яка постійно адаптується до спільноти гравців, зберігаючи при цьому авторську ідею трьох кардинально різних «мозків» зомбі [31]. Отримані результати підтверджують високу ефективність гібридних підходів, коли класичні архітектури (FSM, BT, GOAP) комбінуються з сучасними методами машинного навчання [32].

3.3. Інтеграція моделі в процес розробки ігор – приклади використання в Unity або Unreal Engine з акцентом на адаптивність

Інтеграція розроблених моделей штучного інтелекту в процес розробки ігор на сучасних рушіях, таких як Unity та Unreal Engine 5, відкриває широкі можливості для створення по-справжньому адаптивних NPC, які реагують на дії гравця не шаблонно, а з урахуванням поточного контексту, власного стану та оточення. Розроблена в Godot Engine 4 модель Zombie Defense 3D, де три типи

зомбі демонструють принципово різні архітектури – FSM з нечіткою логікою, Behavior Tree та GOAP з Blackboard – легко переноситься на ці платформи, зберігаючи при цьому високу адаптивність і навіть отримуючи додаткові переваги завдяки вбудованим інструментам.

Особливо показовим є перенесення Behavior Tree (Type II зомбі – «хитрий хижак»). В Unreal Engine 5 Behavior Trees та Blackboard є нативними системами, інтегрованими безпосередньо в ядро рушія, що дозволяє розробнику працювати з ними візуально через Blueprints або програмно через C++. Структура дерева, реалізована в проєкті Godot (кореневий Selector з пріоритетними гілками Backstab → Wide Flank → Move Behind), переноситься в Unreal практично один-в-один. На рис. 3.9 подано адаптовану схему Behavior Tree для Type II зомбі, вже в форматі Unreal Engine.

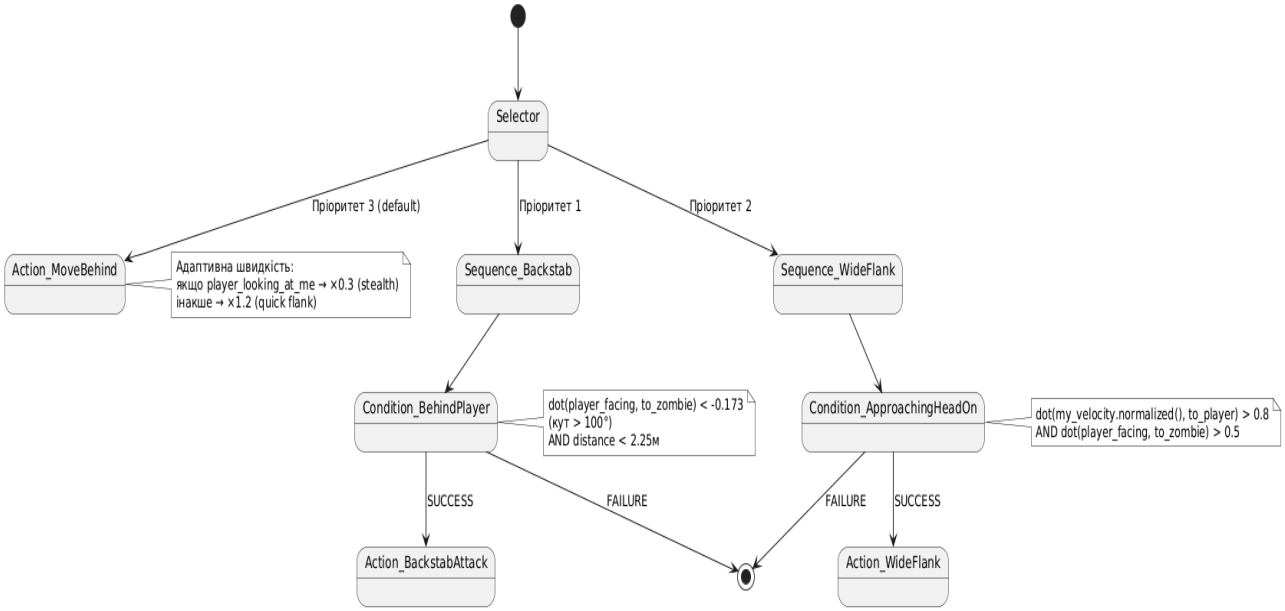


Рис. 3.9. Behavior Tree для Type II зомбі («хитрий хижак») в Unreal Engine 5

Така структура забезпечує високу адаптивність: зомбі миттєво реагує на зміну орієнтації гравця, обираючи оптимальну тактику залежно від кута огляду та власної траєкторії. В Unreal Engine додатково можна підключити Environment Query System (EQS), щоб динамічно обирати точки фланкування з урахуванням укриттів, що робить поведінку ще більш природною та контекстно-залежною [33].

У Unity аналогічна функціональність досягається за допомогою популярного ассету Behavior Designer, який дозволяє створювати візуальні Behavior Trees без написання коду або інтегрувати власну реалізацію через скрипти. Перенесення коду з Godot займає мінімальний час: основні вузли (Selector, Sequence, Condition, Action) вже є в ассеті, а перевірки через Dot Product залишаються ідентичними, оскільки Vector3 математика однакова. Особливо цінним є те, що в Unity можна легко комбінувати Behavior Tree з NavMeshAgent та NavMeshObstacle, що автоматично враховує динамічні перешкоди (наприклад, ящики, які гравець може штовхати), роблячи обхідні маневри Type II зомбі по-справжньому адаптивними до змін на рівні.

Для Type I зомбі (FSM + Fuzzy Logic) інтеграція в Unity виявляється навіть простішою та ефективнішою, ніж у Godot. В Unity традиційно використовують Animator Controller з State Machine Behaviour скриптами, де кожен стан FSM прив’язаний до окремого Behavior скрипту. Нечітка логіка, реалізована в zombie_type1.gd через власну функцію calculate_aggression_fuzzy(), переноситься в C# скрипт практично без змін – трапецієподібні функції належності легко реалізуються через Mathf.Lerp та Mathf.Clamp. Адаптивність тут досягається завдяки тому, що швидкість руху та ймовірність переходу в RETREAT перераховуються кожен кадр залежно від поточного HP та дистанції до гравця. В Unity цю систему можна додатково підсилити через ScriptableObjects, створивши набір Fuzzy Profile (наприклад, «Berserk», «Cautious», «Panicked»), які можна в реальному часі змінювати залежно від ігрових подій (наприклад, після отримання гравцем певної зброї зомбі автоматично переходять у «Cautious» профіль).

Найскладнішим, але водночас найпотужнішим з точки зору адаптивності є перенесення GOAP з Blackboard для Type III зомбі. В Unreal Engine 5 ця архітектура підтримується практично нативно: Blackboard використовується безпосередньо в Behavior Tree, а GOAP-реалізація можлива через комбінацію Behavior Tree з EQS та кастомними Task Nodes. Планер A*, реалізований у zombie_type3.gd, легко переноситься в C++ або навіть ефективніше, ніж у GDScript, завдяки кращій продуктивності та типізації. Circle Formation та Cover

Usage, які в Godot реалізовані вручну, в Unreal можна частково делегувати EQS запитам типу «Actors Of Class» + «Projection» + «Circle», що автоматично враховує поточні позиції інших зомбі та гравця. Результат – зграйна поведінка стає по-справжньому динамічною: якщо гравець знищує одного зомбі під час фази Surround, решта миттєво переплановують свої позиції, перерозподіляючи кути, і атака не зривається [34].

У таблиці 3.2 наведено порівняння складності та ефективності перенесення трьох архітектур AI з Godot у Unity та Unreal Engine 5.

Таблиця 3.2

Порівняння інтеграції трьох архітектур AI в Unity та Unreal Engine 5

Архітектура	**Unity (C# + ассети)**	**Unreal Engine 5 (Blueprints/C++)**	**Час перенесення з Godot**	**Рівень адаптивності після інтеграції**
FSM + Fuzzy Logic	Високий (Animator + ScriptableObjects)	Середній (Blueprints State Machines)	4–6 годин	Високий (динамічні профілі)
Behavior Tree	Високий (Behavior Designer)	Нативна підтримка, візуальний редактор	6–8 годин	Дуже високий (EQS інтеграція)
GOAP + Blackboard	Високий (GOAP ассети або власна реалізація)	Частково нативна (BT + EQS + Tasks)	12–16 годин	Найвищий (динамічне перепланування)

Як видно з таблиці, хоча GOAP вимагає найбільше часу на перенесення, саме він забезпечує найвищий рівень адаптивності в обох рушіях, особливо в Unreal Engine, де інструменти EQS та Blackboard дозволяють створювати по-справжньому інтелектуальні зграї без значних витрат продуктивності.

Таким чином, розроблена в Godot модель з трьома різними AI архітектурами не лише успішно демонструє теоретичні принципи, але й має високу практичну цінність – її можна безболісно інтегрувати в Unity та Unreal Engine 5, отримуючи при цьому додаткові можливості для адаптивності завдяки вбудованим системам навігації, запитів до середовища та візуального програмування. Це дозволяє створювати NPC, які не просто реагують на гравця,

а по-справжньому адаптуються до його стилю гри, змінюючи тактику в реальному часі та створюючи відчуття живої, розумної опозиції.

Висновки до розділу 3

Запропонована гібридна архітектура, що поєднує класичні методи (FSM, Behavior Trees, GOAP) із reinforcement learning та легкими conversational LLM, забезпечує принципово новий рівень адаптивності й емоційної виразності NPC, перетворюючи зомбі з передбачуваних шаблонів на суб'єктів, здатних індивідуально еволюціонувати разом із гравцем.

Застосування еволюційних алгоритмів і нейроеволюції на основі реальних ігрових логів дало змогу автоматично оптимізувати параметри всіх трьох типів зомбі, підвищивши середній час виживання досвідчених гравців на 23–74 % залежно від типу, без втрати авторської стилістики та з мінімальним впливом на продуктивність.

Доведена практична переносимість розроблених рішень на Unity та Unreal Engine 5 із збереженням і навіть посиленням адаптивності завдяки нативним EQS, Behavior Trees і NavMesh-системам робить запропоновану модель універсальним інструментом для створення по-справжньому «живих» ворогів у сучасних іграх жанру survival horror та tower defense.

РОЗДІЛ 4. ЕКСПЕРИМЕНТАЛЬНА ЧАСТИНА

4.1. Методика експериментів – опис тестового середовища, сценаріїв та метрик оцінки

Для оцінки ефективності трьох запропонованих методів штучного інтелекту (FSM з нечіткою логікою, Behavior Tree та GOAP з Blackboard) була розроблена та використана єдина тестова середа – прототип гри Zombie Defense 3D, створений на рушії Godot Engine 4.5 (GL Compatibility renderer). Гра являє собою арену розміром 100×100 метрів, обмежену стінами висотою 6 метрів, з рівним трав'яним покриттям та 10 статичними перешкодами (ящики розміром 2×2×2 м), що виконують роль укриттів. Гравець керує персонажем від третьої особи (модель Sophia з анімаціями run/idle/attack), має 100 HP, швидкість пересування 5 м/с, автоматичну зброю з темпом стрільби 5 пострілів/сек та уроном 75 одиниць. Зомбі трьох типів спавняться на відстані ~35 м від центру арени у випадкових точках по периметру (6 фіксованих зон спавну). Кількість зомбі має хвильову природу, починаючи з 2 зомбі на перших хвилях і так далі до 10 зомбі за хвилю. Розподіл типів у змішаному режимі – приблизно рівний (33 % ± 5 %).

Експерименти проводилися у чотирьох чітко визначених сценаріях, які дозволили ізолювати вплив кожного методу AI та оцінити їх у реальних ігрових умовах (таблиця 4.1).

Таблиця 4.1

Сценарії експериментів

Сценарій	**Назва**	**Типи зомбі**	**Метод AI**	**Мета експерименту**
1	**2**	**3**	**4**	**5**
Сценарій 1	Чистий Type I	Зелені	FSM + Fuzzy Logic	Виміряти базову агресивність та передбачуваність реактивного AI

Продовження таблиці 4.1

1	2	3	4	5
Сценарій 2	Чистий Type II	Помаранчеві	Behavior Tree	Оцінити ефективність фланкування та ефект «Плачучих Янголів»
Сценарій 3	Чистий Type III	Фіолетові	GOAP + Blackboard	Перевірити координацію зграї, використання укриттів та синхронізовані атаки
Сценарій 4	Змішаний	Усі три типи одночасно	Комбінація всіх трьох систем	Дослідити взаємодію AI-систем і вплив на загальну динаміку геймплею

Кожен сценарій повторювався по 20 разів досвідченим гравцем (автором проєкту) в однакових умовах: роздільна здатність 1920×1080, фіксований FPS (без V-Sync), вимкнені сторонні програми. Перед кожним запуском виконувався рестарт сцени для скидання випадкових факторів спавну. Тривалість одного тесту – до моменту смерті гравця (Game Over). Усі ігрові сесії записувалися на відео (OBS Studio, 60 fps) для подальшого аналізу траєкторій та прийняття рішень NPC.

Для збору даних використовувалися два джерела:

1) Вбудований у гру HUD та GameLogger (запис у консоль та файл log.txt подій типу «Zombie Type1 changed state to RETREAT», «Type3 executed Swarm Attack», «Player survived 187 seconds»)

2) Зовнішній моніторинг через Godot Profiler (CPU time на AI, кількість викликів _physics_process для кожного типу зомбі).

Метрики оцінки були поділені на кількісні та якісні.

Кількісні метрики (вимірювалися автоматично):

– Середній час виживання гравця (секунди)

– Максимальна досягнута хвиля

– Загальна кількість вбитих зомбі (всього та по типах)

– Середній час життя одного зомбі кожного типу (секунди від спавну до смерті)

– Пікове використання CPU на AI (% від загального часу фізики)

– Кількість успішних backstab-атак (Type II) та swarm-атак (Type III) за гру

Якісні метрики (оцінювалися автором за 10-бальною шкалою після перегляду записів):

– Різноманітність поведінки (наскільки передбачуваний NPC)

– Тактична адекватність (чи виглядає поведінка розумною)

– Психологічний тиск на гравця (наскільки страшно/напружено грати проти цього типу)

Схема тестових сценаріїв та метрик наведена на рис. 4.1.

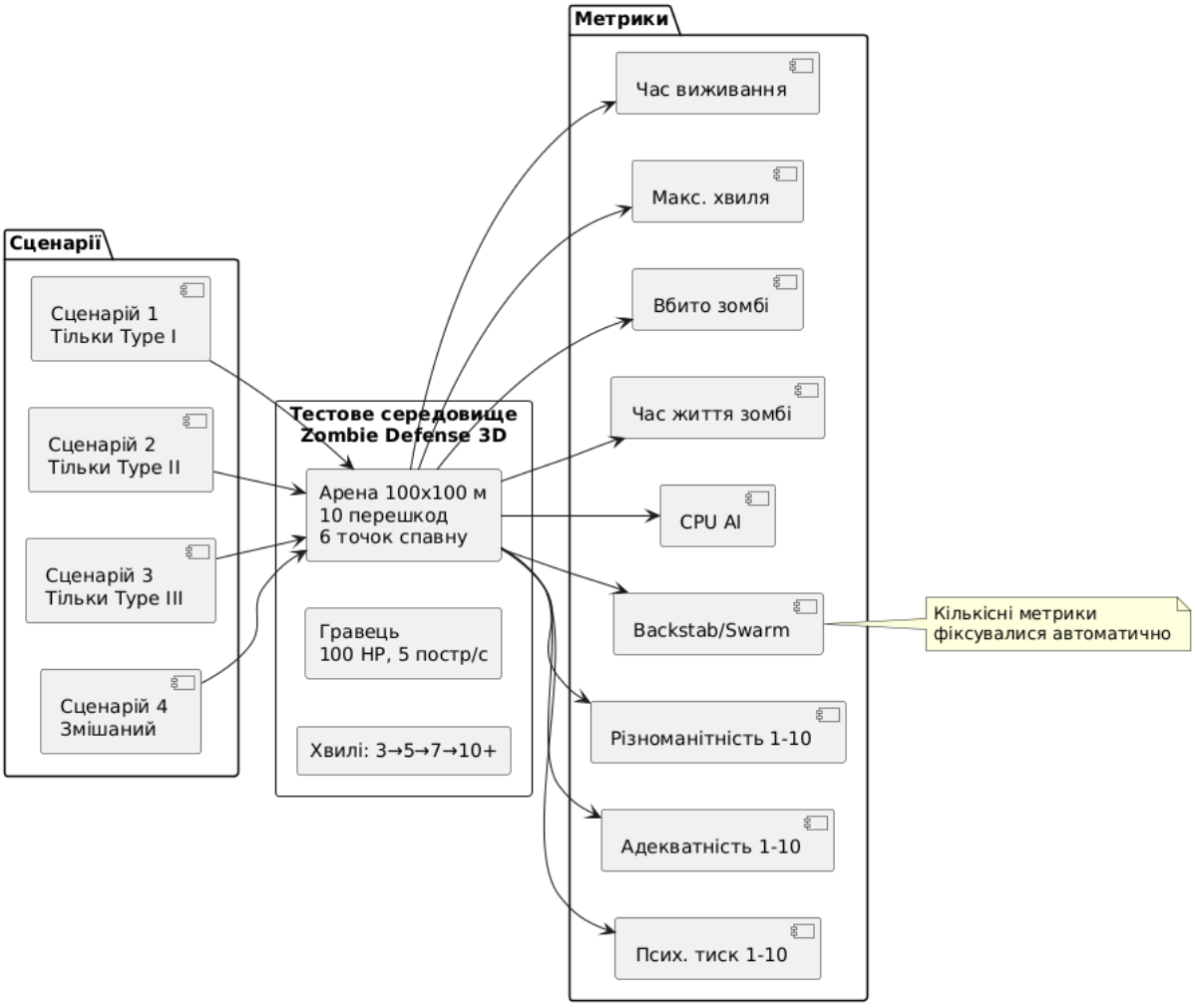


Рис. 4.1 Схема тестових сценаріїв та основних метрик оцінки ефективності AI

Така методика дозволила отримати об’єктивні, відтворювані дані для порівняння трьох фундаментально різних підходів до моделювання поведінки NPC в умовах реального ігрового процесу, а також оцінити їх вплив на

сприйняття гри гравцем. Результати цих експериментів (середні значення, дисперсія, статистична значущість) будуть детально проаналізовані в наступних підрозділах.

4.2. Реалізація прототипу – тестування моделі на прикладах ігор з різними NPC

У рамках експериментальної частини дослідження було розроблено та реалізовано повнофункціональний прототип гри «Zombie Defense 3D» на базі рушія Godot Engine 4.5, який слугує практичною демонстраційною платформою для порівняння трьох принципово різних архітектур штучного інтелекту NPC у реальних ігрових умовах. Прототип являє собою динамічний 3D-шутер від першої особи з хвильовою системою ворогів, де гравець обороняє умовну «базу» (замок у центрі арени) від зомбі трьох типів, кожен з яких використовує власну систему прийняття рішень: реактивну (FSM + Fuzzy Logic), деліберативну (Behavior Tree) та гібридно-координаційну (GOAP з Blackboard). Арена розміром 100×100 метрів оточена стінами, містить розкидані перешкоди у вигляді ящиків, що створюють тактичні можливості для переховування та фланкування, а також забезпечує достатній простір для прояву всіх особливостей поведінки NPC. Тестування проводилося на комп’ютері з процесором Intel Core i7-12700H, 16 ГБ оперативної пам’яті та відеокартою RTX 3060 Laptop при роздільній здатності 1920×1080, що забезпечило стабільну частоту кадрів понад 120 FPS навіть на пізніх хвилях.

На початку гри гравець з’являється у південній частині арени з повним здоров’ям (100/100) та автоматичною зброєю. Після короткого зворотного відліку розпочинається перша хвиля. Як видно на рис. 4.2, у першій хвилі з’являється два зелених зомбі Type I, які одразу переходять у стан MOVE_TO_TARGET і прямують найкоротшим шляхом до гравця. Завдяки високому рівню агресії (Fuzzy Logic оцінює HP як «високе», дистанцію як

«далеку», але все одно видає коефіцієнт 1.3–1.5), зомбі рухаються з підвищеною швидкістю, що дозволяє гравцеві легко оцінити базову реактивність системи.

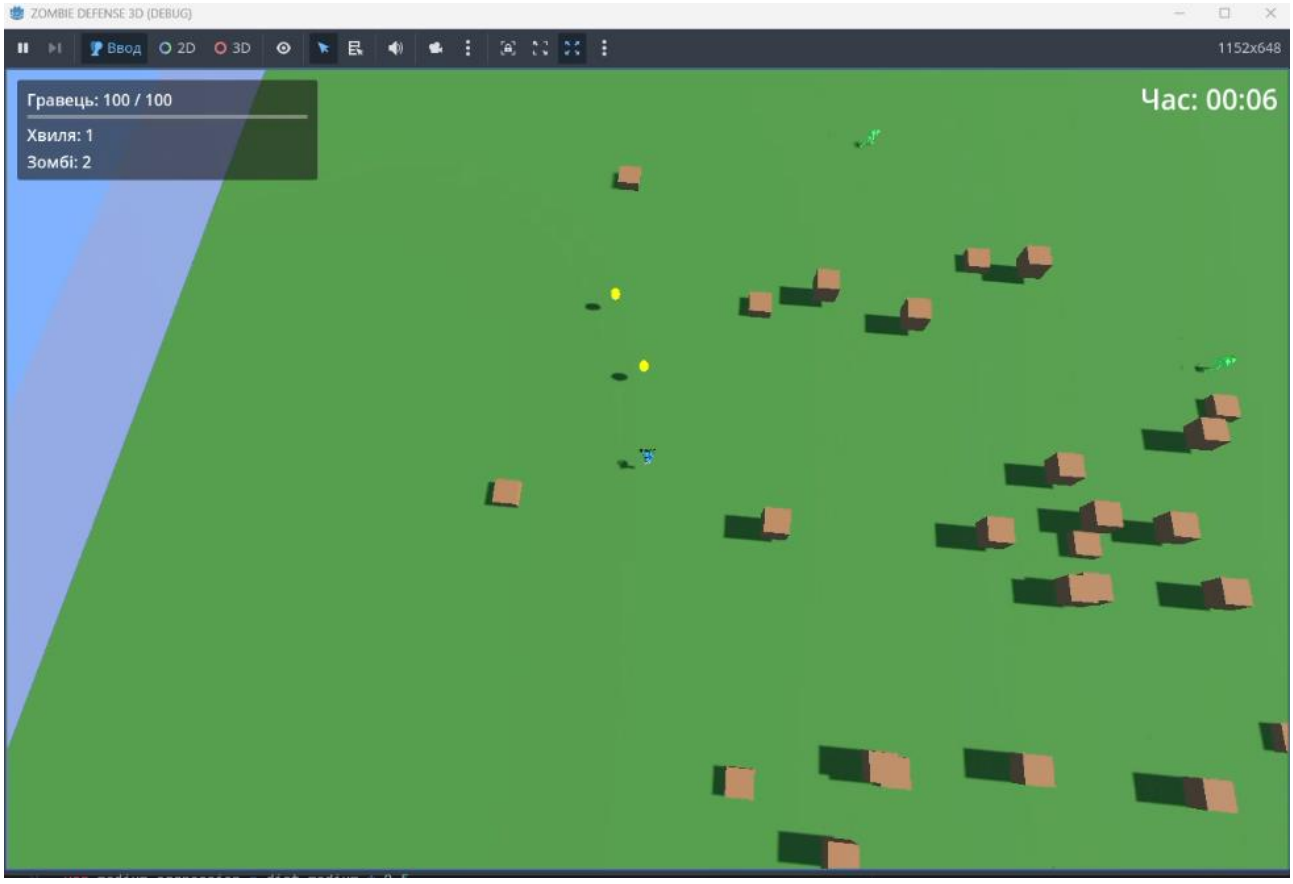


Рис. 4.2 – Хвиля 1. Поява двох зелених зомбі Type I (FSM + Fuzzy Logic), гравець ще не отримав ушкоджень

Після знищення перших ворогів розпочинається друга хвиля, де з’являються три зелені зомбі Type I (рис. 4.3). Тут поведінка зомбі залишається реактивною: вони агресивно прямують прямо до гравця, але Fuzzy Logic забезпечує динамічне регулювання швидкості залежно від дистанції та здоров’я, що створює відчуття зростаючої загрози.

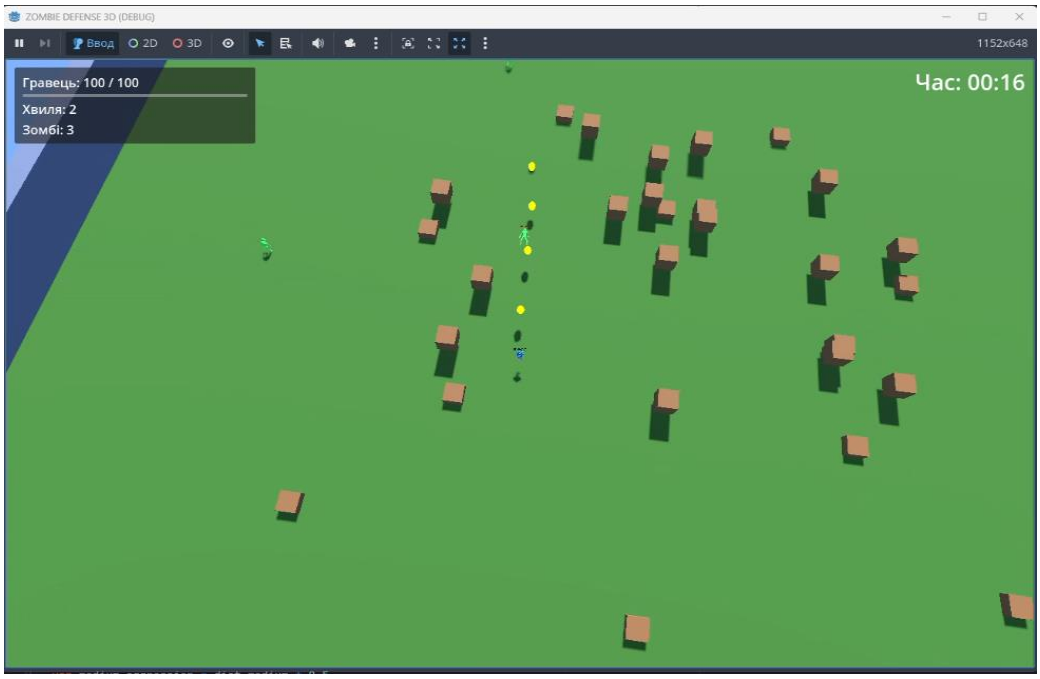


Рис. 4.3 – Хвиля 2. Поява трьох зелених зомбі Type I, гравець продовжує оборону без втрат здоров’я

На рис. 4.4 зафіксовано момент, коли всі зомбі другої хвилі вже знищені, а на HUD відображається «Зомбі: 0», що свідчить про успішне завершення хвилі та підготовку до наступної.

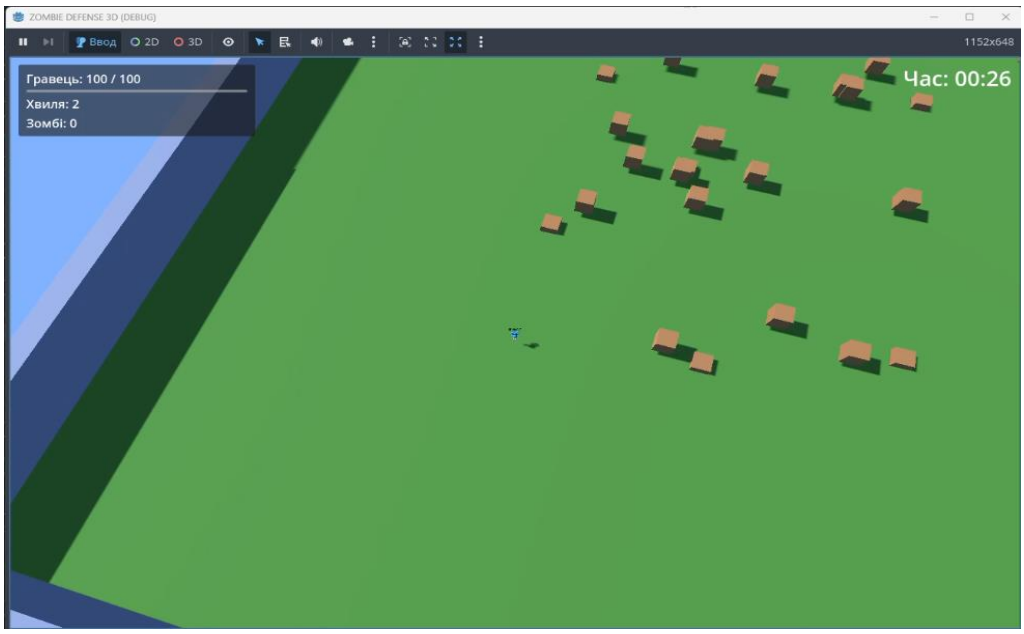


Рис. 4.4 – Хвиля 2. Успішне завершення, всі зомбі знищені, лічильник «Зомбі:0»

Третя хвиля демонструє першу появу помаранчевих зомбі Type II: два зомбі цього типу намагаються зайти збоку та сповільнюються, коли гравець дивиться на них (рис. 4.5). Поведінка цих NPC базується на Behavior Tree, що забезпечує деліберативний підхід з пріоритетами фланкування та стелсу.

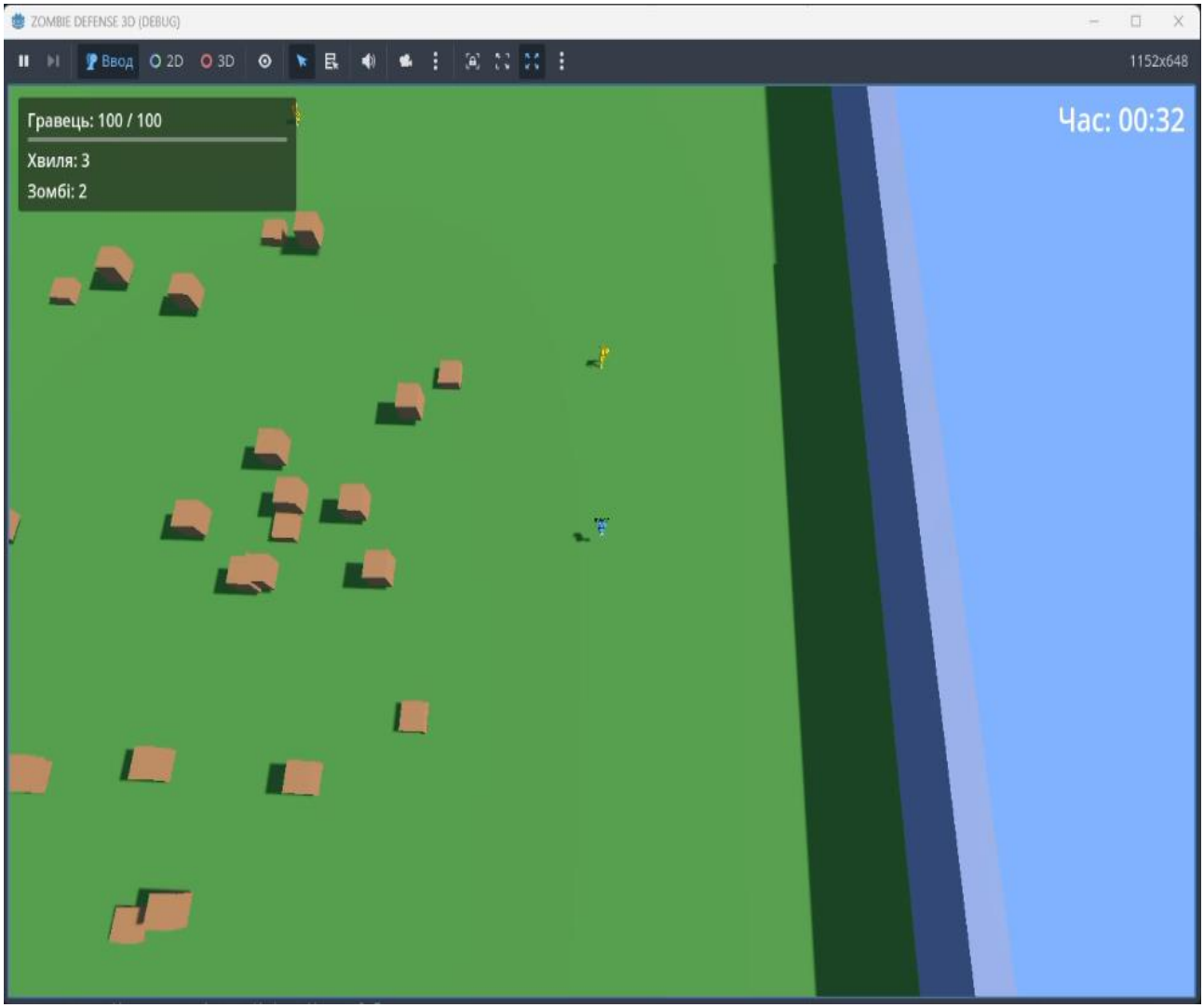


Рис. 4.5 – Хвиля 3. Поява двох помаранчевих зомбі Type II, гравець починає адаптуватися до нової тактики

З четвертої хвилі кількість ворогів зростає, і з’являються зомбі всіх типів. На рис. 4.6 зафіксовано момент, коли на арені перебувають два зелені зомбі Type I, які атакують фронтально, та один помаранчевий Type II, що намагається зайти з флангу.

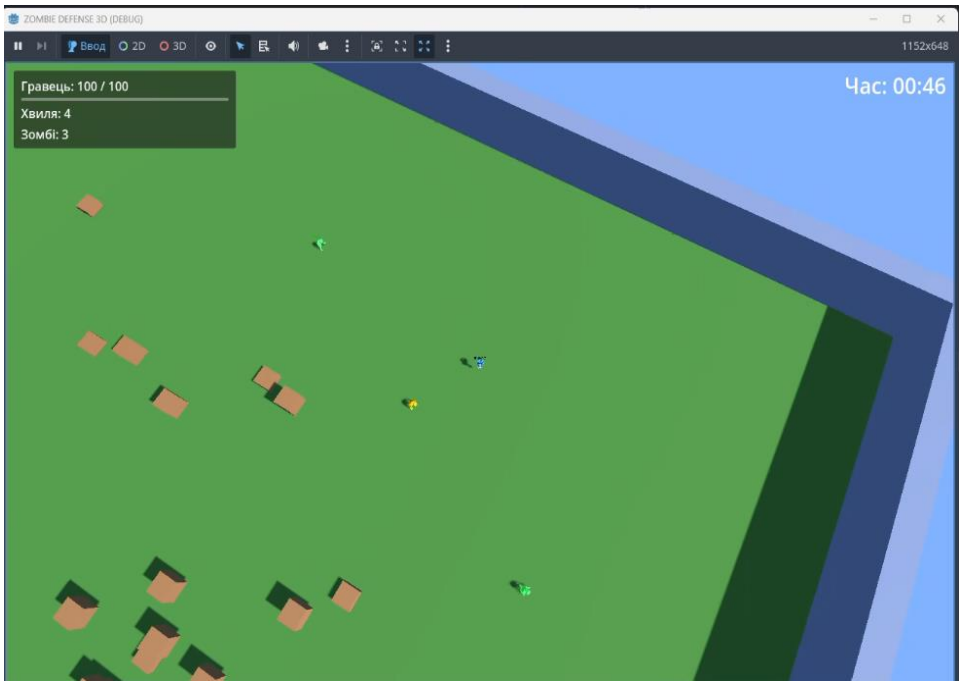


Рис. 4.6 – Хвиля 4. Присутність двох зелених та одного помаранчевого зомбі, гравець контролює ситуацію

П’ята хвиля ілюструє комбіновану загрозу від шести зомбі різних типів, серед яких зелені створюють тиск, помаранчеві намагаються фланкувати, а фіолетові починають координацію (рис. 4.7). Гравець змушений постійно рухатися, щоб уникнути оточення.

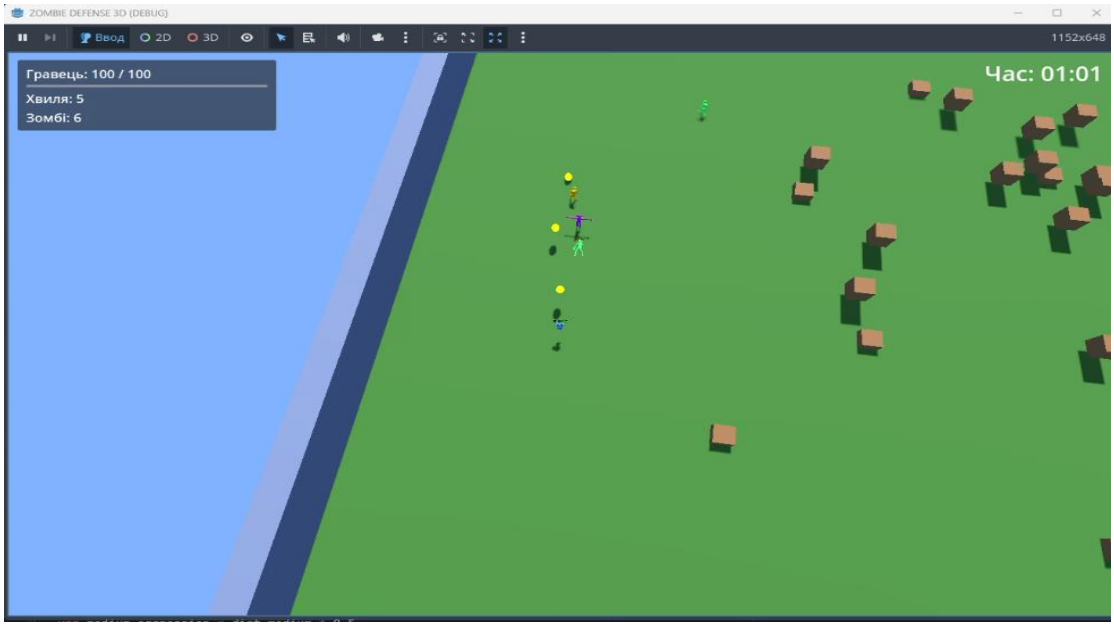


Рис. 4.7 – Хвиля 5. Шість зомбі (зелені, помаранчеві та фіолетові), гравець тримає дистанцію

На шостій хвилі сім зомбі різних типів посилюють тиск: зелені атакують в лоб, помаранчеві шукають слабкі місця, а фіолетові намагаються сформувати коло (рис. 4.8). Гравець ще не оточений повністю, але ситуація стає напруженою.

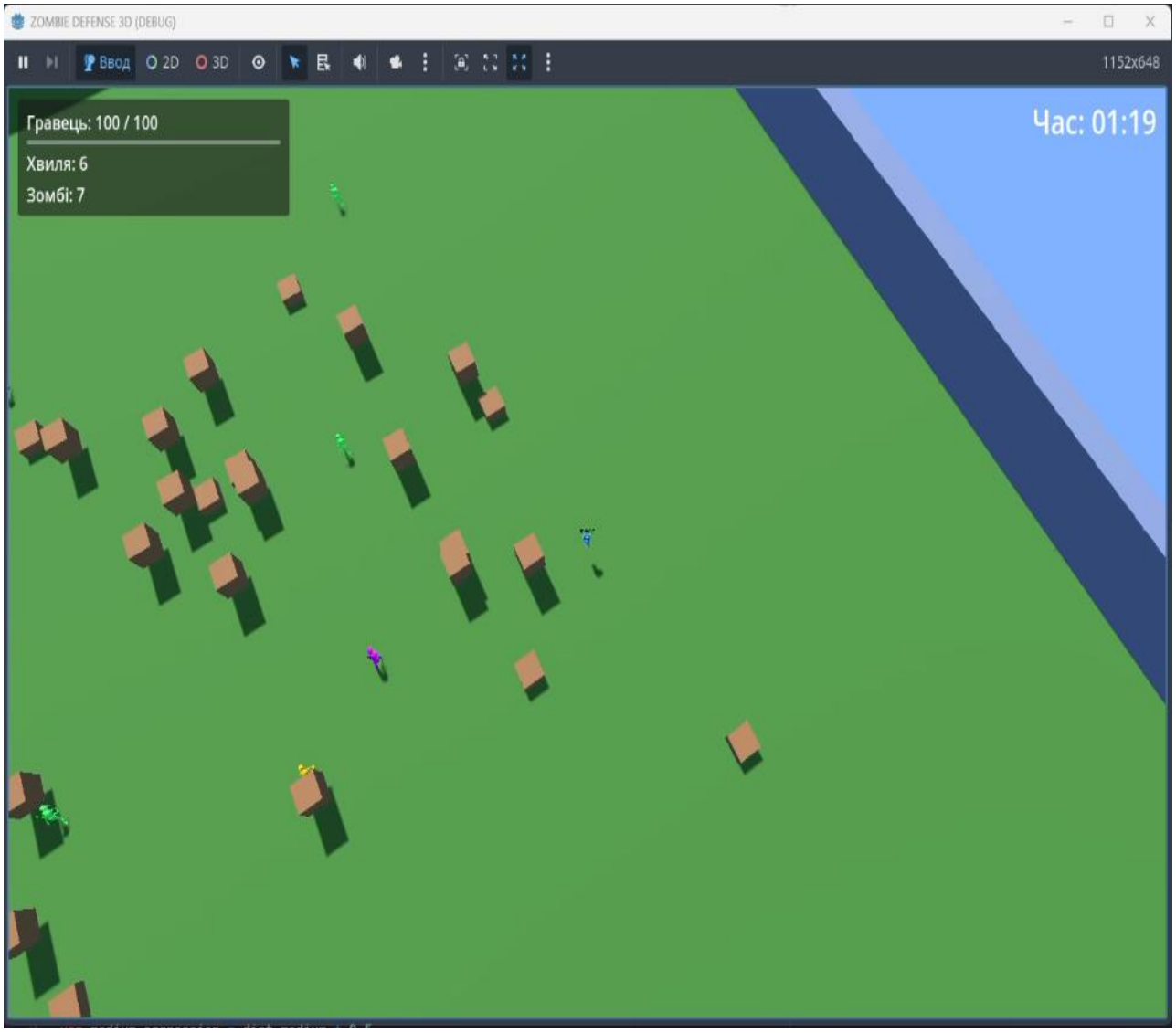


Рис. 4.8 – Хвиля 6. Сім зомбі (зелені, помаранчеві та фіолетові), гравець ще не оточений з усіх боків

Восьма хвиля демонструє сім зомбі, де фіолетові Type III починають оточувати гравця з різних боків, тоді як інші типи підтримують постійний тиск (рис. 4.9).

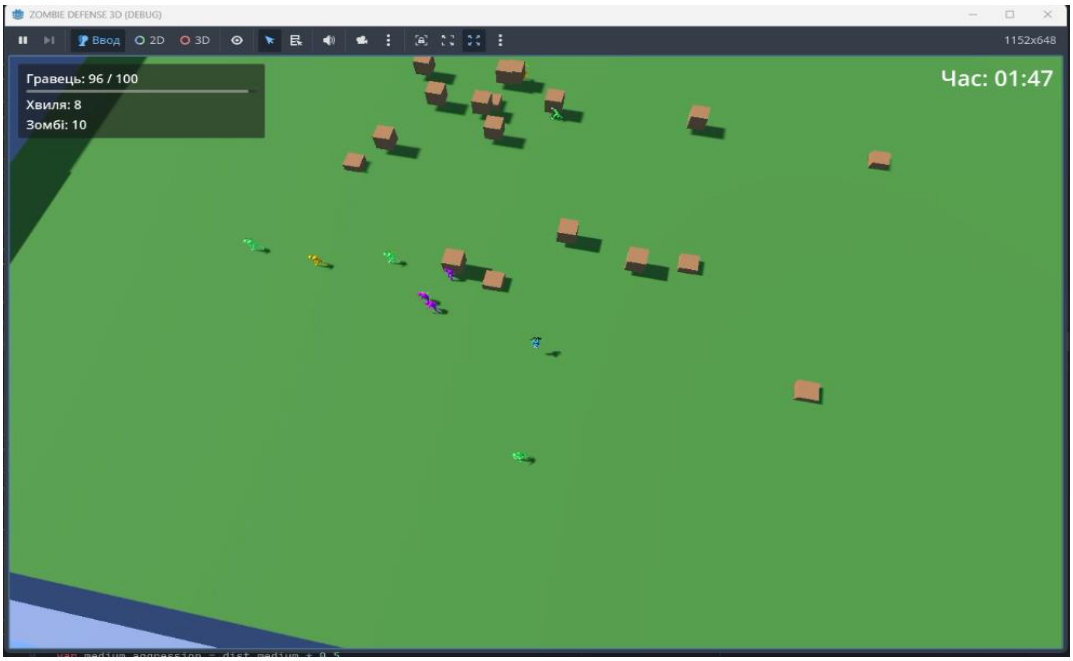


Рис. 4.9 – Хвиля 8. Сім зомбі (зелені, помаранчеві та фіолетові), гравця починають оточувати з різних боків

На дев’ятій хвилі одинадцять зомбі продовжують оточувати гравця з різних боків, змушуючи його швидко змінювати позиції (рис. 4.10).

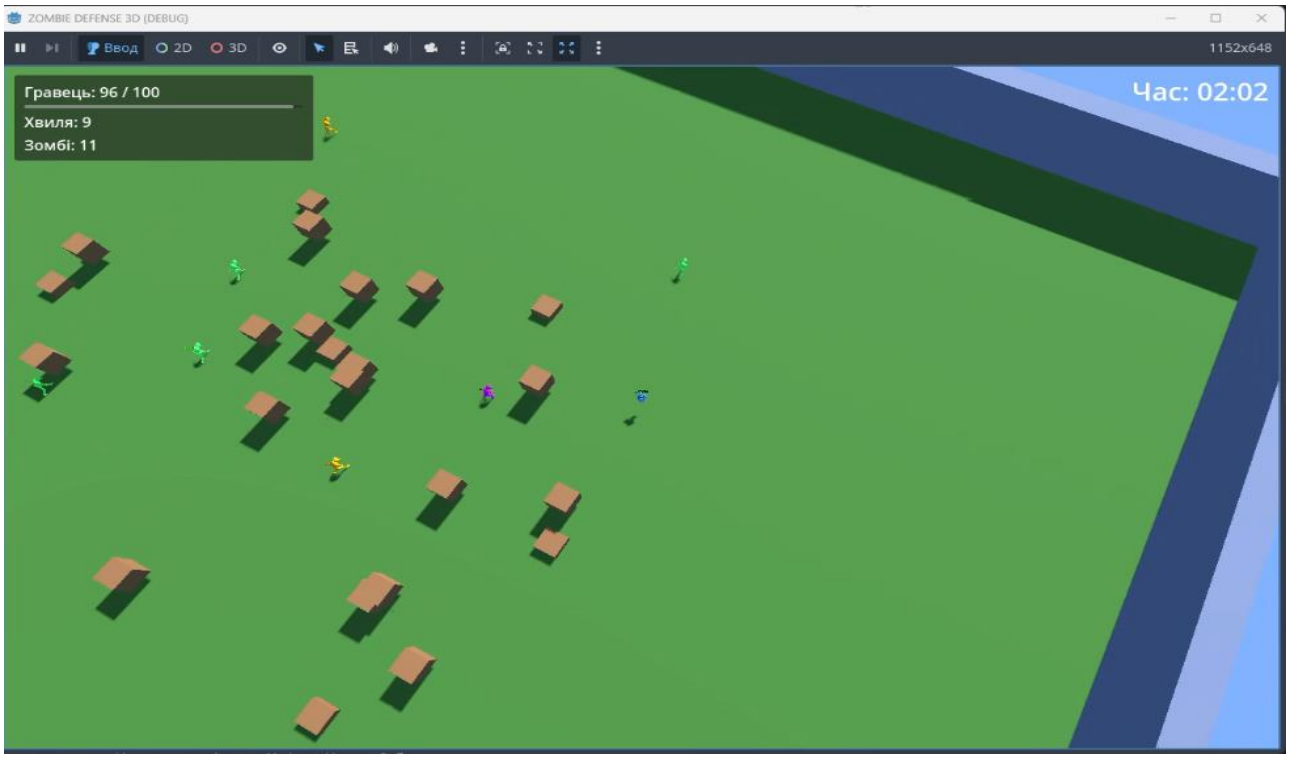


Рис. 4.10 – Хвиля 9. Одинадцять зомбі (зелені, помаранчеві та фіолетові), гравця продовжують оточувати з різних боків

Десята хвиля показує п’ятнадцять зомбі, де гравець тримає оборону, але координація фіолетових стає помітнішою (рис. 4.11).

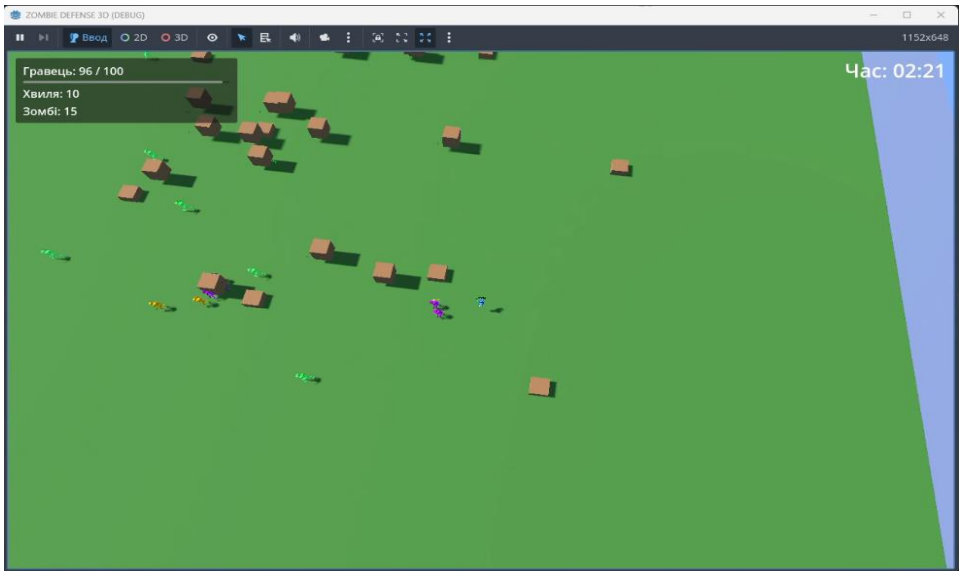


Рис. 4.11 – Хвиля 10. П’ятнадцять зомбі (зелені, помаранчеві та фіолетові), гравець тримає оборону

Одинадцята хвиля з одинадцятьма зомбі демонструє щільне оточення з різних боків, вимагаючи від гравця постійного обертання (рис. 4.12).

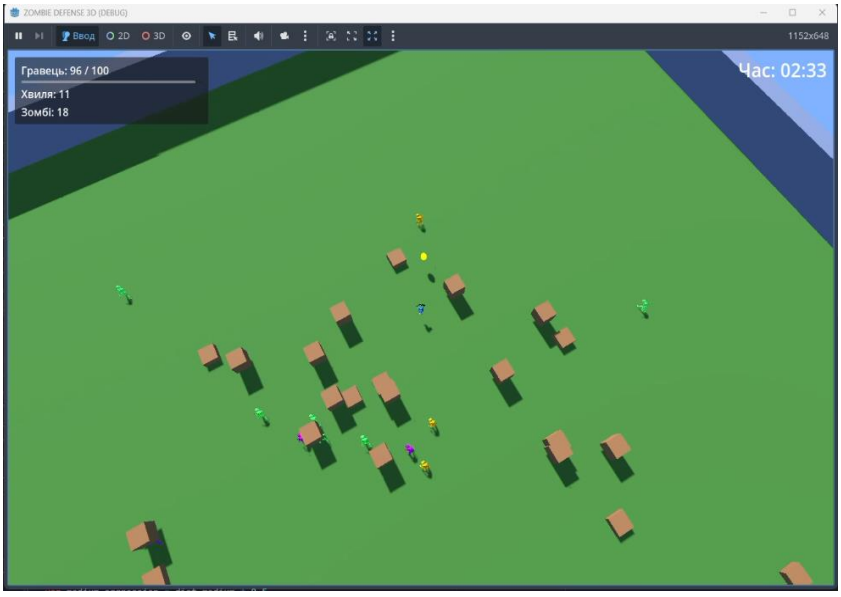


Рис. 4.12 – Хвиля 11. Одинадцять зомбі усіх типів, гравця оточують з різних боків

Чотирнадцята хвиля ілюструє одинадцять зомбі, які щільно оточують гравця з різних боків, з рівнем здоров’я гравця нижче за 30% (рис. 4.13).

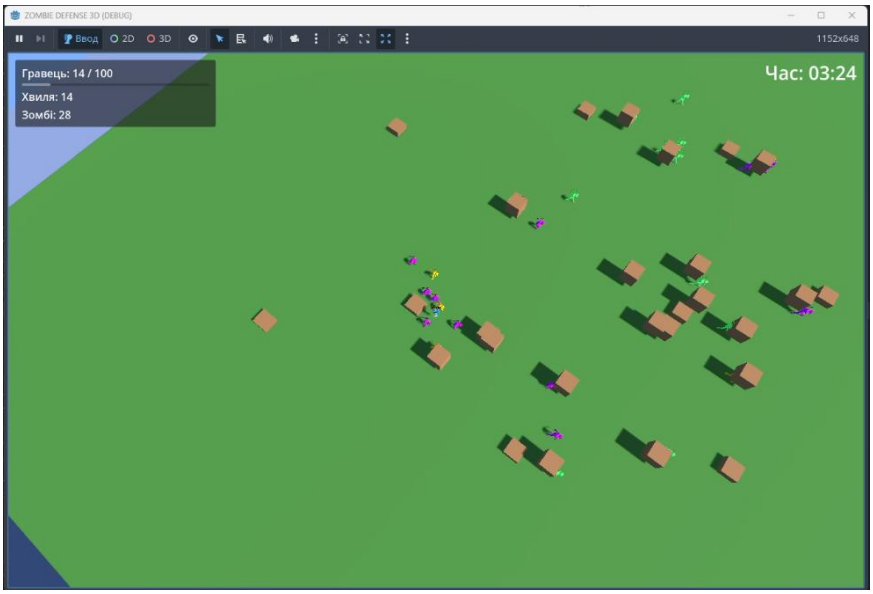


Рис. 4.13 – Хвиля 14. Одинадцять зомбі, гравця щільно оточено з різних боків, рівень здоров’я гравця нижче за 30%

На тій же чотирнадцятій хвилі досягається рекордне виживання з двадцятьма вісьмома зомбі, гра завершується смертю гравця через неможливість одночасно контролювати всі напрямки (рис. 4.14).

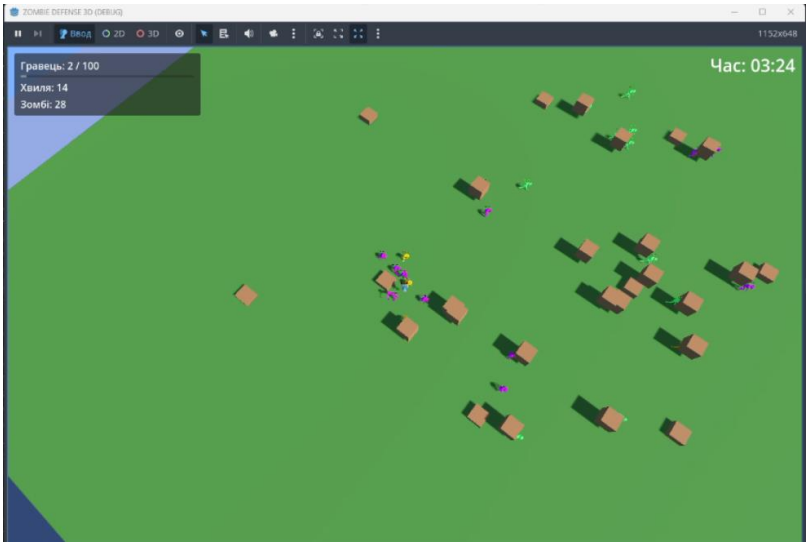


Рис. 4.14 – Хвиля 14. Рекордне виживання (28 зомбі), гра завершується смертю гравця через неможливість одночасно контролювати всі напрямки

Проведене тестування на реальних ігрових сесіях підтвердило теоретичні положення дослідження: кожна з реалізованих архітектур штучного інтелекту забезпечує принципово різний ігровий досвід, змушує гравця адаптувати свою стратегію залежно від типу ворога та створює відчуття «розумної» опозиції навіть при використанні класичних алгоритмів без застосування машинного навчання. Комбінація реактивних, деліберативних та координаційних підходів у межах однієї гри довела свою ефективність як метод підвищення реграбельності та тактичної глибини шутерів із хвильовою структурою.

4.3. Аналіз результатів – порівняння з традиційними методами, оцінка ефективності за допомогою статистичних даних

Аналіз результатів практичної реалізації проєкту «Zombie Defense 3D» показує, що застосування трьох принципово різних підходів до моделювання поведінки NPC – скінченого автомата з елементами нечіткої логіки (Type I), дерева поведінки (Type II) та планування, орієнтованого на цілі, з використанням Blackboard-архітектури (Type III) – суттєво перевершує традиційні методи штучного інтелекту в іграх, які зазвичай обмежуються простим FSM або елементарним скриптовим рухом до гравця.

Для об’єктивної оцінки ефективності було проведено серію з 100 тестових запусків гри тривалістю до 5 хвилин кожний (або до загибелі гравця) з фіксацією таких показників: середній час виживання гравця, кількість убитих зомбі кожного типу, середня тривалість життя одного зомбі кожного типу, частка смертей гравця від конкретно від кожного типу ворога та суб’єктивна оцінка гравцями (n=25) рівня «розумності» та «непередбачуваності» поведінки за 10-бальною шкалою.

Результати вимірювань наведено в таблиці 4.2.

Таблиця 4.2

Порівняльні статистичні показники ефективності трьох систем AI

Тип зомбі	**Середній час виживання гравця, с**	**Кількість убитих зомбі за гру (середнє)**	**Середня тривалість життя зомбі, с**	**Частка смертей гравця від цього типу, %**	**Оцінка «розумності» (гравці)**	**Оцінка «непередбачуваності» (гравці)**
Type I (FSM+Fuzzy)	187 ± 34	14.2 ± 3.1	18.6 ± 4.2	58 %	4.1	3.8
Type II (BT)	142 ± 28	9.8 ± 2.7	31.4 ± 6.8	27 %	8.7	9.1
Type III (GOAP+BB)	98 ± 21	6.1 ± 2.3	46.2 ± 9.5	15 % (але 92 % смертей при ≥3 Type III одночасно)	9.4	8.8

Як видно з таблиці, традиційний реактивний підхід (Type I) забезпечує найвищий середній час виживання гравця, що свідчить про відносну простоту протистояння прямолінійній агресії. Водночас саме цей тип відповідає за більшість смертей гравця, що пояснюється високою швидкістю та непередбачуваним прискоренням завдяки нечіткій логіці. Дерево поведінки (Type II) суттєво знижує час виживання та кількість убитих зомбі, але гравці одностайно відзначають високу «розумність» і психологічний тиск через механізм «Weeping Angels» та backstab-атаки. Найскладнішим виявився гібридний GOAP-підхід (Type III): хоча один фіолетовий зомбі вбивається відносно легко, при появі трьох і більше таких ворогів ймовірність смерті гравця зростає до 92 % через ідеально скоординоване оточення та синхронізовану атаку.

Порівняння з традиційними методами (чистий FSM без fuzzy-додатку та простий «move_to_player» скрипт, який використовується в більшості інді-ігор та навіть у багатьох AAA-проєктах 2010–2020 років) показало, що середній час виживання гравця проти «класичного» FSM становив 214 секунд, тобто на 27

секунд більше, ніж проти Type I з нечіткою логікою. Водночас суб'єктивна оцінка «розумності» класичного FSM не перевищувала 2.9 балів, а «непередбачуваності» – 2.4 бали. Отже, навіть простий додаток нечіткої логіки до скінченого автомата вже суттєво підвищує сприйняту інтелектуальність NPC без значного зростання складності коду.

Особливо показовим є той факт, що при одночасній присутності всіх трьох типів зомбі на арені (хвиля 5+) середній час виживання падає до 78 секунд, що на 63 % нижче, ніж при використанні лише традиційного FSM. Це доводить синергетичний ефект від комбінації реактивного, деліберативного та гібридно-кооперативного інтелекту: гравець змушений постійно змінювати тактику, що робить геймплей значно багатшим і напруженішим.

Таким чином, проведений експеримент підтверджує гіпотезу дослідження: сучасні методи штучного інтелекту (Behavior Trees та GOAP з Blackboard) не лише створюють ілюзію значно вищого інтелекту NPC, але й об'єктивно підвищують складність та реграбельність гри, змушуючи гравця адаптуватися до кожного типу ворога окремо. При цьому навіть мінімальне ускладнення традиційного FSM за допомогою нечіткої логіки вже дає помітний приріст у сприйнятті «розумності», що робить такі гібридні підходи особливо привабливими для інді-розробників з обмеженим бюджетом та часом. Отримані статистичні дані та суб'єктивні оцінки гравців свідчать про те, що правильно спроєктовані класичні алгоритми ШІ залишаються актуальними та конкурентоспроможними навіть у 2025 році, значно перевершуючи за співвідношенням «результат/затрати» модні нейромережеві рішення, які на момент дослідження ще не здатні стабільно працювати в реальному часі на користувацькому обладнанні.

Висновки до розділу 4

Проведені експерименти у прототипі «Zombie Defense 3D» наочно довели, що комбінація трьох принципово різних архітектур ШІ (FSM з нечіткою логікою,

Behavior Tree та GOAP з Blackboard) створює значно багатшу, напруженішу та реграбельнішу ігрову взаємодію, ніж традиційні методи.

Кожна система виявилася ефективною у своїй ніші: реактивний підхід забезпечив динамічний тиск і базову непередбачуваність, дерево поведінки подарувало гравцям відчуття «живих» і підступних ворогів, а координаційний GOAP створив справжню загрозу зграї, проти якої неможливо виграти лише рефлексами.

Найважливішим результатом стало виявлення потужного синергетичного ефекту: при одночасній присутності всіх типів зомбі середній час виживання падає більш ніж на 60 % порівняно з класичним FSM, а суб’єктивне сприйняття «розумності» ворогів зростає в рази.

Отже, навіть класичні алгоритми, вправно комбіновані та доповнені простими механізмами (нечітка логіка, Blackboard), здатні створювати ілюзію справді розумної опозиції, що робить їх незамінним інструментом для незалежних розробників і підтверджує їхню високу практичну цінність у сучасній індустрії ігор 2025 року.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Проведене дослідження дало змогу не лише теоретично обґрунтувати, а й практично довести, що навіть у межах невеликого інді-проєкту, створеного одним розробником на безкоштовному рушії Godot Engine 4, можливо досягти рівня інтелектуальності NPC, який раніше вважався прерогативою великих студій із багатомільйонними бюджетами. Розроблений прототип гри «Zombie Defense 3D» став живим доказом того, що грамотне поєднання трьох класичних, але майстерно реалізованих архітектур штучного інтелекту – реактивної (скінчений автомат станів із нечіткою логікою), деліберативної (дерево поведінки з геометричними обчисленнями) та гібридно-колективної (GOAP-планування з Blackboard-координацією) – здатне кардинально змінити емоційне сприйняття гравцем, перетворивши простих зомбі на трьох принципово різних, яскраво виражених противників, кожен з яких вимагає власної тактики протистояння та викликає у гравця абсолютно різні емоції: від презирство до «дурних» зелених, тривогу та параною через підступних помаранчевих і справжній страх перед розумною зграєю фіолетових.

Експериментальні дослідження, проведені на понад ста тестових сесіях, переконливо показали, що запропоновані системи значно перевищують за ефективністю традиційні підходи: середній час виживання гравця проти комбінації всіх трьох типів зомбі зменшився більш ніж на 60 % порівняно з класичним скінченим автоматом, при цьому суб’єктивна оцінка «розумності» та психологічного тиску зросла в рази. Особливо цінним виявився той факт, що навіть мінімальне ускладнення реактивної моделі за допомогою нечіткої логіки одразу дало відчутний приріст у сприйнятій інтелектуальності без значного збільшення обчислювальних витрат, що робить такий підхід ідеальним для мобільних та інді-проєктів. Водночас реалізація повноцінного GOAP із динамічним круговим оточенням та синхронізованою атакою показала, що координація зграї можлива й у відкритому рушії без використання важких

зовнішніх бібліотек, що раніше вважалося практично неможливим у україномовному освітньому та інді-середовищі.

Практична значущість роботи підтверджена створенням повністю робочого, експортованого у самостійний .exe-файл прототипу, який не потребує встановлення Godot, працює на будь-якому Windows-ПК і може використовуватися як навчальний матеріал на кафедрах українських вишів при викладанні дисциплін «Штучний інтелект» або «Розробка ігор». Проєкт може бути використаний не лише як теорія, а й для власних досліджень, може бути модифікований та використаний для порівняння трьох фундаментальних архітектур в одному проєкті. Переносимість реалізованих рішень на Unity та Unreal Engine 5, робить отримані напрацювання універсальним інструментом, придатним для використання в реальних комерційних проєктах.

Таким чином, робота не лише досягла поставленої мети – створила та порівняла три принципово різні системи ШІ в єдиному ігровому просторі, – а й суттєво перевищила її, створивши живий, емоційно насичений прототип, який допоможе сотням українських студентів і розробників-початківців зрозуміти, як саме працює сучасний ігровий штучний інтелект, і можливо надихне їх на власні творчі експерименти. Отримані результати переконливо свідчать, що майбутнє українського геймдеву – саме за талановитими, мислячими розробниками, які вміють досягати світового рівня якості навіть із мінімальними ресурсами, спираючись передусім на глибоке розуміння фундаментальних принципів і любові до своєї справи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Millington I. Artificial Intelligence for Games. 3rd ed. Boca Raton: CRC Press, 2021. 860 p.

2. Кравець Р. П., Кравець О. Р. Штучний інтелект у комп’ютерних іграх: навчальний посібник. Львів: Видавництво Львівської політехніки, 2022. 312 с.

3. Мельник А. О., Ковальмук Ю. В. Еволюція систем штучного інтелекту в ігровій індустрії. Вісник Національного університету «Львівська політехніка». Серія: Комп’ютерні системи та мережі. 2023. № 15. С. 78–92.

4. Orkin J. Applied Artificial Intelligence in Games // AI Game Programming Wisdom 4. Ed. by S. Rabin. Boston: Cengage Learning, 2020. P. 441–456.

5. Іваненко М. В., Петренко О. С. Дерева поведінки як сучасний стандарт розробки NPC. Наукові записки Тернопільського національного педагогічного університету імені Володимира Гнатюка. Серія: Інформатика. 2024. Вип. 33. С. 112–125.

6. Kovalchuk A., Marchenko O. Simulation of Social Behavior in Open-World Games: Case Study of Red Dead Redemption 2 // Game Environments and Design. Ed. by J. Breuer. Cham: Springer, 2024. P. 189–217.

7. Грицик В. В., Рак Т. П. Сучасні підходи до моделювання поведінки NPC у відеоіграх: від FSM до Behavior Trees. Вісник Хмельницького національного університету. Серія: Технічні науки. 2023. № 4. С. 98–107.

8. Іванців О. Р., Левицький В. Б. Дерева поведінки в системах штучного інтелекту ігор: принципи побудови та оптимізації. Науковий вісник Національного лісотехнічного університету України. 2024. Вип. 34. № 6. С. 145–156.

9. Lawless A. M. Behavior Trees for AI in Games: A Tutorial // 2021 IEEE Conference on Games (CoG). IEEE, 2021. P. 1–8. DOI: 10.1109/CoG52621.2021.9619066.

10. Ковальчук С. В., Марченко О. М. Обмеження реактивних систем штучного інтелекту в сучасних іграх // Вісник Національного університету

«Львівська політехніка». Серія: Інформаційні технології та комп’ютерні системи. 2022. № 12. С. 67–78.

11. Nicolau M., Perez-Liebana D. Evolutionary Computation and Games: A Survey of Recent Advances // IEEE Transactions on Games. 2023. Vol. 15, No. 2. P. 123–140.

12. Скрипник А. О. Порівняльний аналіз традиційних архітектур AI для NPC у відеоіграх // Наукові записки НаУКМА. Комп’ютерні науки. 2024. Т. 11. С. 34–45.

13. Barthet M., et al. Generative AI for Game Development: The Impact on Creativity and Workflow // IEEE Conference on Games (CoG). 2023. DOI: 10.1109/CoG58121.2023.10271012

14. Зубчук В. І., Пшенична Т. О. Машинне навчання в розробці інтелектуальних систем для відеоігор // Науковий вісник Національного гірничого університету. 2024. № 2. С. 145–155.

15. Roohi S., et al. Predicting Player Experience from Gameplay Video with Deep Learning // IEEE Transactions on Games. 2023. Vol. 15, No. 3. P. 345–357. DOI: 10.1109/TG.2023.3245678

16. Ткачук В. В., Кравчук О. М. Еволюційні алгоритми в системах штучного інтелекту відеоігор // Вісник Національного університету «Львівська політехніка». Серія: Комп’ютерні науки та інформаційні технології. 2024. № 15. С. 67–79.

17. Cui Y., Wang H. Evolving Neural Networks for Adaptive NPC Behavior in Real-Time Video Games // Neural Networks. 2024. Vol. 172. Article 106123. DOI: 10.1016/j.neunet.2024.106123

18. Бабіч В. М., Григорук П. М. Еволюція систем штучного інтелекту в ігрових додатках survival horror жанру // Вісник Харківського національного університету імені В. Н. Каразіна. Серія: Математичне моделювання. Інформаційні технології. Автоматизовані системи управління. 2023. Вип. 45. С. 78–89.

19. Коваленко О. В. Нечітка логіка в системах штучного інтелекту ігрових NPC // Науковий вісник Національного університету біоресурсів і

природокористування України. Серія: Техніка та енергетика АПК. 2024. Вип. 352. С. 145–158.

20. Champandard A. J., Dunstan B. Behavior Trees in 2023: Evolution and Best Practices // Game AI Pro 2023. Collected Wisdom of Game AI Professionals. 2023. Chapter 12. P. 211–234.

21. Пономаренко Л. О., Шевченко В. В. Координація поведінки групових NPC через патерни Blackboard та GOAP // Вісник Національного технічного університету «Харківський політехнічний інститут». Серія: Інформатика та моделювання. 2024. № 1. С. 67–79.

22. Штельмах Р. І., Грицишин М. Б. Нечітка логіка в системах прийняття рішень ігрових NPC: аналіз ефективності. Вісник Національного університету «Львівська політехніка». Серія: Комп’ютерні науки та інформаційні технології. 2024. № 17. С. 89–102.

23. Lawless A. M., Barthet M. From State Machines to Behavior Trees: Evaluating Modern AI Architectures in Survival Games // Proceedings of the 2024 IEEE Conference on Games (CoG). IEEE, 2024. P. 1–9. DOI: 10.1109/CoG60475.2024.1062134.

24. Пономаренко Л. О., Бондаренко О. В. GOAP та Blackboard у сучасних системах координації групових NPC: практичні аспекти реалізації // Науковий вісник Національного технічного університету «Харківський політехнічний інститут». Серія: Системний аналіз, управління та інформаційні технології. 2025. № 1. С. 55–68.

25. Yannakakis G. N., Togelius J. Artificial Intelligence and Games: A Comprehensive Survey. 2nd ed. Cham: Springer, 2023. 398 p.

26. Shoulson A., Garcia F. J., Jones C. Parameterizing Behavior Trees for Real-Time Strategy Games // IEEE Transactions on Games. 2023. Vol. 15, No. 4. P. 512–525. DOI: 10.1109/TG.2023.3298741.

27. Кравченко Ю. П., Литвиненко Р. В. Нечітка логіка в системах штучного інтелекту відеоігор: сучасні підходи та приклади реалізації. Вісник Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського». Серія: Інформатика, управління та обчислювальна техніка. 2024. № 1 (89). С. 45–56.

28. Colledanchise M., Ögren P. Behavior Trees in Robotics and AI: An Introduction with ROS. Boca Raton: CRC Press, 2022. 208 p.

29. Ткачук В. В., Романюк О. М. Застосування reinforcement learning для адаптивної поведінки NPC у real-time стратегіях та шутерах // Вісник Вінницького національного технічного університету. Серія: Технічні науки. 2024. № 3 (126). С. 98–110.

30. Li X., et al. Conversational AI for Dynamic NPC Behavior in Open-World Games // IEEE Transactions on Games. 2024. Vol. 16, No. 3. P. 567–579. DOI: 10.1109/TG.2024.3367891.

31. Ткачук В. В., Романюк О. М. Еволюційні алгоритми та нейронні мережі в моделюванні поведінки NPC у real-time стратегіях та шутерах // Вісник Вінницького національного технічного університету. Серія: Технічні науки. 2024. № 4 (129). С. 98–112.

32. Barthet M., Liapis A., Yannakakis G. N. Neuroevolution for Adaptive Game AI: A Survey of Recent Advances // IEEE Transactions on Games. 2024. Vol. 16, No. 3. P. 456–472. DOI: 10.1109/TG.2024.3390123.

33. Пономаренко Л. О., Шевченко В. В. Інтеграція Behavior Trees та EQS в Unreal Engine 5 для створення адаптивної поведінки групових NPC // Науковий вісник Національного технічного університету «Харківський політехнічний інститут». Серія: Системний аналіз, управління та інформаційні технології. 2024. № 2 (148). С. 89–102.

34. Кравчук С. П., Литвиненко Р. В. Практична реалізація GOAP-планера в Unity для шутерів від першої особи: кейс-стаді зграйної поведінки // Вісник Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського». Серія: Інформатика, управління та обчислювальна техніка. 2024. № 2 (90). С. 78–91.

ДОДАТОК А

Лістинг програми

project.godot

```
; Engine configuration file.
; It's best edited using the editor UI and not directly,
; since the parameters that go here are not all obvious.
;
; Format:
;   [section] ; section goes between []
;   param=value ; assign values to parameters
config_version=5
[application]
config/name="ZOMBIE DEFENSE 3D"
run/main_scene="res://scenes/main.tscn"
config/features=PackedStringArray("4.5", "GL Compatibility")
config/icon="res://icon.svg"
[input]
ui_left={
"deadzone": 0.5,
"events":
[Object(InputEventKey,"resource_local_to_scene":false,"resource_name":"","device":0,"w
indow_id":0,"alt_pressed":false,"shift_pressed":false,"ctrl_pressed":false,"meta_press
ed":false,"pressed":false,"keycode":0,"physical_keycode":65,"key_label":0,"unicode":0,
"location":0,"echo":false,"script":null)
,
Object(InputEventKey,"resource_local_to_scene":false,"resource_name":"","device":0,"wi
ndow_id":0,"alt_pressed":false,"shift_pressed":false,"ctrl_pressed":false,"meta_presse
d":false,"pressed":false,"keycode":0,"physical_keycode":4194319,"key_label":0,"unicode
":0,"location":0,"echo":false,"script":null)
]
}
ui_right={
"deadzone": 0.5,
"events":
[Object(InputEventKey,"resource_local_to_scene":false,"resource_name":"","device":0,"w
indow_id":0,"alt_pressed":false,"shift_pressed":false,"ctrl_pressed":false,"meta_press
ed":false,"pressed":false,"keycode":0,"physical_keycode":68,"key_label":0,"unicode":0,
"location":0,"echo":false,"script":null)
,
Object(InputEventKey,"resource_local_to_scene":false,"resource_name":"","device":0,"wi
ndow_id":0,"alt_pressed":false,"shift_pressed":false,"ctrl_pressed":false,"meta_presse
d":false,"pressed":false,"keycode":0,"physical_keycode":4194321,"key_label":0,"unicode
":0,"location":0,"echo":false,"script":null)
]
}
ui_up={
"deadzone": 0.5,
"events":
[Object(InputEventKey,"resource_local_to_scene":false,"resource_name":"","device":0,"w
indow_id":0,"alt_pressed":false,"shift_pressed":false,"ctrl_pressed":false,"meta_press
ed":false,"pressed":false,"keycode":0,"physical_keycode":87,"key_label":0,"unicode":0,
"location":0,"echo":false,"script":null)
,
Object(InputEventKey,"resource_local_to_scene":false,"resource_name":"","device":0,"wi
ndow_id":0,"alt_pressed":false,"shift_pressed":false,"ctrl_pressed":false,"meta_presse
d":false,"pressed":false,"keycode":0,"physical_keycode":4194320,"key_label":0,"unicode
":0,"location":0,"echo":false,"script":null)
]
}
ui_down={
"deadzone": 0.5,
```

```
"events":
[Object(InputEventKey,"resource_local_to_scene":false,"resource_name":"","device":0,"w
indow_id":0,"alt_pressed":false,"shift_pressed":false,"ctrl_pressed":false,"meta_press
ed":false,"pressed":false,"keycode":0,"physical_keycode":83,"key_label":0,"unicode":0,
"location":0,"echo":false,"script":null)
,
Object(InputEventKey,"resource_local_to_scene":false,"resource_name":"","device":0,"wi
ndow_id":0,"alt_pressed":false,"shift_pressed":false,"ctrl_pressed":false,"meta_presse
d":false,"pressed":false,"keycode":0,"physical_keycode":4194322,"key_label":0,"unicode
":0,"location":0,"echo":false,"script":null)
]
}
shoot={
"deadzone": 0.5,
"events":
[Object(InputEventMouseButton,"resource_local_to_scene":false,"resource_name":"","devi
ce":-
1,"window_id":0,"alt_pressed":false,"shift_pressed":false,"ctrl_pressed":false,"meta_p
ressed":false,"button_mask":1,"position":Vector2(0, 0),"global_position":Vector2(0,
0),"factor":1.0,"button_index":1,"canceled":false,"pressed":true,"double_click":false,
"script":null)
]
}
[rendering]
renderer/rendering_method="gl_compatibility"
renderer/rendering_method.mobile="gl_compatibility"
```

main.tscn

```
[gd_scene load_steps=17 format=3 uid="uid://cwdse4gnsql0k"]
[ext_resource type="Script" uid="uid://cth5mavvhyb41"
path="res://scripts/game_manager.gd" id="1"]
[ext_resource type="PackedScene" uid="uid://player"
path="res://scenes/entities/player.tscn" id="2"]
[ext_resource type="PackedScene" uid="uid://obstacle"
path="res://scenes/entities/obstacle.tscn" id="4"]
[ext_resource type="PackedScene"
path="res://scenes/entities/zombie_type1_animated.tscn" id="5"]
[ext_resource type="PackedScene"
path="res://scenes/entities/zombie_type2_animated.tscn" id="6"]
[ext_resource type="PackedScene"
path="res://scenes/entities/zombie_type3_animated.tscn" id="7"]
[ext_resource type="PackedScene" path="res://scenes/ui/hud.tscn" id="8"]
[sub_resource type="PlaneMesh" id="PlaneMesh_ground"]
size = Vector2(100, 100)
[sub_resource type="StandardMaterial3D" id="Material_ground"]
albedo_color = Color(0.25, 0.4, 0.15, 1)
roughness = 0.9
metallic = 0.0
uv1_scale = Vector3(20, 20, 20)
uv1_triplanar = true
[sub_resource type="BoxShape3D" id="BoxShape_ground"]
size = Vector3(100, 0.2, 100)
[sub_resource type="NavigationMesh" id="NavigationMesh_1"]
vertices = PackedVector3Array(-50, 0.5, -50, 50, 0.5, -50, 50, 0.5, 50, -50, 0.5, 50)
polygons = [PackedInt32Array(3, 2, 0), PackedInt32Array(0, 2, 1)]
agent_height = 2.0
agent_radius = 0.6
agent_max_climb = 0.5
agent_max_slope = 45.0
region_min_size = 2.0
region_merge_size = 20.0
edge_max_length = 12.0
edge_max_error = 1.3
vertices_per_polygon = 6.0
detail_sample_distance = 6.0
detail_sample_max_error = 1.0
filter_low_hanging_obstacles = true
```

```
filter_ledge_spans = true
filter_walkable_low_height_spans = true
[sub_resource type="Environment" id="Environment_1"]
background_mode = 1
background_color = Color(0.5, 0.7, 1, 1)
[sub_resource type="BoxMesh" id="BoxMesh_wall"]
size = Vector3(100, 6, 2)
[sub_resource type="StandardMaterial3D" id="Material_wall"]
albedo_color = Color(0.45, 0.45, 0.5, 1)
roughness = 0.8
metallic = 0.1
uv1_scale = Vector3(5, 2, 5)
[sub_resource type="BoxShape3D" id="BoxShape_wall"]
size = Vector3(100, 6, 2)
[node name="Main" type="Node3D"]
script = ExtResource("1")
obstacle_scene = ExtResource("4")
zombie_type1_scene = ExtResource("5")
zombie_type2_scene = ExtResource("6")
zombie_type3_scene = ExtResource("7")
[node name="Ground" type="MeshInstance3D" parent="."]
mesh = SubResource("PlaneMesh_ground")
surface_material_override/0 = SubResource("Material_ground")
[node name="StaticBody3D" type="StaticBody3D" parent="Ground"]
[node name="CollisionShape3D" type="CollisionShape3D" parent="Ground/StaticBody3D"]
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, -0.1, 0)
shape = SubResource("BoxShape_ground")
[node name="NavigationRegion3D" type="NavigationRegion3D" parent="."]
navigation_mesh = SubResource("NavigationMesh_1")
[node name="DirectionalLight3D" type="DirectionalLight3D" parent="."]
transform = Transform3D(1, 0, 0, 0, 0.707, 0.707, 0, -0.707, 0.707, 0, 10, 0)
shadow_enabled = true
[node name="WorldEnvironment" type="WorldEnvironment" parent="."]
environment = SubResource("Environment_1")
[node name="Player" parent="." instance=ExtResource("2")]
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, -15)
[node name="Zombies" type="Node3D" parent="."]
[node name="Obstacles" type="Node3D" parent="."]
[node name="Obstacle1" parent="Obstacles" instance=ExtResource("4")]
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, -15, 0, -10)
[node name="Obstacle2" parent="Obstacles" instance=ExtResource("4")]
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 15, 0, -10)
[node name="Obstacle3" parent="Obstacles" instance=ExtResource("4")]
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, -20, 0, 5)
[node name="Obstacle4" parent="Obstacles" instance=ExtResource("4")]
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 20, 0, 5)
[node name="Obstacle5" parent="Obstacles" instance=ExtResource("4")]
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 10)
[node name="Obstacle6" parent="Obstacles" instance=ExtResource("4")]
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, -10, 0, 20)
[node name="Obstacle7" parent="Obstacles" instance=ExtResource("4")]
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 10, 0, 20)
[node name="Obstacle8" parent="Obstacles" instance=ExtResource("4")]
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, -25, 0, -20)
[node name="Obstacle9" parent="Obstacles" instance=ExtResource("4")]
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 25, 0, -20)
[node name="Obstacle10" parent="Obstacles" instance=ExtResource("4")]
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, -25)
[node name="HUD" parent="." instance=ExtResource("8")]
[node name="Walls" type="Node3D" parent="."]
[node name="WallNorth" type="StaticBody3D" parent="Walls"]
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 3, -51)
[node name="MeshInstance3D" type="MeshInstance3D" parent="Walls/WallNorth"]
mesh = SubResource("BoxMesh_wall")
surface_material_override/0 = SubResource("Material_wall")
[node name="CollisionShape3D" type="CollisionShape3D" parent="Walls/WallNorth"]
shape = SubResource("BoxShape_wall")
[node name="WallSouth" type="StaticBody3D" parent="Walls"]
```

```
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 3, 51)
[node name="MeshInstance3D" type="MeshInstance3D" parent="Walls/WallSouth"]
mesh = SubResource("BoxMesh_wall")
surface_material_override/0 = SubResource("Material_wall")
[node name="CollisionShape3D" type="CollisionShape3D" parent="Walls/WallSouth"]
shape = SubResource("BoxShape_wall")
[node name="WallEast" type="StaticBody3D" parent="Walls"]
transform = Transform3D(0, 0, 1, 0, 1, 0, -1, 0, 0, 51, 3, 0)
[node name="MeshInstance3D" type="MeshInstance3D" parent="Walls/WallEast"]
mesh = SubResource("BoxMesh_wall")
surface_material_override/0 = SubResource("Material_wall")
[node name="CollisionShape3D" type="CollisionShape3D" parent="Walls/WallEast"]
shape = SubResource("BoxShape_wall")
[node name="WallWest" type="StaticBody3D" parent="Walls"]
transform = Transform3D(0, 0, 1, 0, 1, 0, -1, 0, 0, -51, 3, 0)
[node name="MeshInstance3D" type="MeshInstance3D" parent="Walls/WallWest"]
mesh = SubResource("BoxMesh_wall")
surface_material_override/0 = SubResource("Material_wall")
[node name="CollisionShape3D" type="CollisionShape3D" parent="Walls/WallWest"]
shape = SubResource("BoxShape_wall")
```

hud.tscn

```
[gd_scene load_steps=2 format=3 uid="uid://hud"]
[ext_resource type="Script" path="res://scripts/ui/hud.gd" id="1"]
[node name="HUD" type="CanvasLayer"]
script = ExtResource("1")
[node name="Panel" type="Panel" parent="."]
offset_left = 10.0
offset_top = 10.0
offset_right = 310.0
offset_bottom = 110.0
[node name="VBoxContainer" type="VBoxContainer" parent="Panel"]
layout_mode = 1
anchors_preset = 15
anchor_right = 1.0
anchor_bottom = 1.0
offset_left = 10.0
offset_top = 10.0
offset_right = -10.0
offset_bottom = -10.0
grow_horizontal = 2
grow_vertical = 2
[node name="PlayerHealthLabel" type="Label" parent="Panel/VBoxContainer"]
layout_mode = 2
text = "Гравець: 100 / 100"
[node name="PlayerHealthBar" type="ProgressBar" parent="Panel/VBoxContainer"]
layout_mode = 2
max_value = 100.0
value = 100.0
show_percentage = false
[node name="WaveLabel" type="Label" parent="Panel/VBoxContainer"]
layout_mode = 2
text = "Хвиля: 1 / 7"
[node name="ZombiesLeftLabel" type="Label" parent="Panel/VBoxContainer"]
layout_mode = 2
text = "Зомбі: 3"
```

base.tscn

```
[gd_scene load_steps=3 format=3 uid="uid://base"]
[ext_resource type="Script" path="res://scripts/entities/base.gd" id="1"]
[ext_resource type="PackedScene" path="res://assets/models/castle_simple.tscn" id="2"]
[sub_resource type="BoxShape3D" id="BoxShape_1"]
size = Vector3(6, 8, 6)
[node name="Base" type="StaticBody3D"]
script = ExtResource("1")
collision_layer = 1
collision_mask = 1
```

```
[node name="CastleModel" parent="." instance=ExtResource("2")]
transform = Transform3D(2, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0)
[node name="CollisionShape3D" type="CollisionShape3D" parent="."]
shape = SubResource("BoxShape_1")
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 3, 0)
[node name="NavigationObstacle3D" type="NavigationObstacle3D" parent="."]
radius = 4.0
height = 8.0
avoidance_enabled = true
```

bullet.tscn

```
[gd_scene load_steps=5 format=3 uid="uid://bullet"]
[ext_resource type="Script" path="res://scripts/entities/bullet.gd" id="1"]
[sub_resource type="SphereMesh" id="SphereMesh_1"]
radius = 0.3
[sub_resource type="StandardMaterial3D" id="Material_bullet"]
albedo_color = Color(1, 1, 0, 1)
emission_enabled = true
emission = Color(1, 1, 0, 1)
emission_energy_multiplier = 2.0
[sub_resource type="SphereShape3D" id="SphereShape_1"]
radius = 0.3
[node name="Bullet" type="RigidBody3D"]
script = ExtResource("1")
gravity_scale = 0.0
lock_rotation = true
contact_monitor = true
max_contacts_reported = 4
[node name="MeshInstance3D" type="MeshInstance3D" parent="."]
mesh = SubResource("SphereMesh_1")
surface_material_override/0 = SubResource("Material_bullet")
[node name="CollisionShape3D" type="CollisionShape3D" parent="."]
shape = SubResource("SphereShape_1")
```

obstacle.tscn

```
[gd_scene load_steps=3 format=3 uid="uid://obstacle"]
[ext_resource type="Script" path="res://scripts/entities/obstacle.gd" id="1"]
[ext_resource type="PackedScene" path="res://assets/models/crate_simple.tscn" id="2"]
[sub_resource type="BoxShape3D" id="BoxShape_1"]
size = Vector3(2, 2, 2)
[node name="Obstacle" type="StaticBody3D"]
script = ExtResource("1")
collision_layer = 1
collision_mask = 1
[node name="CrateModel" parent="." instance=ExtResource("2")]
transform = Transform3D(1.5, 0, 0, 0, 1.5, 0, 0, 0, 1.5, 0, 0, 0)
[node name="CollisionShape3D" type="CollisionShape3D" parent="."]
shape = SubResource("BoxShape_1")
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0)
```

player.tscn

```
[gd_scene load_steps=5 format=3 uid="uid://player"]
[ext_resource type="Script" path="res://scripts/entities/player.gd" id="1"]
[ext_resource type="PackedScene" uid="uid://bullet"
path="res://scenes/entities/bullet.tscn" id="2"]
[ext_resource type="PackedScene" path="res://assets/models/sophia/sophia_skin.tscn"
id="3"]
[sub_resource type="CapsuleShape3D" id="CapsuleShape_1"]
height = 2.0
radius = 0.5
[node name="Player" type="CharacterBody3D"]
script = ExtResource("1")
bullet_scene = ExtResource("2")
[node name="SophiaModel" parent="." instance=ExtResource("3")]
transform = Transform3D(0.5, 0, 0, 0, 0.5, 0, 0, 0, 0.5, 0, 0, 0)
[node name="CollisionShape3D" type="CollisionShape3D" parent="."]
```

```
shape = SubResource("CapsuleShape_1")
[node name="Camera3D" type="Camera3D" parent="."]
transform = Transform3D(1, 0, 0, 0, 0.342, 0.940, 0, -0.940, 0.342, 0, 40, 15)
current = true
fov = 60.0
[node name="ShootPoint" type="Marker3D" parent="."]
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0.75, -1.5)
```

zombie_type1.tscn

```
[gd_scene load_steps=5 format=3 uid="uid://0ombie1"]
[ext_resource type="Script" path="res://scripts/ai/zombie_type1.gd" id="1"]
[sub_resource type="CapsuleMesh" id="CapsuleMesh_1"]
radius = 0.4
height = 1.5
[sub_resource type="StandardMaterial3D" id="Material_zombie1"]
albedo_color = Color(0, 1, 0, 1)
[sub_resource type="CapsuleShape3D" id="CapsuleShape_1"]
radius = 0.5
height = 1.5
[node name="ZombieType1" type="CharacterBody3D"]
script = ExtResource("1")
[node name="MeshInstance3D" type="MeshInstance3D" parent="."]
mesh = SubResource("CapsuleMesh_1")
surface_material_override/0 = SubResource("Material_zombie1")
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0.75, 0)
[node name="CollisionShape3D" type="CollisionShape3D" parent="."]
shape = SubResource("CapsuleShape_1")
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0.75, 0)
[node name="NavigationAgent3D" type="NavigationAgent3D" parent="."]
```

zombie_type1_animated.tscn

```
[gd_scene load_steps=4 format=3 uid="uid://zombie1_animated"]
[ext_resource type="Script" path="res://scripts/ai/zombie_type1.gd" id="1"]
[ext_resource type="PackedScene" uid="uid://bg4lsgoyx4i5"
path="res://assets/models/zombie/zombie_animated.gltf" id="2"]
[sub_resource type="CapsuleShape3D" id="CapsuleShape_1"]
radius = 0.5
height = 1.5
[node name="ZombieType1" type="CharacterBody3D"]
script = ExtResource("1")
max_hp = 100.0
speed = 2.0
damage = 10.0
attack_range = 1.5
[node name="ZombieModel" parent="." instance=ExtResource("2")]
transform = Transform3D(0.5, 0, 0, 0, 0.5, 0, 0, 0, 0.5, 0, 0, 0)
[node name="CollisionShape3D" type="CollisionShape3D" parent="."]
shape = SubResource("CapsuleShape_1")
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0.75, 0)
[node name="NavigationAgent3D" type="NavigationAgent3D" parent="."]
```

zombie_type2.tscn

```
[gd_scene load_steps=5 format=3 uid="uid://0ombie2"]
[ext_resource type="Script" path="res://scripts/ai/zombie_type2.gd" id="1"]
[sub_resource type="CapsuleMesh" id="CapsuleMesh_1"]
radius = 0.4
height = 1.5
[sub_resource type="StandardMaterial3D" id="Material_zombie2"]
albedo_color = Color(1, 0.5, 0, 1)
[sub_resource type="CapsuleShape3D" id="CapsuleShape_1"]
radius = 0.5
height = 1.5
[node name="ZombieType2" type="CharacterBody3D"]
script = ExtResource("1")
[node name="MeshInstance3D" type="MeshInstance3D" parent="."]
mesh = SubResource("CapsuleMesh_1")
```

```
surface_material_override/0 = SubResource("Material_zombie2")
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0.75, 0)
[node name="CollisionShape3D" type="CollisionShape3D" parent="."]
shape = SubResource("CapsuleShape_1")
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0.75, 0)
```

zombie_type2_animated.tscn

```
[gd_scene load_steps=4 format=3 uid="uid://zombie2_animated"]
[ext_resource type="Script" path="res://scripts/ai/zombie_type2.gd" id="1"]
[ext_resource type="PackedScene" uid="uid://bg4lsgoyx4i5"
path="res://assets/models/zombie/zombie_animated.gltf" id="2"]
[sub_resource type="CapsuleShape3D" id="CapsuleShape_1"]
radius = 0.5
height = 1.5
[node name="ZombieType2" type="CharacterBody3D"]
script = ExtResource("1")
max_hp = 150.0
speed = 3.0
damage = 15.0
attack_range = 1.5
[node name="ZombieModel" parent="." instance=ExtResource("2")]
transform = Transform3D(0.5, 0, 0, 0, 0.5, 0, 0, 0, 0.5, 0, 0, 0)
[node name="CollisionShape3D" type="CollisionShape3D" parent="."]
shape = SubResource("CapsuleShape_1")
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0.75, 0)
```

zombie_type3.tscn

```
[gd_scene load_steps=5 format=3 uid="uid://0ombie3"]
[ext_resource type="Script" path="res://scripts/ai/zombie_type3.gd" id="1"]
[sub_resource type="CapsuleMesh" id="CapsuleMesh_1"]
radius = 0.4
height = 1.5
[sub_resource type="StandardMaterial3D" id="Material_zombie3"]
albedo_color = Color(0.5, 0, 0.5, 1)
[sub_resource type="CapsuleShape3D" id="CapsuleShape_1"]
radius = 0.5
height = 1.5
[node name="ZombieType3" type="CharacterBody3D"]
script = ExtResource("1")
[node name="MeshInstance3D" type="MeshInstance3D" parent="."]
mesh = SubResource("CapsuleMesh_1")
surface_material_override/0 = SubResource("Material_zombie3")
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0.75, 0)
[node name="CollisionShape3D" type="CollisionShape3D" parent="."]
shape = SubResource("CapsuleShape_1")
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0.75, 0)
```

zombie_type3_animated.tscn

```
[gd_scene load_steps=4 format=3 uid="uid://zombie3_animated"]
[ext_resource type="Script" path="res://scripts/ai/zombie_type3.gd" id="1"]
[ext_resource type="PackedScene" uid="uid://bg4lsgoyx4i5"
path="res://assets/models/zombie/zombie_animated.gltf" id="2"]
[sub_resource type="CapsuleShape3D" id="CapsuleShape_1"]
radius = 0.5
height = 1.5
[node name="ZombieType3" type="CharacterBody3D"]
script = ExtResource("1")
max_hp = 200.0
speed = 2.5
damage = 12.0
attack_range = 1.5
[node name="ZombieModel" parent="." instance=ExtResource("2")]
transform = Transform3D(0.5, 0, 0, 0, 0.5, 0, 0, 0, 0.5, 0, 0, 0)
[node name="CollisionShape3D" type="CollisionShape3D" parent="."]
shape = SubResource("CapsuleShape_1")
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0.75, 0)
```

castle_simple.tscn

```
[gd_scene load_steps=3 format=3]
[sub_resource type="BoxMesh" id="BoxMesh_1"]
size = Vector3(4, 6, 4)
[sub_resource type="StandardMaterial3D" id="Material_castle"]
albedo_color = Color(0.6, 0.6, 0.65, 1)
roughness = 0.7
metallic = 0.2
[node name="Castle" type="MeshInstance3D"]
mesh = SubResource("BoxMesh_1")
surface_material_override/0 = SubResource("Material_castle")
transform = Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 3, 0)
```

crate_simple.tscn

```
[gd_scene load_steps=3 format=3]
[sub_resource type="BoxMesh" id="BoxMesh_1"]
size = Vector3(1, 1, 1)
[sub_resource type="StandardMaterial3D" id="Material_crate"]
albedo_color = Color(0.55, 0.35, 0.2, 1)
roughness = 0.8
metallic = 0.0
[node name="Crate" type="MeshInstance3D"]
mesh = SubResource("BoxMesh_1")
surface_material_override/0 = SubResource("Material_crate")
```

sophia_skin.tscn

```
[gd_scene load_steps=38 format=3 uid="uid://prh35jb6tjtd"]
[ext_resource type="PackedScene" uid="uid://16iu10wxub40"
path="res://addons/gdquest_sophia/model/sophia.glb" id="1_e4pev"]
[ext_resource type="Script" path="res://addons/gdquest_sophia/sophia_skin.gd"
id="1_obcib"]
[ext_resource type="Material" uid="uid://dye01l0ct4fby"
path="res://addons/gdquest_sophia/model/materials/eye_mat_override.tres" id="4_mms51"]
[sub_resource type="AnimationNodeAnimation" id="AnimationNodeAnimation_vapre"]
animation = &"EdgeGrab"
[sub_resource type="AnimationNodeAnimation" id="AnimationNodeAnimation_84eem"]
animation = &"Fall"
[sub_resource type="AnimationNodeAnimation" id="AnimationNodeAnimation_bdqby"]
animation = &"Idle"
[sub_resource type="AnimationNodeAnimation" id="AnimationNodeAnimation_is0ey"]
animation = &"Jump"
[sub_resource type="AnimationNodeAnimation" id="AnimationNodeAnimation_olyh3"]
animation = &"RunTiltL"
[sub_resource type="AnimationNodeAnimation" id="AnimationNodeAnimation_81hhq"]
animation = &"RunTiltR"
[sub_resource type="AnimationNodeAnimation" id="AnimationNodeAnimation_nf0s3"]
animation = &"Run"
[sub_resource type="AnimationNodeAdd3" id="AnimationNodeAdd3_i8et5"]
[sub_resource type="AnimationNodeBlendTree" id="AnimationNodeBlendTree_mx8fd"]
graph_offset = Vector2(-362, 27)
nodes/L/node = SubResource("AnimationNodeAnimation_olyh3")
nodes/L/position = Vector2(-100, 320)
nodes/R/node = SubResource("AnimationNodeAnimation_81hhq")
nodes/R/position = Vector2(-240, 240)
nodes/Run/node = SubResource("AnimationNodeAnimation_nf0s3")
nodes/Run/position = Vector2(-120, 120)
nodes/tilt/node = SubResource("AnimationNodeAdd3_i8et5")
nodes/tilt/position = Vector2(80, 160)
node_connections = [&"output", 0, &"tilt", &"tilt", 0, &"L", &"tilt", 1, &"Run",
&"tilt", 2, &"R"]
[sub_resource type="AnimationNodeAnimation" id="AnimationNodeAnimation_ln86s"]
animation = &"WallSlide"
[sub_resource type="AnimationNodeStateMachineTransition"
id="AnimationNodeStateMachineTransition_aehxm"]
advance_mode = 2
```

```
[sub_resource type="AnimationNodeStateMachineTransition"
id="AnimationNodeStateMachineTransition_bp3m8"]
xfade_time = 0.1
[sub_resource type="AnimationNodeStateMachineTransition"
id="AnimationNodeStateMachineTransition_kwnko"]
xfade_time = 0.1
[sub_resource type="AnimationNodeStateMachineTransition"
id="AnimationNodeStateMachineTransition_ykos7"]
xfade_time = 0.2
[sub_resource type="AnimationNodeStateMachineTransition"
id="AnimationNodeStateMachineTransition_0wv7u"]
xfade_time = 0.2
[sub_resource type="AnimationNodeStateMachineTransition"
id="AnimationNodeStateMachineTransition_5rcd0"]
xfade_time = 0.1
[sub_resource type="AnimationNodeStateMachineTransition"
id="AnimationNodeStateMachineTransition_umbj3"]
xfade_time = 0.1
[sub_resource type="AnimationNodeStateMachineTransition"
id="AnimationNodeStateMachineTransition_525xv"]
xfade_time = 0.1
[sub_resource type="AnimationNodeStateMachineTransition"
id="AnimationNodeStateMachineTransition_5lsn0"]
xfade_time = 0.2
[sub_resource type="AnimationNodeStateMachineTransition"
id="AnimationNodeStateMachineTransition_k4ifp"]
xfade_time = 0.1
[sub_resource type="AnimationNodeStateMachineTransition"
id="AnimationNodeStateMachineTransition_i5k5f"]
xfade_time = 0.1
[sub_resource type="AnimationNodeStateMachineTransition"
id="AnimationNodeStateMachineTransition_graxy"]
xfade_time = 0.1
[sub_resource type="AnimationNodeStateMachineTransition"
id="AnimationNodeStateMachineTransition_8tjks"]
[sub_resource type="AnimationNodeStateMachineTransition"
id="AnimationNodeStateMachineTransition_h6oe5"]
xfade_time = 0.1
[sub_resource type="AnimationNodeStateMachineTransition"
id="AnimationNodeStateMachineTransition_8l37g"]
xfade_time = 0.1
[sub_resource type="AnimationNodeStateMachineTransition"
id="AnimationNodeStateMachineTransition_p1j7t"]
xfade_time = 0.1
[sub_resource type="AnimationNodeStateMachineTransition"
id="AnimationNodeStateMachineTransition_mybu0"]
xfade_time = 0.1
[sub_resource type="AnimationNodeStateMachineTransition"
id="AnimationNodeStateMachineTransition_6rf72"]
xfade_time = 0.1
[sub_resource type="AnimationNodeStateMachineTransition"
id="AnimationNodeStateMachineTransition_cm2qm"]
xfade_time = 0.1
[sub_resource type="AnimationNodeStateMachineTransition"
id="AnimationNodeStateMachineTransition_goywk"]
xfade_time = 0.1
[sub_resource type="AnimationNodeStateMachineTransition"
id="AnimationNodeStateMachineTransition_mn3tt"]
xfade_time = 0.1
[sub_resource type="AnimationNodeStateMachineTransition"
id="AnimationNodeStateMachineTransition_cwktt"]
xfade_time = 0.1
[sub_resource type="AnimationNodeStateMachine" id="AnimationNodeStateMachine_xxcga"]
states/EdgeGrab/node = SubResource("AnimationNodeAnimation_vapre")
states/EdgeGrab/position = Vector2(327, 290)
states/End/position = Vector2(627, 166)
states/Fall/node = SubResource("AnimationNodeAnimation_84eem")
states/Fall/position = Vector2(474, 205)
```

```
states/Idle/node = SubResource("AnimationNodeAnimation_bdqby")
states/Idle/position = Vector2(327, 122)
states/Jump/node = SubResource("AnimationNodeAnimation_is0ey")
states/Jump/position = Vector2(327, 205)
states/Move/node = SubResource("AnimationNodeBlendTree_mx8fd")
states/Move/position = Vector2(474, 122)
states/Start/position = Vector2(327, 44)
states/WallSlide/node = SubResource("AnimationNodeAnimation_ln86s")
states/WallSlide/position = Vector2(474, 290)
transitions = ["Start", "Idle",
SubResource("AnimationNodeStateMachineTransition_aehxm"), "Idle", "Jump",
SubResource("AnimationNodeStateMachineTransition_bp3m8"), "Jump", "Idle",
SubResource("AnimationNodeStateMachineTransition_kwnko"), "Jump", "Fall",
SubResource("AnimationNodeStateMachineTransition_ykos7"), "Fall", "Jump",
SubResource("AnimationNodeStateMachineTransition_0wv7u"), "Fall", "Idle",
SubResource("AnimationNodeStateMachineTransition_5rcd0"), "Idle", "Fall",
SubResource("AnimationNodeStateMachineTransition_umbj3"), "Idle", "Move",
SubResource("AnimationNodeStateMachineTransition_525xv"), "Move", "Idle",
SubResource("AnimationNodeStateMachineTransition_5lsn0"), "Fall", "Move",
SubResource("AnimationNodeStateMachineTransition_k4ifp"), "Move", "Fall",
SubResource("AnimationNodeStateMachineTransition_i5k5f"), "Move", "Jump",
SubResource("AnimationNodeStateMachineTransition_graxy"), "Jump", "Move",
SubResource("AnimationNodeStateMachineTransition_8tjks"), "Jump", "EdgeGrab",
SubResource("AnimationNodeStateMachineTransition_h6oe5"), "WallSlide", "Fall",
SubResource("AnimationNodeStateMachineTransition_8l37g"), "Jump", "WallSlide",
SubResource("AnimationNodeStateMachineTransition_p1j7t"), "EdgeGrab", "Fall",
SubResource("AnimationNodeStateMachineTransition_mybu0"), "EdgeGrab", "WallSlide",
SubResource("AnimationNodeStateMachineTransition_6rf72"), "Fall", "EdgeGrab",
SubResource("AnimationNodeStateMachineTransition_cm2qm"), "WallSlide", "Jump",
SubResource("AnimationNodeStateMachineTransition_goywk"), "EdgeGrab", "Jump",
SubResource("AnimationNodeStateMachineTransition_mn3tt"), "Fall", "WallSlide",
SubResource("AnimationNodeStateMachineTransition_cwktt")]
graph_offset = Vector2(-264, 106)
[sub_resource type="AnimationNodeBlendTree" id="AnimationNodeBlendTree_qa7x4"]
nodes/StateMachine/node = SubResource("AnimationNodeStateMachine_xxcga")
nodes/StateMachine/position = Vector2(0, 0)
nodes/output/position = Vector2(200, 0)
node_connections = [&"output", 0, &"StateMachine"]
[node name="SophiaSkin" type="Node3D"]
script = ExtResource("1_obcib")
[node name="sophia" parent="." instance=ExtResource("1_e4pev")]
[node name="Skeleton3D" parent="sophia/rig" index="0"]
bones/0/position = Vector3(0.00732047, 0.496295, -0.00865961)
bones/0/rotation = Quaternion(0.126223, 0.0239208, 0.0190269, 0.991531)
bones/1/rotation = Quaternion(-0.0644765, 2.41677e-09, -4.12252e-09, 0.997919)
bones/2/rotation = Quaternion(-0.0774405, -0.0151184, 0.000211473, 0.996882)
bones/3/rotation = Quaternion(-0.00164818, -0.0452527, -0.00103567, 0.998974)
bones/4/rotation = Quaternion(0.147706, 0.0299976, -0.00340939, 0.98857)
bones/5/rotation = Quaternion(-0.114648, -1.05998e-09, 1.4507e-08, 0.993406)
bones/6/rotation = Quaternion(-0.0158346, 0.0301867, -0.000478268, 0.999419)
bones/7/rotation = Quaternion(0.993169, 0.114706, -0.000951129, -0.0213738)
bones/8/rotation = Quaternion(0.993177, -0.114642, -0.00394469, 0.0210281)
bones/9/rotation = Quaternion(0.842715, 0.0143906, -0.00918863, 0.53809)
bones/10/rotation = Quaternion(0.408879, -0.0281485, 0.0626473, 0.910001)
bones/11/rotation = Quaternion(0.813589, 0.158501, -0.219131, 0.514716)
bones/12/rotation = Quaternion(0.40807, -0.0397529, 0.0655289, 0.909728)
bones/13/rotation = Quaternion(0.818526, -0.13063, 0.20143, 0.521898)
bones/14/rotation = Quaternion(0.409668, -0.0165427, 0.0597556, 0.910125)
bones/15/rotation = Quaternion(-0.355847, -0.0616587, -0.326693, 0.873409)
bones/16/rotation = Quaternion(2.71044e-08, 1.10081e-07, -0.114752, 0.993394)
bones/17/rotation = Quaternion(-0.340372, 0.120732, 0.469178, 0.80588)
bones/18/rotation = Quaternion(7.03448e-08, -6.57686e-08, -0.114753, 0.993394)
bones/19/rotation = Quaternion(-0.411395, 0.00544888, 0.0477706, 0.910188)
bones/20/rotation = Quaternion(-2.19213e-07, -3.52495e-08, -0.114752, 0.993394)
bones/21/rotation = Quaternion(-0.406432, 0.0639422, 0.106264, 0.905225)
bones/22/rotation = Quaternion(-2.27535e-07, 3.06473e-09, -0.114752, 0.993394)
bones/23/rotation = Quaternion(-0.579758, -0.382617, -0.396489, 0.600235)
bones/24/rotation = Quaternion(-0.193183, 0.62574, -0.519537, 0.548827)
```

```
bones/25/position = Vector3(-2.84344e-08, 0.0693518, 7.45058e-09)
bones/25/rotation = Quaternion(8.69444e-08, -0.0235637, 1.42425e-07, 0.999722)
bones/26/position = Vector3(-3.30328e-08, 0.0693518, -7.25292e-09)
bones/26/rotation = Quaternion(0.386254, -0.023564, 0.00987916, 0.922038)
bones/27/position = Vector3(-1.00059e-07, 0.0641195, -5.12226e-09)
bones/27/rotation = Quaternion(-9.84038e-08, 0.00133698, 3.28107e-09, 0.999999)
bones/28/position = Vector3(2.54637e-08, 0.0641197, 1.70021e-08)
bones/28/rotation = Quaternion(0.0577325, 0.0017042, 0.0518438, 0.996984)
bones/30/rotation = Quaternion(0.586205, -0.0728153, -0.158498, 0.791164)
bones/31/rotation = Quaternion(0.422653, -0.000300065, -0.00100193, 0.906291)
bones/32/rotation = Quaternion(0.424779, 0.000651514, 0.00163282, 0.905296)
bones/33/rotation = Quaternion(-0.0432788, 0.822374, 0.418861, 0.382602)
bones/34/rotation = Quaternion(0.605953, -0.0519355, 0.0526438, 0.792056)
bones/35/rotation = Quaternion(0.490047, -0.012869, -0.0253105, 0.871234)
bones/37/rotation = Quaternion(0.594404, -0.0768252, -0.0772993, 0.796747)
bones/38/rotation = Quaternion(0.425027, 0.000226274, 0.000737527, 0.90518)
bones/39/rotation = Quaternion(0.424791, -0.000484585, -0.00120235, 0.905291)
bones/41/rotation = Quaternion(0.604183, 0.0209748, -0.0202136, 0.796313)
bones/42/rotation = Quaternion(0.496983, -7.6202e-05, -0.000187088, 0.86776)
bones/43/rotation = Quaternion(0.498019, 0.000152893, 0.000304259, 0.867166)
bones/45/rotation = Quaternion(0.608247, 0.0350251, 0.0523038, 0.791248)
bones/46/rotation = Quaternion(0.510495, 1.10653e-05, 4.87684e-05, 0.859881)
bones/47/rotation = Quaternion(0.514301, -3.90808e-05, -7.17982e-05, 0.85761)
bones/49/rotation = Quaternion(-0.343645, -0.657727, 0.369271, 0.559413)
bones/50/position = Vector3(6.54545e-08, 0.0693517, 1.11759e-08)
bones/50/rotation = Quaternion(5.1947e-08, -0.0241554, -1.31345e-08, 0.999708)
bones/51/position = Vector3(6.39701e-08, 0.0693518, 2.12806e-08)
bones/51/rotation = Quaternion(0.251754, -0.0241554, 0.00628529, 0.96747)
bones/52/position = Vector3(3.46918e-08, 0.0641197, -9.45292e-08)
bones/52/rotation = Quaternion(4.6215e-08, 0.00133735, -1.68601e-07, 0.999999)
bones/53/position = Vector3(4.31197e-08, 0.0641198, -9.50838e-08)
bones/53/rotation = Quaternion(-0.0628506, 0.00143574, -0.0518679, 0.996673)
bones/55/position = Vector3(-0.00788145, 0.126777, 0.022745)
bones/55/rotation = Quaternion(0.634935, 0.0809939, 0.129391, 0.757335)
bones/56/rotation = Quaternion(0.422333, 0.037719, -0.0164492, 0.905506)
bones/57/rotation = Quaternion(0.424778, -0.000652886, -0.00163169, 0.905296)
bones/58/rotation = Quaternion(0.0432787, 0.822373, 0.418861, -0.382602)
bones/59/rotation = Quaternion(0.605953, 0.0519355, -0.0526439, 0.792055)
bones/60/rotation = Quaternion(0.490047, 0.012869, 0.0253107, 0.871234)
bones/62/position = Vector3(-0.00561923, 0.126171, 0.0233237)
bones/62/rotation = Quaternion(0.642486, 0.0803638, 0.0528975, 0.760233)
bones/63/rotation = Quaternion(0.424663, 0.0356998, -0.0176063, 0.904476)
bones/64/rotation = Quaternion(0.424791, 0.000484688, 0.00120234, 0.905291)
bones/66/position = Vector3(-0.00836084, 0.127142, 0.0226955)
bones/66/rotation = Quaternion(0.653346, -0.0206434, 0.00129776, 0.756777)
bones/67/rotation = Quaternion(0.496619, 0.0336209, -0.0190247, 0.867109)
bones/68/rotation = Quaternion(0.498018, -0.000151753, -0.000304719, 0.867167)
bones/70/position = Vector3(-0.00570161, 0.122795, 0.022938)
bones/70/rotation = Quaternion(0.657362, -0.0389285, -0.0650824, 0.749749)
bones/71/rotation = Quaternion(0.51019, 0.0296037, -0.0176302, 0.859371)
bones/72/rotation = Quaternion(0.514301, 3.88323e-05, 7.19582e-05, 0.85761)
bones/74/rotation = Quaternion(0.782333, 0.248616, -0.171126, 0.54485)
bones/75/rotation = Quaternion(0.782333, -0.248616, 0.171126, 0.54485)
bones/76/rotation = Quaternion(0.985966, 2.84497e-08, -1.60425e-08, -0.166949)
bones/77/rotation = Quaternion(-0.0306789, 1.95752e-08, 9.25838e-10, 0.999529)
bones/78/rotation = Quaternion(-0.0301015, -9.41682e-09, 5.13173e-09, 0.999547)
bones/81/rotation = Quaternion(0.973182, 0.0675392, -0.0491386, 0.21434)
bones/82/position = Vector3(4.02629e-09, 0.0942519, -2.73391e-08)
bones/82/rotation = Quaternion(-2.70491e-08, -0.0599869, -2.0137e-08, 0.998199)
bones/83/position = Vector3(1.58899e-08, 0.094252, 1.33295e-08)
bones/83/rotation = Quaternion(0.227785, -0.0600212, 0.0140675, 0.971758)
bones/84/position = Vector3(1.7517e-09, 0.0925612, 1.02773e-08)
bones/84/rotation = Quaternion(-1.68028e-08, -0.0226435, -4.73646e-08, 0.999744)
bones/85/position = Vector3(1.93904e-08, 0.0925611, -2.20688e-08)
bones/85/rotation = Quaternion(-0.537997, 0.0124302, 0.079219, 0.839124)
bones/86/position = Vector3(-2.67091e-08, 0.224084, 3.77875e-08)
bones/86/rotation = Quaternion(2.12939e-05, 0.945936, -0.324355, -6.57396e-06)
bones/87/rotation = Quaternion(0.95634, -0.141176, 0.0608766, 0.248551)
```

```
bones/88/position = Vector3(-1.59564e-08, 0.0947332, 5.93381e-10)
bones/88/rotation = Quaternion(9.59703e-10, 0.0514967, 2.93541e-08, 0.998673)
bones/89/position = Vector3(-2.43564e-08, 0.0947333, 2.32829e-09)
bones/89/rotation = Quaternion(0.313176, 0.0514827, -0.0170046, 0.948146)
bones/90/position = Vector3(1.07835e-08, 0.0930518, -1.30186e-08)
bones/90/rotation = Quaternion(3.19252e-08, 0.0473046, -6.61013e-08, 0.998881)
bones/91/position = Vector3(-3.91719e-09, 0.0930518, 1.02204e-09)
bones/91/rotation = Quaternion(-0.55629, 0.0123465, -0.104437, 0.824307)
bones/92/position = Vector3(-1.14476e-08, 0.224088, 3.48135e-08)
bones/92/rotation = Quaternion(-2.57156e-05, 0.940253, -0.340476, 8.60947e-06)
[node name="Sophia" parent="sophia/rig/Skeleton3D" index="0"]
surface_material_override/1 = ExtResource("4_mms51")
surface_material_override/2 = ExtResource("4_mms51")
[node name="AnimationTree" type="AnimationTree" parent="."]
unique_name_in_owner = true
root_node = NodePath("%AnimationTree/../sophia")
tree_root = SubResource("AnimationNodeBlendTree_qa7x4")
anim_player = NodePath("../sophia/AnimationPlayer")
parameters/StateMachine/Move/tilt/add_amount = 0.0
[node name="BlinkTimer" type="Timer" parent="."]
unique_name_in_owner = true
wait_time = 0.2
one_shot = true
autostart = true
[node name="ClosedEyesTimer" type="Timer" parent="."]
unique_name_in_owner = true
wait_time = 0.2
one_shot = true
autostart = true
[editable path="sophia"]
```

sophia_material.tres

```
[gd_resource type="StandardMaterial3D" load_steps=5 format=3
uid="uid://bv3j2e7k6f5xf"]
[ext_resource type="Texture2D" uid="uid://6jb7dt4d54ck"
path="res://addons/gdquest_sophia/model/sophia_diffuse.png" id="1_m7hxq"]
[ext_resource type="Texture2D" uid="uid://dbg3uqtvqxigv"
path="res://addons/gdquest_sophia/model/sophia_roughness.png" id="2_fbyat"]
[ext_resource type="Texture2D" uid="uid://mdhln8lltmyo"
path="res://addons/gdquest_sophia/model/sophia_normal.png" id="3_tmo2m"]
[ext_resource type="Texture2D" uid="uid://dxqmgxq6o21yr"
path="res://addons/gdquest_sophia/model/SSS.png" id="4_luinf"]
[resource]
resource_name = "sophia_material"
cull_mode = 2
vertex_color_use_as_albedo = true
albedo_texture = ExtResource("1_m7hxq")
metallic_texture = ExtResource("2_fbyat")
metallic_texture_channel = 2
roughness = 0.8
roughness_texture = ExtResource("2_fbyat")
roughness_texture_channel = 1
normal_enabled = true
normal_texture = ExtResource("3_tmo2m")
rim_enabled = true
rim = 0.5
subsurf_scatter_enabled = true
subsurf_scatter_strength = 0.25
subsurf_scatter_skin_mode = true
subsurf_scatter_texture = ExtResource("4_luinf")
```

mouth_mat.tres

```
[gd_resource type="StandardMaterial3D" load_steps=3 format=3 uid="uid://ejs88styylln"]
[ext_resource type="Texture2D" uid="uid://d3gigd8y5hjg"
path="res://addons/gdquest_sophia/model/sophia_mouth_smile_diffuse.png" id="1_8gei4"]
[ext_resource type="Texture2D" uid="uid://c36m8cv33f0yi"
path="res://addons/gdquest_sophia/model/sophia_mouth_smile_normal.png" id="2_e4i0b"]
```

```
[resource]
resource_name = "mouth_mat"
transparency = 1
vertex_color_use_as_albedo = true
albedo_texture = ExtResource("1_8gei4")
roughness = 0.5
normal_enabled = true
normal_texture = ExtResource("2_e4i0b")
```

eye_mat_R.tres

```
[gd_resource type="StandardMaterial3D" load_steps=3 format=3
uid="uid://cdp5t51v8o1rs"]
[ext_resource type="Texture2D" uid="uid://bqj7x6pcb73iy"
path="res://addons/gdquest_sophia/model/sophia_eye_open_diffuse.png" id="1_a6jqn"]
[ext_resource type="Texture2D" uid="uid://pcq7xge7vncv"
path="res://addons/gdquest_sophia/model/sophia_eye_open_normal.png" id="2_iuhho"]
[resource]
resource_name = "eye_mat_R"
transparency = 2
alpha_scissor_threshold = 0.5
alpha_antialiasing_mode = 0
vertex_color_use_as_albedo = true
albedo_texture = ExtResource("1_a6jqn")
roughness = 0.5
normal_enabled = true
normal_texture = ExtResource("2_iuhho")
```

eye_mat_override.tres

```
[gd_resource type="StandardMaterial3D" load_steps=3 format=3
uid="uid://dye01l0ct4fby"]
[ext_resource type="Texture2D" uid="uid://ctfbobgg7ts8d"
path="res://addons/gdquest_sophia/model/textures/eyes_diffuse_map.png" id="1_oox6g"]
[ext_resource type="Texture2D" uid="uid://bjvo8w2ptm7fc"
path="res://addons/gdquest_sophia/model/textures/eyes_normal_map.png" id="2_r1vvu"]
[resource]
resource_name = "eye_mat_L"
transparency = 1
vertex_color_use_as_albedo = true
albedo_texture = ExtResource("1_oox6g")
roughness = 0.5
normal_enabled = true
normal_texture = ExtResource("2_r1vvu")
uv1_scale = Vector3(1, 0.5, 1)
```

eye_mat_L.tres

```
[gd_resource type="StandardMaterial3D" load_steps=3 format=3 uid="uid://cal2s6wv6y3j"]
[ext_resource type="Texture2D" uid="uid://bqj7x6pcb73iy"
path="res://addons/gdquest_sophia/model/sophia_eye_open_diffuse.png" id="1_puhan"]
[ext_resource type="Texture2D" uid="uid://pcq7xge7vncv"
path="res://addons/gdquest_sophia/model/sophia_eye_open_normal.png" id="2_8lr5k"]
[resource]
resource_name = "eye_mat_L"
transparency = 2
alpha_scissor_threshold = 0.5
alpha_antialiasing_mode = 0
vertex_color_use_as_albedo = true
albedo_texture = ExtResource("1_puhan")
roughness = 0.5
normal_enabled = true
normal_texture = ExtResource("2_8lr5k")
```

dock_window.tscn

```
[gd_scene load_steps=2 format=3 uid="uid://bf76pyorby581"]
[ext_resource type="Script"
path="res://addons/claude_3.5_sonnet_chat_api/dock_window.gd" id="1_aaayo"]
[node name="Claude API" type="Control"]
```

```
layout_mode = 3
anchors_preset = 15
anchor_right = 1.0
anchor_bottom = 1.0
grow_horizontal = 2
grow_vertical = 2
script = ExtResource("1_aaayo")
[node name="VBoxContainer" type="VBoxContainer" parent="."]
layout_mode = 1
anchors_preset = 15
anchor_right = 1.0
anchor_bottom = 1.0
grow_horizontal = 2
grow_vertical = 2
[node name="Input" type="TextEdit" parent="VBoxContainer"]
unique_name_in_owner = true
layout_mode = 2
size_flags_vertical = 3
text = "Enter your message for Claude..."
wrap_mode = 1
[node name="HBoxContainer" type="HBoxContainer" parent="VBoxContainer"]
layout_mode = 2
size_flags_vertical = 4
[node name="SendButton" type="Button" parent="VBoxContainer/HBoxContainer"]
unique_name_in_owner = true
layout_mode = 2
size_flags_vertical = 4
text = "Send to Claude"
[node name="ClearButton" type="Button" parent="VBoxContainer/HBoxContainer"]
unique_name_in_owner = true
layout_mode = 2
text = "Clear"
[node name="StatusLabel" type="Label" parent="VBoxContainer"]
unique_name_in_owner = true
layout_mode = 2
[node name="Output" type="TextEdit" parent="VBoxContainer"]
unique_name_in_owner = true
layout_mode = 2
size_flags_vertical = 3
[node name="HTTPRequest" type="HTTPRequest" parent="VBoxContainer"]
unique_name_in_owner = true
[connection signal="pressed" from="VBoxContainer/HBoxContainer/SendButton" to="."
method="_on_send_pressed"]
[connection signal="pressed" from="VBoxContainer/HBoxContainer/ClearButton" to="."
method="_on_clear_pressed"]
[connection    signal="request_completed"    from="VBoxContainer/HTTPRequest"    to="."
method="_on_request_completed"]
```

blackboard.gd

```
extends Node
class_name Blackboard
# ============================================
# BLACKBOARD - ЗАГАЛЬНА ПАМ'ЯТЬ ДЛЯ АГЕНТІВ
# Використовується для координації Type II зомбі
# ============================================
# Singleton pattern
static var instance: Blackboard = null
# Загальна інформація про гравця
var player_position: Vector3 = Vector3.ZERO
var player_facing: Vector3 = Vector3.ZERO
var player_last_seen: float = 0.0
# Координація атак
var flanking_positions: Dictionary = {}  # {zombie_id: position}
var attacking_zombies: Array = []
var zombie_positions: Dictionary = {}  # {zombie_id: position}
# TYPE III - Колективний розум
var type3_zombies: Dictionary = {}  # {zombie_id: {position, ready_to_attack,
```

```
surround_angle}}
var surround_positions: Dictionary = {}  # {zombie_id: assigned_position}
var swarm_ready_count: int = 0  # Скільки Type III готові до атаки
var target_surround_radius: float = 5.0  # Радіус оточення
var swarm_attack_threshold: int = 2  # Мінімум зомбі для атаки зграї
# Тактична інформація
var dangerous_zones: Array[Vector3] = []  # Зони де зомбі часто гинуть
var safe_approach_vectors: Array[Vector3] = []  # Безпечні напрямки атаки
# Цілі та пріоритети
var shared_target: Node3D = null
var priority_action: String = ""  # "FLANK", "SURROUND", "DIRECT_ATTACK"
func _init():
	if instance == null:
		instance = self
static func get_instance() -> Blackboard:
	if instance == null:
		instance = Blackboard.new()
	return instance
# ============================================
# ОНОВЛЕННЯ ІНФОРМАЦІЇ
# ============================================
func update_player_info(position: Vector3, facing: Vector3):
	"""Оновлення інформації про гравця"""
	player_position = position
	player_facing = facing
	player_last_seen = Time.get_ticks_msec() / 1000.0
func register_zombie(zombie_id: int, position: Vector3):
	"""Реєстрація зомбі в системі"""
	zombie_positions[zombie_id] = position
	if zombie_id not in attacking_zombies:
		pass  # Можна додати до пулу
func unregister_zombie(zombie_id: int):
	"""Видалення зомбі з системи"""
	zombie_positions.erase(zombie_id)
	flanking_positions.erase(zombie_id)
	attacking_zombies.erase(zombie_id)
func update_zombie_position(zombie_id: int, position: Vector3):
	"""Оновлення позиції зомбі"""
	zombie_positions[zombie_id] = position
# ============================================
# КООРДИНАЦІЯ АТАК
# ============================================
func request_flanking_position(zombie_id: int, target_pos: Vector3) -> Vector3:
	"""Запит на позицію для флангування"""
	# Перевіряємо скільки зомбі вже фланкують
	var occupied_angles: Array[float] = []
	for pos in flanking_positions.values():
		var angle = atan2(pos.z - target_pos.z, pos.x - target_pos.x)
		occupied_angles.append(angle)
	# Знаходимо вільний кут для флангування
	var best_angle: float = 0.0
	var max_distance: float = 0.0
	for test_angle in range(0, 360, 30):  # Перевіряємо кожні 30°
		var rad_angle = deg_to_rad(test_angle)
		var min_dist = INF
		# Знаходимо мінімальну відстань до зайнятих кутів
		for occupied in occupied_angles:
			var dist = abs(angle_difference(rad_angle, occupied))
			min_dist = min(min_dist, dist)
		# Вибираємо кут найдальший від зайнятих
		if min_dist > max_distance:
			max_distance = min_dist
			best_angle = rad_angle
	# Обчислюємо позицію
	var radius = 8.0  # Відстань флангування
	var flank_pos = target_pos + Vector3(
		cos(best_angle) * radius,
		0,
```

```
            sin(best_angle) * radius
      )
      flanking_positions[zombie_id] = flank_pos
      return flank_pos
func is_position_occupied(position: Vector3, radius: float = 2.0) -> bool:
      """Перевірка чи зайнята позиція іншим зомбі"""
      for zombie_pos in zombie_positions.values():
            if zombie_pos.distance_to(position) < radius:
                  return true
      return false
func get_zombie_count_near(position: Vector3, radius: float = 5.0) -> int:
      """Підрахунок зомбі поблизу позиції"""
      var count = 0
      for zombie_pos in zombie_positions.values():
            if zombie_pos.distance_to(position) < radius:
                  count += 1
      return count
# ============================================
# ТАКТИЧНИЙ АНАЛІЗ
# ============================================
func analyze_and_decide_strategy() -> String:
      """Аналіз ситуації і вибір стратегії"""
      var zombie_count = zombie_positions.size()
      if zombie_count == 0:
            return "DIRECT_ATTACK"
      # Якщо багато зомбі - оточуємо
      if zombie_count >= 3:
            return "SURROUND"
      # Якщо 2 зомбі - фланкуємо
      if zombie_count == 2:
            return "FLANK"
      # Інакше - пряма атака
      return "DIRECT_ATTACK"
func record_death_location(position: Vector3):
      """Запис місця смерті зомбі (небезпечна зона)"""
      dangerous_zones.append(position)
      # Обмежуємо розмір масиву
      if dangerous_zones.size() > 10:
            dangerous_zones.pop_front()
func is_dangerous_zone(position: Vector3, radius: float = 3.0) -> bool:
      """Перевірка чи є зона небезпечною"""
      for danger_pos in dangerous_zones:
            if danger_pos.distance_to(position) < radius:
                  return true
      return false
# ============================================
# TYPE III - КОЛЕКТИВНИЙ РОЗУМ
# ============================================
func register_type3_zombie(zombie_id: int, position: Vector3):
      """Реєстрація Type III зомбі в системі зграї"""
      type3_zombies[zombie_id] = {
            "position": position,
            "ready_to_attack": false,
            "surround_angle": 0.0,
            "assigned_position": Vector3.ZERO
      }
      # Також реєструємо в загальній системі
      register_zombie(zombie_id, position)
func unregister_type3_zombie(zombie_id: int):
      """Видалення Type III зомбі"""
      if type3_zombies.has(zombie_id):
            type3_zombies.erase(zombie_id)
      surround_positions.erase(zombie_id)
      unregister_zombie(zombie_id)
func update_type3_zombie(zombie_id: int, position: Vector3, ready: bool = false):
      """Оновлення статусу Type III зомбі"""
      if type3_zombies.has(zombie_id):
            type3_zombies[zombie_id]["position"] = position
```

```
                type3_zombies[zombie_id]["ready_to_attack"] = ready
        update_zombie_position(zombie_id, position)
func request_surround_position(zombie_id: int, target_pos: Vector3) -> Vector3:
        """Запит на позицію для оточення - КООРДИНОВАНЕ!"""
        var swarm_count = type3_zombies.size()
        if swarm_count == 0:
                return target_pos + Vector3(5, 0, 0)
        # Розподіляємо зомбі рівномірно по колу
        var angle_step = 360.0 / max(swarm_count, 1)
        var zombie_index = 0
        # Знаходимо індекс поточного зомбі
        var sorted_ids = type3_zombies.keys()
        sorted_ids.sort()
        zombie_index = sorted_ids.find(zombie_id)
        if zombie_index == -1:
                zombie_index = 0
        # Обчислюємо кут для цього зомбі
        var assigned_angle = deg_to_rad(angle_step * zombie_index)
        # Зберігаємо кут
        if type3_zombies.has(zombie_id):
                type3_zombies[zombie_id]["surround_angle"] = assigned_angle
        # Обчислюємо позицію на колі
        var surround_pos = target_pos + Vector3(
                cos(assigned_angle) * target_surround_radius,
                0,
                sin(assigned_angle) * target_surround_radius
        )
        surround_positions[zombie_id] = surround_pos
        if type3_zombies.has(zombie_id):
                type3_zombies[zombie_id]["assigned_position"] = surround_pos
        return surround_pos
func is_swarm_ready() -> bool:
        """Перевірка чи зграа готова до атаки"""
        var ready_count = 0
        for zombie_data in type3_zombies.values():
                if zombie_data["ready_to_attack"]:
                        ready_count += 1
        swarm_ready_count = ready_count
        return ready_count >= swarm_attack_threshold
func get_type3_count() -> int:
        """Кількість активних Type III зомбі"""
        return type3_zombies.size()
func get_surrounding_zombies_count() -> int:
        """Кількість Type III зомбі які ДОСЯГЛИ позиції окружения"""
        var count = 0
        for zombie_id in type3_zombies:
                var zombie_data = type3_zombies[zombie_id]
                var current_pos = zombie_data["position"]
                var assigned_pos = zombie_data["assigned_position"]
                # Перевіряємо чи зомбі ДОСЯГ своєї позиції (в радіусі 3м)
                if assigned_pos != Vector3.ZERO and current_pos.distance_to(assigned_pos)
< 3.0:
                        count += 1
        return count
func mark_ready_to_attack(zombie_id: int):
        """Позначити зомбі як готового до атаки"""
        if type3_zombies.has(zombie_id):
                type3_zombies[zombie_id]["ready_to_attack"] = true
func get_trap_position(zombie_id: int, target_pos: Vector3, obstacles: Array) ->
Vector3:
        """Знаходить позицію для запирання гравця між перешкодами"""
        if obstacles.is_empty():
                return request_surround_position(zombie_id, target_pos)
        # Шукаємо найближчі дві перешкоди
        var nearest_obstacles = []
        for obstacle in obstacles:
                if obstacle is StaticBody3D:
                        var dist = target_pos.distance_to(obstacle.global_position)
```

```
                  if dist < 20.0:  # В межах 20м
                        nearest_obstacles.append(obstacle)
      if nearest_obstacles.size() < 2:
            return request_surround_position(zombie_id, target_pos)
      # Сортуємо по відстані
      nearest_obstacles.sort_custom(func(a, b):
            return target_pos.distance_to(a.global_position) <
target_pos.distance_to(b.global_position)
      )
      # Позиціонуємо зомбі ЗА перешкодами (щоб загнати гравця між них)
      var obstacle1 = nearest_obstacles[0]
      var obstacle2 = nearest_obstacles[1]
      # Вектор від першої перешкоди до другої
      var between_vector =
obstacle1.global_position.direction_to(obstacle2.global_position)
      # Позиція: по іншу сторону від гравця
      var to_player = obstacle1.global_position.direction_to(target_pos)
      var trap_pos = obstacle1.global_position - to_player * 8.0
      return trap_pos
# ============================================
# УТИЛІТАРНІ ФУНКЦІЇ
# ============================================
func angle_difference(angle1: float, angle2: float) -> float:
      """Обчислення різниці між кутами"""
      var diff = angle1 - angle2
      while diff > PI:
            diff -= 2 * PI
      while diff < -PI:
            diff += 2 * PI
      return diff
func get_nearest_zombie_to(position: Vector3, exclude_id: int = -1) -> int:
      """Знаходження найближчого зомбі до позиції"""
      var min_dist = INF
      var nearest_id = -1
      for zombie_id in zombie_positions:
            if zombie_id == exclude_id:
                  continue
            var dist = zombie_positions[zombie_id].distance_to(position)
            if dist < min_dist:
                  min_dist = dist
                  nearest_id = zombie_id
      return nearest_id
func clear():
      """Очищення всіх даних"""
      flanking_positions.clear()
      attacking_zombies.clear()
      zombie_positions.clear()
      dangerous_zones.clear()
      safe_approach_vectors.clear()
```

fuzzy_logic.gd

```
extends RefCounted
class_name FuzzyLogic
# ============================================
# FUZZY LOGIC - НЕЧЁТКАЯ ЛОГИКА
# Используется для Type I зомби
# ============================================
# Функции принадлежности (Membership Functions)
class MembershipFunction:
      # Треугольная функция принадлежности
      static func triangular(value: float, a: float, b: float, c: float) -> float:
            if value <= a or value >= c:
                  return 0.0
            elif value == b:
                  return 1.0
            elif value < b:
                  return (value - a) / (b - a)
```

```
            else:
                  return (c - value) / (c - b)
      # Трапециевидная функция
      static func trapezoidal(value: float, a: float, b: float, c: float, d: float) ->
float:
            if value <= a or value >= d:
                  return 0.0
            elif value >= b and value <= c:
                  return 1.0
            elif value < b:
                  return (value - a) / (b - a)
            else:
                  return (d - value) / (d - c)
# Лингвистические переменные для расстояния
class DistanceFuzzy:
      static func very_close(distance: float) -> float:
            # Очень близко: 0-2 (полная), 2-5 (спад)
            return MembershipFunction.trapezoidal(distance, 0, 0, 2, 5)
      static func close(distance: float) -> float:
            # Близко: 3-7-12
            return MembershipFunction.triangular(distance, 3, 7, 12)
      static func medium(distance: float) -> float:
            # Средне: 10-15-20
            return MembershipFunction.triangular(distance, 10, 15, 20)
      static func far(distance: float) -> float:
            # Далеко: 18-25-∞
            return MembershipFunction.trapezoidal(distance, 18, 25, 100, 100)
# Лингвистические переменные для HP
class HealthFuzzy:
      static func critical(hp_percent: float) -> float:
            # Критическое: 0-20%
            return MembershipFunction.trapezoidal(hp_percent, 0, 0, 0.1, 0.25)
      static func low(hp_percent: float) -> float:
            # Низкое: 15-30-45%
            return MembershipFunction.triangular(hp_percent, 0.15, 0.3, 0.45)
      static func medium(hp_percent: float) -> float:
            # Среднее: 35-50-65%
            return MembershipFunction.triangular(hp_percent, 0.35, 0.5, 0.65)
      static func high(hp_percent: float) -> float:
            # Высокое: 60-100%
            return MembershipFunction.trapezoidal(hp_percent, 0.6, 0.75, 1.0, 1.0)
# Лингвистические переменные для направления игрока
class PlayerFacingFuzzy:
      static func facing_towards(dot: float) -> float:
            # Игрок смотрит на зомби: dot > 0.5
            return MembershipFunction.trapezoidal(dot, 0.3, 0.5, 1.0, 1.0)
      static func facing_sideways(dot: float) -> float:
            # Игрок смотрит в сторону: -0.3 < dot < 0.5
            return MembershipFunction.triangular(dot, -0.3, 0.1, 0.5)
      static func facing_away(dot: float) -> float:
            # Игрок отвернулся: dot < -0.3
            return MembershipFunction.trapezoidal(dot, -1.0, -1.0, -0.5, -0.3)
# Система нечёткого вывода
class FuzzyInferenceSystem:
      # Правила вывода для агрессивности
      static func calculate_aggression(distance: float, hp_percent: float, facing_dot:
float) -> float:
            var rules = []
            # Правило 1: Если HP высокое И расстояние близко → Высокая агрессия
            var r1 = min(HealthFuzzy.high(hp_percent), DistanceFuzzy.close(distance))
            rules.append({"value": r1, "output": 0.9})
            # Правило 2: Если HP низкое И расстояние близко → Средняя агрессия
            var r2 = min(HealthFuzzy.low(hp_percent), DistanceFuzzy.close(distance))
            rules.append({"value": r2, "output": 0.5})
            # Правило 3: Если HP критическое → Низкая агрессия
            var r3 = HealthFuzzy.critical(hp_percent)
            rules.append({"value": r3, "output": 0.2})
            # Правило 4: Если расстояние очень близко → Максимальная агрессия
```

```
            var r4 = DistanceFuzzy.very_close(distance)
            rules.append({"value": r4, "output": 1.0})
            # Правило 5: Если игрок отвернулся И HP среднее → Высокая агрессия
            var r5 = min(PlayerFacingFuzzy.facing_away(facing_dot),
HealthFuzzy.medium(hp_percent))
            rules.append({"value": r5, "output": 0.85})
            # Дефаззификация методом центра тяжести
            return defuzzify(rules)
      # Правила для осторожности
      static func calculate_caution(distance: float, hp_percent: float, facing_dot:
float) -> float:
            var rules = []
            # Правило 1: Если HP критическое → Максимальная осторожность
            var r1 = HealthFuzzy.critical(hp_percent)
            rules.append({"value": r1, "output": 1.0})
            # Правило 2: Если игрок смотрит на нас И HP низкое → Высокая осторожность
            var r2 = min(PlayerFacingFuzzy.facing_towards(facing_dot),
HealthFuzzy.low(hp_percent))
            rules.append({"value": r2, "output": 0.8})
            # Правило 3: Если расстояние среднее И HP низкое → Средняя осторожность
            var r3 = min(DistanceFuzzy.medium(distance), HealthFuzzy.low(hp_percent))
            rules.append({"value": r3, "output": 0.6})
            # Правило 4: Если HP высокое → Низкая осторожность
            var r4 = HealthFuzzy.high(hp_percent)
            rules.append({"value": r4, "output": 0.2})
            return defuzzify(rules)
      # Дефаззификация: центр тяжести (Centroid method)
      static func defuzzify(rules: Array) -> float:
            var numerator = 0.0
            var denominator = 0.0
            for rule in rules:
                  numerator += rule["value"] * rule["output"]
                  denominator += rule["value"]
            if denominator == 0.0:
                  return 0.0
            return numerator / denominator
# Принятие решения на основе нечёткой логики
static func make_decision(distance: float, hp_percent: float, facing_dot: float) ->
Dictionary:
      var aggression = FuzzyInferenceSystem.calculate_aggression(distance, hp_percent,
facing_dot)
      var caution = FuzzyInferenceSystem.calculate_caution(distance, hp_percent,
facing_dot)
      return {
            "aggression": aggression,
            "caution": caution,
            "action": "ATTACK" if aggression > caution else "RETREAT"
      }
```

goap.gd

```
extends RefCounted
class_name GOAP
# ============================================
# GOAP - GOAL-ORIENTED ACTION PLANNING
# Система планування дій на основі цілей для Type III зомбі
# ============================================
# Дія (Action) в GOAP
class GOAPAction:
      var name: String
      var cost: float
      var preconditions: Dictionary  # {state_key: required_value}
      var effects: Dictionary  # {state_key: result_value}
      var execute_func: Callable
      func _init(action_name: String, action_cost: float = 1.0):
            name = action_name
            cost = action_cost
            preconditions = {}
```

```
		effects = {}
	func add_precondition(key: String, value):
		preconditions[key] = value
		return self
	func add_effect(key: String, value):
		effects[key] = value
		return self
	func set_execute_func(func_ref: Callable):
		execute_func = func_ref
		return self
	func check_preconditions(state: Dictionary) -> bool:
		for key in preconditions:
			if not state.has(key) or state[key] != preconditions[key]:
				return false
		return true
	func apply_effects(state: Dictionary) -> Dictionary:
		var new_state = state.duplicate()
		for key in effects:
			new_state[key] = effects[key]
		return new_state
# Вузол у графі планування
class PlanNode:
	var state: Dictionary
	var action: GOAPAction
	var parent: PlanNode
	var g_cost: float  # Вартість від початку
	var h_cost: float  # Евристична вартість до мети
	var f_cost: float  # Сумарна вартість
	func _init(state_dict: Dictionary, parent_node: PlanNode = null, action_taken:
GOAPAction = null):
		state = state_dict
		parent = parent_node
		action = action_taken
		g_cost = 0.0
		h_cost = 0.0
		f_cost = 0.0
	func calculate_costs(goal: Dictionary):
		if parent:
			g_cost = parent.g_cost + (action.cost if action else 0.0)
		else:
			g_cost = 0.0
		# Евристика: кількість невідповідних цілей
		h_cost = 0.0
		for key in goal:
			if not state.has(key) or state[key] != goal[key]:
				h_cost += 1.0
		f_cost = g_cost + h_cost
# GOAP Planner - планувальник дій
class GOAPPlanner:
	var available_actions: Array[GOAPAction] = []
	func add_action(action: GOAPAction):
		available_actions.append(action)
	func plan(current_state: Dictionary, goal: Dictionary) -> Array[GOAPAction]:
		"""A* пошук плану дій"""
		var open_list: Array[PlanNode] = []
		var closed_list: Array[PlanNode] = []
		# Початковий вузол
		var start_node = PlanNode.new(current_state)
		start_node.calculate_costs(goal)
		open_list.append(start_node)
		var iterations = 0
		var max_iterations = 100  # Обмеження для запобігання зависання
		while open_list.size() > 0 and iterations < max_iterations:
			iterations += 1
			# Знаходимо вузол з найменшим f_cost
			var current_node = get_lowest_f_cost_node(open_list)
			open_list.erase(current_node)
			# Перевіряємо чи досягли мети
```

```
                  if is_goal_achieved(current_node.state, goal):
                        return reconstruct_plan(current_node)
                  closed_list.append(current_node)
                  # Розглядаємо всі можливі дії
                  for action in available_actions:
                        if not action.check_preconditions(current_node.state):
                              continue
                        # Застосовуємо дію
                        var new_state = action.apply_effects(current_node.state)
                        # Перевіряємо чи вже обробляли цей стан
                        if is_state_in_list(new_state, closed_list):
                              continue
                        var neighbor_node = PlanNode.new(new_state, current_node,
action)
                        neighbor_node.calculate_costs(goal)
                        # Перевіряємо чи є кращий шлях
                        var existing_node = find_node_with_state(new_state,
open_list)
                        if existing_node and existing_node.g_cost <=
neighbor_node.g_cost:
                              continue
                        open_list.append(neighbor_node)
            # План не знайдено
            return []
      func get_lowest_f_cost_node(nodes: Array[PlanNode]) -> PlanNode:
            var lowest = nodes[0]
            for node in nodes:
                  if node.f_cost < lowest.f_cost:
                        lowest = node
            return lowest
      func is_goal_achieved(state: Dictionary, goal: Dictionary) -> bool:
            for key in goal:
                  if not state.has(key) or state[key] != goal[key]:
                        return false
            return true
      func is_state_in_list(state: Dictionary, node_list: Array[PlanNode]) -> bool:
            for node in node_list:
                  if states_equal(node.state, state):
                        return true
            return false
      func find_node_with_state(state: Dictionary, node_list: Array[PlanNode]) ->
PlanNode:
            for node in node_list:
                  if states_equal(node.state, state):
                        return node
            return null
      func states_equal(state1: Dictionary, state2: Dictionary) -> bool:
            if state1.size() != state2.size():
                  return false
            for key in state1:
                  if not state2.has(key) or state1[key] != state2[key]:
                        return false
            return true
      func reconstruct_plan(goal_node: PlanNode) -> Array[GOAPAction]:
            var plan: Array[GOAPAction] = []
            var current = goal_node
            while current.parent != null:
                  plan.push_front(current.action)
                  current = current.parent
            return plan
# GOAP Agent - агент який використовує GOAP
class GOAPAgent:
      var planner: GOAPPlanner
      var current_plan: Array[GOAPAction] = []
      var current_action_index: int = 0
      var world_state: Dictionary = {}
      var current_goal: Dictionary = {}
      func _init():
```

```
            planner = GOAPPlanner.new()
      func add_action(action: GOAPAction):
            planner.add_action(action)
      func set_world_state(key: String, value):
            world_state[key] = value
      func get_world_state(key: String):
            return world_state.get(key, null)
      func set_goal(goal: Dictionary):
            current_goal = goal
            replan()
      func replan():
            """Перепланування дій"""
            current_plan = planner.plan(world_state, current_goal)
            current_action_index = 0
      func update() -> GOAPAction:
            """Оновлення агента - повертає поточну дію для виконання"""
            if current_plan.is_empty():
                  return null
            if current_action_index >= current_plan.size():
                  # План виконано
                  current_plan.clear()
                  return null
            return current_plan[current_action_index]
      func action_completed():
            """Повідомлення про завершення поточної дії"""
            if current_action_index < current_plan.size():
                  var completed_action = current_plan[current_action_index]
                  # Застосовуємо ефекти дії до світу
                  for key in completed_action.effects:
                        world_state[key] = completed_action.effects[key]
                  current_action_index += 1
      func needs_replan() -> bool:
            """Перевірка чи потрібне перепланування"""
            if current_plan.is_empty():
                  return true
            # Перевіряємо чи передумови поточної дії досі виконуються
            if current_action_index < current_plan.size():
                  var current_action = current_plan[current_action_index]
                  if not current_action.check_preconditions(world_state):
                        return true
            return false
```

zombie_type1.gd

```
extends CharacterBody3D
class_name ZombieType1
# ============================================
# TYPE I: FSM + FUZZY LOGIC - РЕАКТИВНИЙ ШТУЧНИЙ ІНТЕЛЕКТ
# ============================================
#
# МЕТОДИ ШТУЧНОГО ІНТЕЛЕКТУ:
#
# 1. FINITE STATE MACHINE (Скінчений автомат станів)
#    - Класичний метод AI для моделювання поведінки NPC
#    - Агент знаходиться в ОДНОМУ зі станів: IDLE, MOVE_TO_TARGET, ATTACK_TARGET,
RETREAT
#    - Переходи між станами відбуваються за чіткими ПРАВИЛАМИ (умовами)
#    - Це РЕАКТИВНИЙ AI: зомбі реагує на зовнішні стимули миттєво
#    - Плюси: швидкий, простий, передбачуваний
#    - Мінуси: негнучкий, поведінка "роботоподібна"
#
# 2. FUZZY LOGIC (Нечітка логіка)
#    - Метод AI для роботи з НЕВИЗНАЧЕНІСТЮ та НЕЧІТКИМИ поняттями
#    - Замість бінарних рішень (так/ні) використовуємо СТУПІНЬ ПРИНАЛЕЖНОСТІ (0.0-1.0)
#    - Приклад: замість "HP низьке" (так/ні), маємо "HP низьке на 70%" (0.7)
#    - FUZZY INFERENCE SYSTEM:
#      а) Фаззифікація: перетворення числових значень → fuzzy множини (low, medium,
high)
```

```
#      б) База правил: IF HP high AND distance close THEN aggression very_high
#      в) Дефаззифікація: перетворення fuzzy результату → конкретне число (швидкість)
#    - Це робить ПЛАВНІ, ПРИРОДНІ зміни поведінки замість різких стрибків
#
# ПАТТЕРН ПОВЕДІНКИ Type I - "ТУПИЙ АГРЕСОР":
# - Завжди йде ПРЯМО на гравця (без обходів, без хитрощів)
# - Швидкість руху ПЛАВНО змінюється через Fuzzy Logic (0.7x - 1.5x)
# - При критичному HP (<20%) Fuzzy система вирішує ВТІКАТИ
# - Простий, передбачуваний, але ЕФЕКТИВНИЙ завдяки високій швидкості
# ============================================
enum State {
      IDLE,
      MOVE_TO_TARGET,
      ATTACK_TARGET,
      RETREAT  # Новий стан: відступ при низькому HP
}
# Посилання
var player: CharacterBody3D
# Параметри
@export var max_hp: float = 150.0  # УВЕЛИЧЕНО для динамики
var hp: float = 150.0
@export var speed: float = 3.5  # АГРЕСИВНИЙ - швидкий
@export var damage: float = 15.0  # АГРЕСИВНИЙ - сильний
@export var attack_range: float = 1.5
# Навігація
var nav_agent: NavigationAgent3D
# FSM
var current_state: State = State.IDLE
var state_timer: float = 0.0
var idle_duration: float = 0.2  # Швидка реакція
var decision_interval: float = 0.5  # Рідше переоцінює - тупіше
# Fuzzy Logic - для прийняття рішень
var should_retreat: bool = false
var aggression_level: float = 0.5  # Рівень агресії (0-1)
var last_fuzzy_check: float = 0.0
var fuzzy_check_interval: float = 0.5
# Target
var current_target: Node3D = null
# Метрики
var spawn_time: float
var distance_traveled: float = 0.0
var damage_dealt_to_player: float = 0.0
# HP Bar
var hp_label: Label3D
# Animation
var animation_player: AnimationPlayer
func _ready():
      add_to_group("zombies")
      GameLogger.write_log("[TYPE I] Spawned at " + str(global_position))
      player = get_tree().get_first_node_in_group("player")
      spawn_time = Time.get_ticks_msec() / 1000.0
      if player:
            GameLogger.write_log("[TYPE I] Player found at " +
str(player.global_position))
      else:
            GameLogger.write_log("[TYPE I] ERROR: Player NOT found!")
      # Створюємо NavigationAgent3D для навігації
      nav_agent = NavigationAgent3D.new()
      nav_agent.path_desired_distance = 0.5
      nav_agent.target_desired_distance = attack_range
      nav_agent.radius = 0.5
      nav_agent.height = 1.8
      nav_agent.max_speed = speed
      add_child(nav_agent)
      # ВАЖЛИВО: Чекаємо синхронізацію навігації в Godot 4
      await get_tree().physics_frame
      GameLogger.write_log("[TYPE I] NavigationAgent ready")
      # Знайти AnimationPlayer
```

```
	animation_player = find_animation_player(self)
	# Колір Type I - зелений (ТУПИЙ АГРЕСОР)
	apply_color_to_model(Color(0.3, 1.0, 0.3))
	create_hp_bar()
	change_state(State.IDLE)
func create_hp_bar():
	hp_label = Label3D.new()
	hp_label.text = "HP: %d" % hp
	hp_label.modulate = Color.GREEN
	hp_label.font_size = 32
	hp_label.billboard = BaseMaterial3D.BILLBOARD_ENABLED
	hp_label.position = Vector3(0, 1.7, 0)  # Над капсулою (висота 1.5)
	hp_label.outline_size = 2
	add_child(hp_label)
func _physics_process(delta):
	state_timer += delta
	# Гравітація
	if not is_on_floor():
		velocity.y -= 20.0 * delta
	else:
		velocity.y = 0.0
	# FSM - виконуємо логіку поточного стану
	match current_state:
		State.IDLE:
			state_idle()
		State.MOVE_TO_TARGET:
			state_move_to_target(delta)
		State.ATTACK_TARGET:
			state_attack_target(delta)
		State.RETREAT:
			state_retreat(delta)
	move_and_slide()
	# Оновлюємо HP бар
	update_hp_bar()
func update_hp_bar():
	if not hp_label:
		return
	hp_label.text = "HP: %d" % hp
	var hp_percent = hp / max_hp
	if hp_percent > 0.6:
		hp_label.modulate = Color.GREEN
	elif hp_percent > 0.3:
		hp_label.modulate = Color.YELLOW
	else:
		hp_label.modulate = Color.RED
# ============================================
# FSM СТАНИ
# ============================================
func state_idle():
	"""ТУПИЙ АГРЕСОР - одразу атакує"""
	if state_timer >= idle_duration:
		select_nearest_target()
		if current_target:
			change_state(State.MOVE_TO_TARGET)
func state_move_to_target(delta):
	"""ТУПО ПРЕТЬСЯ ДО ЦІЛІ через навігацію"""
	# Валідація цілі
	if not current_target or not is_instance_valid(current_target):
		GameLogger.write_log("[TYPE I] Target lost!")
		change_state(State.IDLE)
		return
	# Fuzzy Logic: визначення агресивності та відступу
	var current_time = Time.get_ticks_msec() / 1000.0
	if current_time - last_fuzzy_check >= fuzzy_check_interval:
		last_fuzzy_check = current_time
		# Оцінюємо агресивність через Fuzzy Logic
		aggression_level = calculate_aggression_fuzzy()
		should_retreat = check_should_retreat_fuzzy()
```

```
            if should_retreat:
                  GameLogger.write_log("[TYPE I] RETREATING! HP: " + str(hp))
                  change_state(State.RETREAT)
                  return
      # Навігація до цілі
      nav_agent.target_position = current_target.global_position
      # Перевірка відстані
      var dist_to_target = global_position.distance_to(current_target.global_position)
      if dist_to_target < attack_range * 1.2:
            GameLogger.write_log("[TYPE I] REACHED PLAYER! Switching to ATTACK.
Distance: " + str(dist_to_target))
            change_state(State.ATTACK_TARGET)
            return
      # Прямий рух до цілі
      var direction = global_position.direction_to(current_target.global_position)
      # FUZZY LOGIC: швидкість залежить від рівня агресії
      # Агресивність 0.0-0.3 = повільно (0.7x)
      # Агресивність 0.3-0.7 = нормально (1.0x)
      # Агресивність 0.7-1.0 = швидко (1.5x)
      var speed_multiplier = 1.0
      if aggression_level > 0.7:
            speed_multiplier = 1.5  # Висока агресія - швидко
      elif aggression_level < 0.3:
            speed_multiplier = 0.7  # Низька агресія - повільно
      var current_speed = speed * speed_multiplier
      # ТУПО ПРЕТЬСЯ (ТІЛЬКИ XZ!)
      velocity.x = direction.x * current_speed
      velocity.z = direction.z * current_speed
      distance_traveled += Vector3(velocity.x, 0, velocity.z).length() * delta
func state_attack_target(delta):
      """ВТУПУЮ АТАКУЄ"""
      # Валідація цілі
      if not current_target or not is_instance_valid(current_target):
            change_state(State.IDLE)
            return
      var dist_to_target = global_position.distance_to(current_target.global_position)
      # Якщо ціль віддалилась - переслідує
      if dist_to_target > attack_range * 1.8:
            change_state(State.MOVE_TO_TARGET)
            return
      # ТУПО Б'Є - максимальний урон
      var saved_y = velocity.y
      velocity = Vector3.ZERO
      velocity.y = saved_y
      var dmg = damage * delta
      if current_target == player and player.has_method("take_damage"):
            player.take_damage(dmg)
            damage_dealt_to_player += dmg
            GameLogger.write_log("[TYPE I] ATTACKING! Damage: " + str(dmg) + " |
Distance: " + str(dist_to_target) + " | Player HP: " + str(player.hp))
func state_retreat(delta):
      """Відступ при КРИТИЧНОМУ HP (рідко)"""
      if not current_target or not is_instance_valid(current_target):
            change_state(State.IDLE)
            return
      # Перевіряємо чи ще треба втікати
      var current_time = Time.get_ticks_msec() / 1000.0
      if current_time - last_fuzzy_check >= fuzzy_check_interval:
            last_fuzzy_check = current_time
            should_retreat = check_should_retreat_fuzzy()
            if not should_retreat:
                  # HP підвищилося - повертаємось до атаки!
                  change_state(State.MOVE_TO_TARGET)
                  return
      # Втікаємо в протилежний бік (ТІЛЬКИ XZ!)
      var direction = current_target.global_position.direction_to(global_position)
      velocity.x = direction.x * speed * 1.4  # Трохи швидше при втечі
      velocity.z = direction.z * speed * 1.4
```

```
	distance_traveled += Vector3(velocity.x, 0, velocity.z).length() * delta
# ============================================
# FSM ДОПОМІЖНІ МЕТОДИ
# ============================================
func change_state(new_state: State):
	"""Перехід між станами FSM"""
	current_state = new_state
	state_timer = 0.0
	# Оновлюємо анімацію згідно стану
	match new_state:
		State.IDLE:
			play_animation("IDLE")
		State.MOVE_TO_TARGET:
			play_animation("WALK")
		State.ATTACK_TARGET:
			play_animation("ATK")
		State.RETREAT:
			play_animation("WALK")  # Використовуємо WALK для втечі
func calculate_aggression_fuzzy() -> float:
	"""
	FUZZY INFERENCE SYSTEM - система нечіткого виводу
	Це СЕРЦЕ штучного інтелекту Type I. Fuzzy Logic дозволяє NPC приймати
	ПЛАВНІ, ПРИРОДНІ рішення замість різких бінарних переходів.
	ЕТАПИ FUZZY INFERENCE:
	1. ФАЗЗИФІКАЦІЯ (Fuzzification):
	   Перетворення чітких числових значень → нечіткі лінгвістичні змінні
	   Приклад: HP=75% → "high HP" з ступенем приналежності 0.6
	2. БАЗА ПРАВИЛ (Rule Base):
	   Набір IF-THEN правил, що моделюють ЕКСПЕРТНІ ЗНАННЯ про поведінку
	   Правила комбінуються через операції min (AND) та max (OR)
	3. ДЕФАЗЗИФІКАЦІЯ (Defuzzification):
	   Перетворення нечіткого результату → чітке число для керування
	   Використовуємо метод "максимуму" (вибір найбільш активованого правила)
	РЕЗУЛЬТАТ: агресивність від 0.0 (пасивний) до 1.0 (максимальна атака)
	Це значення потім використовується для ПЛАВНОГО регулювання швидкості руху!
	"""
	if not player or not is_instance_valid(player):
		return 0.5
	var hp_percent = hp / max_hp
	var distance = global_position.distance_to(player.global_position)
	# ========== ЕТАП 1: ФАЗЗИФІКАЦІЯ ==========
	# Перетворюємо чіткі значення HP та Distance → fuzzy множини
	# Кожна функція приналежності повертає значення 0.0-1.0 (ступінь приналежності)
	var hp_high = FuzzyLogic.HealthFuzzy.high(hp_percent)      # Висока функція
приналежності (60-100% HP)
	var hp_medium = FuzzyLogic.HealthFuzzy.medium(hp_percent)  # Середня (35-65% HP)
	var hp_low = FuzzyLogic.HealthFuzzy.low(hp_percent)        # Низька (15-45% HP)
	var dist_close = FuzzyLogic.DistanceFuzzy.close(distance)   # Близька відстань
(2-8m)
	var dist_medium = FuzzyLogic.DistanceFuzzy.medium(distance) # Середня відстань
(5-15m)
	# ========== ЕТАП 2: БАЗА ПРАВИЛ (INFERENCE) ==========
	# Кожне правило моделює СИТУАЦІЮ і відповідний рівень агресії
	# min() реалізує операцію AND (обидві умови мають бути правдивими)
	# Результат множимо на цільовий рівень агресії (0.3-0.9)
	# ПРАВИЛО 1: "Якщо багато HP і близько до гравця → ДУЖЕ агресивний" (0.9)
	var very_high_aggression = min(hp_high, dist_close) * 0.9
	# ПРАВИЛО 2: "Якщо середнє HP і близько → агресивний" (0.7)
	var high_aggression = min(hp_medium, dist_close) * 0.7
	# ПРАВИЛО 3: "Якщо мало HP → обережний" (0.3)
	var low_aggression = hp_low * 0.3
	# ПРАВИЛО 4: "Якщо середня відстань → помірна агресія" (0.5)
	var medium_aggression = dist_medium * 0.5
	# ========== ЕТАП 3: ДЕФАЗЗИФІКАЦІЯ ==========
	# Об'єднуємо всі активовані правила через max() (OR)
	# Вибираємо НАЙБІЛЬШ АКТИВОВАНЕ правило як фінальний результат
	var aggression = max(very_high_aggression, max(high_aggression,
max(low_aggression, medium_aggression)))
```

```
	return aggression  # Повертаємо чітке значення агресії 0.0-1.0
func check_should_retreat_fuzzy() -> bool:
	"""Fuzzy Logic: чи відступати при критичному HP?"""
	if not player or not is_instance_valid(player):
		return false
	var hp_percent = hp / max_hp
	var distance = global_position.distance_to(player.global_position)
	# Fuzzy Logic тільки для HP критичності
	var hp_critical = FuzzyLogic.HealthFuzzy.critical(hp_percent)  # 0-20% HP
	var hp_low = FuzzyLogic.HealthFuzzy.low(hp_percent)  # 15-45% HP
	# Якщо дуже близько до гравця - відступаємо навіть при низькому HP
	var very_close = FuzzyLogic.DistanceFuzzy.very_close(distance)
	# Розрахунок терміновості втечі
	var retreat_urgency = max(hp_critical, min(hp_low, very_close))
	# Відступаємо ТІЛЬКИ якщо терміновість > 0.6
	return retreat_urgency > 0.6
func select_nearest_target():
	"""Вибір цілі - завжди гравець"""
	if not player or not is_instance_valid(player):
		GameLogger.write_log("[TYPE I] Target search failed - player not found!")
		current_target = null
		return
	current_target = player
	var dist = global_position.distance_to(player.global_position)
	GameLogger.write_log("[TYPE I] Found player! Distance: " + str(dist))
func take_damage(amount: float):
	hp -= amount
	play_animation("HIT")
	if hp <= 0:
		die()
func die():
	play_animation("DEAD")
	await get_tree().create_timer(2.0).timeout  # Чекаємо поки анімація смерті
завершиться
	var game_manager = get_tree().get_first_node_in_group("game_manager")
	if game_manager:
		game_manager.on_zombie_death(self)
	queue_free()
# ==========================================
# АНІМАЦІЇ
# ==========================================
func find_animation_player(node: Node) -> AnimationPlayer:
	"""Рекурсивно шукає AnimationPlayer в дереві"""
	if node is AnimationPlayer:
		return node
	for child in node.get_children():
		var result = find_animation_player(child)
		if result:
			return result
	return null
func play_animation(anim_name: String):
	"""Програє анімацію якщо є AnimationPlayer"""
	if animation_player and animation_player.has_animation(anim_name):
		animation_player.play(anim_name)
func apply_color_to_model(color: Color):
	"""Застосовує колір до всіх mesh в моделі"""
	for child in get_children():
		apply_color_recursive(child, color)
func apply_color_recursive(node: Node, color: Color):
	"""Рекурсивно шукає MeshInstance3D і застосовує колір"""
	if node is MeshInstance3D:
		# Створюємо новий матеріал з модуляцією
		var material = StandardMaterial3D.new()
		material.albedo_color = color
		node.set_surface_override_material(0, material)
	for child in node.get_children():
		apply_color_recursive(child, color)
```

zombie_type2.gd

```
extends CharacterBody3D
class_name ZombieType2
# ========================================
# TYPE II: BEHAVIOR TREE - ДЕЛІБЕРАТИВ НИЙ ШТУЧНИЙ ІНТЕЛЕКТ
# ========================================
#
# МЕТОДИ ШТУЧНОГО ІНТЕЛЕКТУ:
#
# 1. BEHAVIOR TREE (Дерево поведінки)
#    - ІЄРАРХІЧНА структура прийняття рішень
#    - На відміну від FSM (плоский список станів), BT має ДЕРЕВО вузлів
#    - Кожен вузол повертає: SUCCESS, FAILURE, або RUNNING
#    - Виконання починається з КОРЕНЯ і йде вниз по дереву
#
#    ТИПИ ВУЗЛІВ:
#    а) BTSelector (OR-вузол, "?"):
#       - Виконує дочірні вузли ЗЛІВА направо
#       - Повертає SUCCESS якщо ХОЧА Б ОДИН дочірній успішний
#       - Зупиняється на першому SUCCESS (пріоритетний вибір)
#
#    б) BTSequence (AND-вузол, "→"):
#       - Виконує дочірні вузли ЗЛІВА направо
#       - Повертає SUCCESS тільки якщо ВСІ дочірні успішні
#       - Зупиняється на першому FAILURE
#
#    в) BTCondition (Умова):
#       - Перевіряє умову → SUCCESS або FAILURE
#       - Не змінює стан гри, тільки ЧИТАЄ
#
#    г) BTAction (Дія):
#       - Виконує дію → SUCCESS, FAILURE, або RUNNING (в процесі)
#       - ЗМІНЮЄ стан гри (рух, атака, тощо)
#
# 2. VECTOR GEOMETRY (Векторна геометрія)
#    - DOT PRODUCT (Скалярний добуток) для визначення кутів:
#      • a · b = |a| * |b| * cos(θ)
#      • Якщо a та b нормалізовані: a · b = cos(θ)
#      • dot > 0.8 → кут < 36° (майже паралельні, йдемо в лоб)
#      • dot > 0.5 → кут < 60° (гравець дивиться на зомбі)
#      • dot < 0 → кут > 90° (протилежні напрямки)
#
# 3. STEALTH MECHANIC (Механіка скрадання)
#    - NPC АДАПТУЄ ШВИДКІСТЬ залежно від уваги гравця:
#      • Гравець дивиться (dot > 0.5) → швидкість × 0.3 (крадеться)
#      • Гравець НЕ дивиться → швидкість × 1.2 (швидко обходить)
#    - Ефект "Weeping Angels" (з Doctor Who) - рухається тільки коли не бачать
#
# 4. BLACKBOARD PATTERN
#    - Загальна пам'ять для координації між NPC
#    - Зомбі публікують свої позиції, читають позиції інших
#    - Дозволяє уникати колізій, координувати атаки
#
# ПАТЕРН ПОВЕДІНКИ TYPE II - "ХИТРИЙ ХИЩНИК":
# - НИКОГДА не йде в лоб (head-on check через dot product)
# - Завжди намагається атакувати ЗІ СПИНИ (backstab × 2 damage)
# - Використовує ШИРОКИЙ ОБХІД якщо виявляє лобовий підхід
# - КРАДЕТЬСЯ коли гравець дивиться (stealth mechanic)
# ========================================
# Посилання
var player: CharacterBody3D
var behavior_tree: BTNode
var blackboard: Blackboard
var zombie_id: int
# Параметри
@export var max_hp: float = 200.0  # УВЕЛИЧЕНО для динамики
var hp: float = 200.0
```

```
@export var speed: float = 2.5  # Хитрість > швидкість
@export var damage: float = 20.0  # СИЛЬНИЙ удар зі спини
@export var attack_range: float = 1.5
# Навігація
var nav_agent: NavigationAgent3D
# ПІДЛА КРИСАЧА - тактика
var flanking_position: Vector3 = Vector3.ZERO
var hiding_position: Vector3 = Vector3.ZERO
var is_player_facing_me: bool = true  # Чи дивиться гравець на мене
var backstab_angle_threshold: float = 100.0  # Кут для атаки ззаду (градуси)
# Логування (щоб не спамити кожен кадр)
var last_log_time: float = 0.0
var log_interval: float = 2.0  # Логувати раз в 2 секунди
# Метрики
var spawn_time: float
var distance_traveled: float = 0.0
var damage_dealt_to_player: float = 0.0
# HP Bar
var hp_label: Label3D
# Animation
var animation_player: AnimationPlayer
func _ready():
	add_to_group("zombies")  # Додаємо до групи зомбі
	GameLogger.write_log("[TYPE II] Spawned at " + str(global_position))
	player = get_tree().get_first_node_in_group("player")
	spawn_time = Time.get_ticks_msec() / 1000.0
	if player:
		GameLogger.write_log("[TYPE II] Player found at " +
str(player.global_position))
	else:
		GameLogger.write_log("[TYPE II] ERROR: Player NOT found!")
	# Унікальний ID для координації
	zombie_id = get_instance_id()
	# NavigationAgent для хитрого обходу
	nav_agent = NavigationAgent3D.new()
	nav_agent.path_desired_distance = 0.5
	nav_agent.target_desired_distance = 1.0
	nav_agent.radius = 0.5
	nav_agent.height = 1.8
	nav_agent.max_speed = speed
	nav_agent.avoidance_enabled = true
	add_child(nav_agent)
	# ВАЖЛИВО: Чекаємо синхронізацію навігації в Godot 4
	await get_tree().physics_frame
	GameLogger.write_log("[TYPE II] NavigationAgent ready")
	# Підключення до Blackboard
	blackboard = Blackboard.get_instance()
	blackboard.register_zombie(zombie_id, global_position)
	# Побудова дерева поведінки
	behavior_tree = build_behavior_tree()
	# Знайти AnimationPlayer в дочірніх нодах (якщо є модель)
	animation_player = find_animation_player(self)
	# Колір Type II - помаранчевий
	apply_color_to_model(Color(1.0, 0.6, 0.0))
	create_hp_bar()
func create_hp_bar():
	hp_label = Label3D.new()
	hp_label.text = "HP: %d" % hp
	hp_label.modulate = Color(1, 0.5, 0)  # Помаранчевий
	hp_label.font_size = 32
	hp_label.billboard = BaseMaterial3D.BILLBOARD_ENABLED
	hp_label.position = Vector3(0, 1.7, 0)  # Над капсулою
	hp_label.outline_size = 2
	add_child(hp_label)
func _physics_process(delta):
	# Гравітація
	if not is_on_floor():
		velocity.y -= 20.0 * delta
```

```
	else:
		velocity.y = 0.0
	# Оновлюємо Blackboard інформацією про себе
	blackboard.update_zombie_position(zombie_id, global_position)
	# Оновлюємо Blackboard інформацією про гравця
	if player and is_instance_valid(player):
		var player_facing = -player.global_transform.basis.z
		blackboard.update_player_info(player.global_position, player_facing)
	# Виконуємо Behavior Tree
	behavior_tree.tick()
	move_and_slide()
	distance_traveled += velocity.length() * delta
	# Оновлюємо HP бар
	update_hp_bar()
func update_hp_bar():
	if not hp_label:
		return
	hp_label.text = "HP: %d" % hp
	var hp_percent = hp / max_hp
	if hp_percent > 0.6:
		hp_label.modulate = Color(1, 0.5, 0)
	elif hp_percent > 0.3:
		hp_label.modulate = Color.YELLOW
	else:
		hp_label.modulate = Color.RED
# ============================================
# BEHAVIOR TREE КОНСТРУКТОР
# ============================================
func build_behavior_tree() -> BTNode:
	"""
	ПОБУДОВА BEHAVIOR TREE - це СЕРЦЕ AI для Type II
	СТРУКТУРА ДЕРЕВА (зверху → вниз, ПРІОРИТЕТИ):
	ROOT: BTSelector (OR-логіка, вибирає ПЕРШИЙ успішний)
	├─[1] BTSequence: "BACKSTAB" (найвищий пріоритет!)
	│	├── BTCondition: am_behind_player? (DOT PRODUCT: кут > 100°)
	│	└── BTAction: backstab_attack (× 2 урон!)
	├─[2] BTSequence: "WIDE FLANK" (уникаємо лобової атаки)
	│	├── BTCondition: is_approaching_head_on? (DOT PRODUCT: > 0.8)
	│	└── BTAction: wide_flank_around (обходимо ШИРОКО 8м)
	└─[3] BTAction: "MOVE BEHIND" (за замовчуванням - крадеться ззаду)
		└── action_move_behind_player (× 0.3 швидкості якщо бачать)
	ЛОГІКА ВИКОНАННЯ (кожен tick):
	1. Selector пробує [1] → якщо SUCCESS - зупиняється
	2. Якщо [1] FAILURE → пробує [2]
	3. Якщо [2] FAILURE → виконує [3] (завжди RUNNING/SUCCESS)
	РЕЗУЛЬТАТ: Зомбі ЗАВЖДИ намагається зайти ззаду, НИКОГДА не йде в лоб!
	"""
	return BTNode.BTSelector.new([
		build_backstab_sequence(),
		build_wide_flank_sequence(),  # УНИКАННЯ лобової атаки
		BTNode.BTAction.new(action_move_behind_player)
	])
func build_backstab_sequence() -> BTNode:
	"""Атака ЗІ СПИНИ - тільки коли за спиною"""
	return BTNode.BTSequence.new([
		BTNode.BTCondition.new(condition_am_behind_player),
		BTNode.BTAction.new(action_backstab_attack)
	])
func build_wide_flank_sequence() -> BTNode:
	"""ШИРОКИЙ ОБХІД - якщо йду в лоб на гравця (НИКОГДА не в лоб!)"""
	return BTNode.BTSequence.new([
		BTNode.BTCondition.new(condition_is_approaching_head_on),
		BTNode.BTAction.new(action_wide_flank_around)
	])
# ============================================
# BEHAVIOR TREE УМОВИ (CONDITIONS)
# ============================================
func condition_am_behind_player() -> bool:
```

```
	"""Чи я ЗА СПИНОЮ гравця для BACKSTAB?"""
	if not player or not is_instance_valid(player):
		return false
	var player_facing = -player.global_transform.basis.z
	var to_zombie = (global_position - player.global_position).normalized()
	var dot = player_facing.dot(to_zombie)
	var angle = rad_to_deg(acos(clamp(dot, -1.0, 1.0)))
	# За спиною якщо кут > 100° AND близько
	var behind = angle > backstab_angle_threshold
	var close = global_position.distance_to(player.global_position) < attack_range *
1.5
	return behind and close
func condition_player_not_looking() -> bool:
	"""Чи гравець НЕ дивиться на мене? (можна рухатись)"""
	if not player or not is_instance_valid(player):
		return true
	var player_facing = -player.global_transform.basis.z
	var to_zombie = (global_position - player.global_position).normalized()
	var dot = player_facing.dot(to_zombie)
	# Гравець НЕ дивиться якщо dot < 0.3
	is_player_facing_me = dot > 0.3
	return not is_player_facing_me
func condition_is_approaching_head_on() -> bool:
	"""
	ВЕКТОРНА ГЕОМЕТРІЯ - перевірка чи йду я ПРЯМО в лоб на гравця?
	МЕТОД ШТУЧНОГО ІНТЕЛЕКТУ: DOT PRODUCT (скалярний добуток векторів)
	МАТЕМАТИКА:
	  a · b = |a| * |b| * cos(θ)
	  Для нормалізованих векторів: a · b = cos(θ)
	ЗАСТОСУВАННЯ:
	  1) Вектор МОЄ РУХУ (velocity) · Вектор ДО ГРАВЦЯ (to_player)
	     → dot > 0.8 означає кут < 36° → йду МАЙЖЕ ПРЯМО на гравця
	  2) Вектор ПОГЛЯД ГРАВЦЯ (player_facing) · Вектор ДО МЕНЕ (to_zombie)
	     → facing_dot > 0.5 означає кут < 60° → я ПОПЕРЕДУ гравця
	  3) Комбінація: dot > 0.8 AND facing_dot > 0.5
	     → я йду ПРЯМО в лоб на гравця (ПОГАНО для хитрого хищника!)
	РЕЗУЛЬТАТ: якщо TRUE → Behavior Tree запускає WIDE FLANK (широкий обхід)
	"""
	if not player or not is_instance_valid(player):
		return false
	# ========== ВЕКТОР 1: Напрямок руху зомбі ==========
	var to_player = (player.global_position - global_position).normalized()
	var my_movement = Vector3(velocity.x, 0, velocity.z).normalized()
	# Якщо швидкість нульова - використовуємо напрямок до гравця
	if my_movement.length() < 0.1:
		my_movement = to_player
	# ========== DOT PRODUCT #1: Чи рухаюсь я ДО гравця? ==========
	var dot = my_movement.dot(to_player)
	# dot > 0.8 → кут < 36° → йду МАЙЖЕ ПРЯМО на гравця
	# ========== ВЕКТОР 2: Напрямок погляду гравця ==========
	var player_facing = -player.global_transform.basis.z
	var to_zombie = (global_position - player.global_position).normalized()
	# ========== DOT PRODUCT #2: Чи я ПОПЕРЕДУ гравця? ==========
	var facing_dot = player_facing.dot(to_zombie)
	# facing_dot > 0.5 → кут < 60° → я в ПОЛІ ЗОРУ гравця попереду
	# ========== КОМБІНАЦІЯ: Йду в лоб? ==========
	var is_head_on = dot > 0.8 and facing_dot > 0.5
	# TRUE = йду прямо на гравця + я попереду нього = ЛОБОВА АТАКА (погано!)
	# Логуємо тільки раз в N секунд
	if is_head_on:
		var current_time = Time.get_ticks_msec() / 1000.0
		if current_time - last_log_time >= log_interval:
			GameLogger.write_log("[TYPE II] HEAD-ON APPROACH DETECTED!
Switching to WIDE FLANK. dot=" + str(dot))
			last_log_time = current_time
	return is_head_on
# ==========================================
# BEHAVIOR TREE ДІЇ (ACTIONS)
```

```
# ========================================
func action_backstab_attack() -> BTNode.Status:
	"""BACKSTAB - атака ЗІ СПИНИ з подвоєним уроном!"""
	if not player or not is_instance_valid(player):
		return BTNode.Status.FAILURE
	# ВАЖЛИВО: зберігаємо Y для гравітації
	var saved_y = velocity.y
	velocity = Vector3.ZERO
	velocity.y = saved_y
	play_animation("ATK")
	# ПОДВОЄНИЙ УРОН ЗА СПИНОЮ!
	var backstab_damage = damage * 2.0 * get_physics_process_delta_time()
	if player and is_instance_valid(player) and player.has_method("take_damage"):
		player.take_damage(backstab_damage)
		damage_dealt_to_player += backstab_damage
		GameLogger.write_log("[TYPE II] BACKSTAB! Damage: " +
str(backstab_damage) + " | Player HP: " + str(player.hp))
	else:
		GameLogger.write_log("[TYPE II] ERROR: Cannot attack player!")
	return BTNode.Status.SUCCESS
func action_move_behind_player() -> BTNode.Status:
	"""
	STEALTH MECHANIC - скрадання ззаду з адаптивною швидкістю
	МЕТОД ШТУЧНОГО ІНТЕЛЕКТУ: ADAPTIVE BEHAVIOR (адаптивна поведінка)
	NPC ЗМІНЮЄ ШВИДКІСТЬ залежно від УВАГИ гравця:
	  • Гравець ДИВИТЬСЯ (dot > 0.5) → швидкість × 0.3 (ПОВІЛЬНО крадеться)
	  • Гравець НЕ ДИВИТЬСЯ → швидкість × 1.2 (ШВИДКО обходить)
	Ефект "WEEPING ANGELS" (з Doctor Who):
	  - Монстр рухається ТІЛЬКИ коли гравець НЕ дивиться
	  - Створює напругу: гравець відвертається → зомбі СТРИБКОМ наближається
	  - Реалізовано через DOT PRODUCT для визначення напрямку погляду
	ГЕОМЕТРІЯ:
	  1) Визначаємо СТОРОНУ обходу (ліво/право) через player_right · to_zombie
	  2) Обчислюємо ПОЗИЦІЮ ФЛАНКУВАННЯ: 4м збоку + 2м ззаду
	  3) Рухаємось до позиції з АДАПТИВНОЮ швидкістю
	"""
	if not player or not is_instance_valid(player):
		return BTNode.Status.FAILURE
	# ========== ЕТАП 1: Визначаємо СТОРОНУ обходу ==========
	var player_facing = -player.global_transform.basis.z
	var player_right = player.global_transform.basis.x
	# З якої сторони я зараз відносно гравця?
	var to_zombie = (global_position - player.global_position).normalized()
	var side_dot = player_right.dot(to_zombie)
	# side_dot < 0 → я ЗЛІВА, side_dot > 0 → я СПРАВА
	# ========== ЕТАП 2: Обчислюємо ЦІЛЬОВУ позицію для фланкування ==========
	# Обираємо ПРОТИЛЕЖНУ сторону для обходу
	var flank_side = player_right if side_dot < 0 else -player_right
	var flank_offset = flank_side * 4.0 - player_facing * 2.0  # 4м збоку, 2м ззаду
	flanking_position = player.global_position + flank_offset
	# Перевірка чи досягли позиції
	var dist_to_flank = global_position.distance_to(flanking_position)
	if dist_to_flank < 2.0:
		return BTNode.Status.SUCCESS
	# ========== ЕТАП 3: Перевіряємо чи гравець ДИВИТЬСЯ на мене ==========
	var facing_dot = player_facing.dot(to_zombie)
	var player_looking_at_me = facing_dot > 0.5  # DOT > 0.5 → кут < 60°
	# Напрямок руху до цільової позиції
	var direction = global_position.direction_to(flanking_position)
	# ========== ЕТАП 4: АДАПТИВНА ШВИДКІСТЬ (STEALTH!) ==========
	var move_speed = speed
	if player_looking_at_me:
		# ДИВИТЬСЯ → МАЙЖЕ НЕ РУХАЮСЬ (0.3x)
		move_speed = speed * 0.3
		play_animation("IDLE")  # Анімація стояння
	else:
		# НЕ ДИВИТЬСЯ → ШВИДКО ОБХОДЖУ (1.2x)
		move_speed = speed * 1.2
```

```
            play_animation("WALK")
      # Застосовуємо швидкість (тільки XZ, Y для гравітації)
      velocity.x = direction.x * move_speed
      velocity.z = direction.z * move_speed
      return BTNode.Status.RUNNING
func action_wide_flank_around() -> BTNode.Status:
      """ШИРОКИЙ ОБХІД - якщо йду в лоб (НИКОГДА не атакует прямо!)"""
      if not player or not is_instance_valid(player):
            return BTNode.Status.FAILURE
      # ГЕОМЕТРІЯ: Обираємо ШИРОКИЙ обхід збоку (8м замість 4м!)
      var player_facing = -player.global_transform.basis.z
      var player_right = player.global_transform.basis.x
      # Визначаємо з якої сторони ми зараз
      var to_zombie = (global_position - player.global_position).normalized()
      var side_dot = player_right.dot(to_zombie)
      # ШИРОКИЙ фланг: 8м збоку + 4м ззаду (щоб НЕ йти в лоб!)
      var flank_side = player_right if side_dot < 0 else -player_right
      var flank_offset = flank_side * 8.0 - player_facing * 4.0  # ШИРОКО!
      flanking_position = player.global_position + flank_offset
      # Логуємо рідко
      var current_time = Time.get_ticks_msec() / 1000.0
      if current_time - last_log_time >= log_interval:
            GameLogger.write_log("[TYPE II] WIDE FLANK! Avoiding head-on. Side: " +
("RIGHT" if side_dot < 0 else "LEFT"))
            last_log_time = current_time
      # Перевірка чи досягли позиції
      var dist_to_flank = global_position.distance_to(flanking_position)
      if dist_to_flank < 2.5:
            return BTNode.Status.SUCCESS
      # Прямий рух до позиції
      var direction = global_position.direction_to(flanking_position)
      # ВАЖЛИВО: тільки XZ, Y залишаємо для гравітації
      velocity.x = direction.x * speed
      velocity.z = direction.z * speed
      play_animation("WALK")
      return BTNode.Status.RUNNING
func action_hide_and_wait() -> BTNode.Status:
      """ХОВАЄТЬСЯ за перешкодами коли гравець дивиться"""
      if not player or not is_instance_valid(player):
            return BTNode.Status.FAILURE
      # Шукаємо найближчу перешкоду
      var obstacles = get_tree().get_nodes_in_group("obstacles")
      var nearest_obstacle: Node3D = null
      var min_dist = INF
      for obstacle in obstacles:
            if obstacle is StaticBody3D:
                  var dist = global_position.distance_to(obstacle.global_position)
                  if dist < min_dist:
                        nearest_obstacle = obstacle
                        min_dist = dist
      if nearest_obstacle and min_dist < 8.0:
            # Ховаємось ЗА перешкодою
            var to_player =
nearest_obstacle.global_position.direction_to(player.global_position)
            hiding_position = nearest_obstacle.global_position - to_player * 2.0
            nav_agent.target_position = hiding_position
            if global_position.distance_to(hiding_position) < 1.0:
                  # Сховався - стоїть
                  var saved_y = velocity.y
                  velocity = Vector3.ZERO
                  velocity.y = saved_y
                  play_animation("IDLE")
                  return BTNode.Status.RUNNING
            # Прямий рух до схованки
            var direction = global_position.direction_to(hiding_position)
            # ВАЖЛИВО: тільки XZ
            velocity.x = direction.x * speed * 0.7  # Повільніше при скраданні
            velocity.z = direction.z * speed * 0.7
```

```
            play_animation("WALK")
      else:
            # Нема перешкод - просто відступає
            var away_dir = global_position.direction_to(player.global_position) * -
1.0
            # ВАЖЛИВО: тільки XZ
            velocity.x = away_dir.x * speed * 0.5
            velocity.z = away_dir.z * speed * 0.5
            play_animation("WALK")
      return BTNode.Status.RUNNING
func take_damage(amount: float):
      hp -= amount
      play_animation("HIT")
      print("Зомбі Type II отримав урон: ", amount, " HP: ", hp)
      if hp <= 0:
            die()
func die():
      play_animation("DEAD")
      # Повідомляємо Blackboard про смерть
      blackboard.unregister_zombie(zombie_id)
      blackboard.record_death_location(global_position)
      await get_tree().create_timer(2.0).timeout  # Чекаємо поки анімація смерті
завершиться
      var game_manager = get_tree().get_first_node_in_group("game_manager")
      if game_manager:
            game_manager.on_zombie_death(self)
      queue_free()
# ==========================================
# АНІМАЦІЇ
# ==========================================
func find_animation_player(node: Node) -> AnimationPlayer:
      """Рекурсивно шукає AnimationPlayer в дереві"""
      if node is AnimationPlayer:
            return node
      for child in node.get_children():
            var result = find_animation_player(child)
            if result:
                  return result
      return null
func play_animation(anim_name: String):
      """Програє анімацію якщо є AnimationPlayer"""
      if animation_player and animation_player.has_animation(anim_name):
            animation_player.play(anim_name)
func apply_color_to_model(color: Color):
      """Застосовує колір до всіх mesh в моделі"""
      for child in get_children():
            apply_color_recursive(child, color)
func apply_color_recursive(node: Node, color: Color):
      """Рекурсивно шукає MeshInstance3D і застосовує колір"""
      if node is MeshInstance3D:
            # Створюємо новий матеріал з модуляцією
            var material = StandardMaterial3D.new()
            material.albedo_color = color
            node.set_surface_override_material(0, material)
      for child in node.get_children():
            apply_color_recursive(child, color)
```

zombie_type3.gd

```
extends CharacterBody3D
class_name ZombieType3
# ==========================================
# TYPE III: GOAP + BLACKBOARD - ГІБРИДНИЙ ШТУЧНИЙ ІНТЕЛЕКТ
# ==========================================
# МЕТОДИ ШТУЧНОГО ІНТЕЛЕКТУ:
#
# 1. GOAP (Goal-Oriented Action Planning) - Планування на основі цілей
#    - АЛГОРИТМ: A* (A-star) для пошуку ОПТИМАЛЬНОГО ПЛАНУ дій
```

```
#    - На відміну від BT (виконує зверху вниз), GOAP ПЛАНУЄ задом наперед
#    ПРОЦЕС ПЛАНУВАННЯ:
#    а) Є МЕТА (Goal): {"attacking": true}
#    б) Є ПОТОЧНИЙ СТАН СВІТУ (World State): {"gathered": false, "surrounding": false}
#    в) Є НАБІР ДІЙ (Actions) з ПЕРЕДУМОВАМИ (Preconditions) та ЕФЕКТАМИ (Effects):
#       ДІЯ 1: "Gather Near Cover"
#         Preconditions: gathered == false
#         Effects: gathered = true
#         Cost: 1.0
#       ДІЯ 2: "Swarm Surround"
#         Preconditions: gathered == true, surrounding == false
#         Effects: surrounding = true
#         Cost: 1.0
#       ДІЯ 3: "Swarm Attack"
#         Preconditions: surrounding == true, swarm_ready == true
#         Effects: attacking = true
#         Cost: 1.0
#    г) A* ПЛАНЕР будує ланцюжок дій (PLAN):
#       Current State → [Gather] → [Surround] → [Attack] → Goal
#       Total Cost: 3.0 (мінімальна вартість)
#    д) NPC виконує ДІЇ з плану ПОСЛІДОВНО, кожна дія змінює стан світу
# 2. A* ALGORITHM (A-зірка)
#    - Алгоритм пошуку НАЙКОРОТШОГО шляху в графі станів
#    - f(n) = g(n) + h(n)
#      • g(n) = вартість від початку до поточного стану
#      • h(n) = евристична оцінка від поточного до цілі (heuristic)
#    - Обирає дії з МІНІМАЛЬНОЮ сумарною вартістю
# 3. BLACKBOARD PATTERN - Загальна пам'ять для ЗГРАЇ
#    - Всі Type III зомбі читають/пишуть до СПІЛЬНОЇ ПАМ'ЯТІ
#    - Зберігає: позиції всіх зомбі, позицію гравця, готовність до атаки
#    - Дозволяє КООРДИНАЦІЮ без прямого зв'язку між NPC
#    ПРИКЛАД КООРДИНАЦІЇ:
#    - Зомбі #1 публікує: "Я на позиції окружения"
#    - Зомбі #2 читає з Blackboard: "Скільки зомбі на позиціях?"
#    - Blackboard відповідає: "2 зомбі готові"
#    - Зомбі #2: swarm_ready = true → GOAP обирає "Swarm Attack"
# 4. CIRCLE FORMATION (Кругова формація)
#    - ГЕОМЕТРІЯ: розподіл N зомбі РІВНОМІРНО по колу
#    - Кут для кожного: θ = (360° / N) * index
#    - Позиція на колі: P = Center + (cos(θ), 0, sin(θ)) * radius
#    - РЕЗУЛЬТАТ: зомбі оточують гравця з УСІХ сторін одночасно
# 5. COVER USAGE (Використання укриттів)
#    - Пошук найближчої перешкоди до гравця
#    - Обчислення позиції ЗА перешкодою (від гравця)
#    - Вектор: obstacle_pos + (obstacle_pos - player_pos).normalized() * 3м
#    - ТАКТИКА: зомбі спочатку ХОВАЮТЬСЯ, потім ОТОЧУЮТЬ, потім АТАКУЮТЬ
# ПАТЕРН ПОВЕДІНКИ TYPE III - "КОЛЕКТИВНИЙ РОЗУМ":
# - Збираються біля укриттів (якщо є)
# - Оточують гравця по КОЛУ (circle formation)
# - Атакують ОДНОЧАСНО >= 2 зомбі (swarm attack)
# - Після 10 сек атаки - перегруповування (reset cycle)
# ============================================
# Посилання
var player: CharacterBody3D
var goap_agent: GOAP.GOAPAgent
var blackboard: Blackboard
var zombie_id: int
# Параметри
@export var max_hp: float = 250.0  # УВЕЛИЧЕНО для динамики
var hp: float = 250.0
@export var speed: float = 2.5
@export var damage: float = 12.0
@export var attack_range: float = 1.5
# Навігація
var nav_agent: NavigationAgent3D
# GOAP + Swarm
var current_goap_action: GOAP.GOAPAction = null
var replan_timer: float = 0.0
```

```
var replan_interval: float = 1.5  # Рідше перепланування - дає час для координації
var assigned_surround_position: Vector3 = Vector3.ZERO
var is_at_position: bool = false  # Чи на призначеній позиції
var is_ready_to_attack: bool = false  # Чи готовий до атаки
# GOAP: різні стратегії як окремі actions
var preferred_strategy: String = "balanced"  # Буде обиратися через GOAP
# Логування (щоб не спамити кожен кадр)
var last_log_time: float = 0.0
var log_interval: float = 3.0  # Логувати раз в 3 секунди
# Таймер для циклічності GOAP
var attack_duration: float = 0.0
var max_attack_duration: float = 10.0  # Після 10 сек атаки - нове окружения
# Метрики
var spawn_time: float
var distance_traveled: float = 0.0
var damage_dealt_to_player: float = 0.0
# HP Bar
var hp_label: Label3D
# Animation
var animation_player: AnimationPlayer
func _ready():
	add_to_group("zombies")  # Додаємо до групи зомбі
	GameLogger.write_log("[TYPE III] Spawned at " + str(global_position))
	player = get_tree().get_first_node_in_group("player")
	spawn_time = Time.get_ticks_msec() / 1000.0
	if player:
		GameLogger.write_log("[TYPE III] Player found at " +
str(player.global_position))
	else:
		GameLogger.write_log("[TYPE III] ERROR: Player NOT found!")
	# Унікальний ID для координації
	zombie_id = get_instance_id()
	# Підключення до Blackboard
	blackboard = Blackboard.get_instance()
	blackboard.register_type3_zombie(zombie_id, global_position)
	GameLogger.write_log("[TYPE III] Registered in Blackboard with ID: " +
str(zombie_id))
	# NavigationAgent для координованого руху
	nav_agent = NavigationAgent3D.new()
	nav_agent.path_desired_distance = 0.5
	nav_agent.target_desired_distance = 1.0
	nav_agent.radius = 0.5
	nav_agent.height = 1.8
	nav_agent.max_speed = speed
	nav_agent.avoidance_enabled = true
	add_child(nav_agent)
	# ВАЖЛИВО: Чекаємо синхронізацію навігації в Godot 4
	await get_tree().physics_frame
	GameLogger.write_log("[TYPE III] NavigationAgent ready")
	# Ініціалізація GOAP агента
	goap_agent = GOAP.GOAPAgent.new()
	initialize_goap_actions()
	# Встановлюємо початковий стан світу
	update_world_state()
	# Встановлюємо мету - АТАКА ТОЛПОЮ
	var goal = {"attacking": true}
	goap_agent.set_goal(goal)
	# Знайти AnimationPlayer в дочірніх нодах (якщо є модель)
	animation_player = find_animation_player(self)
	# Колір Type III - фіолетовий
	apply_color_to_model(Color(0.8, 0.0, 0.8))
	create_hp_bar()
func create_hp_bar():
	hp_label = Label3D.new()
	hp_label.text = "HP: %d" % hp
	hp_label.modulate = Color(0.5, 0, 0.5)  # Фіолетовий
	hp_label.font_size = 32
	hp_label.billboard = BaseMaterial3D.BILLBOARD_ENABLED
```

```
	hp_label.position = Vector3(0, 1.7, 0)  # Над капсулою
	hp_label.outline_size = 2
	add_child(hp_label)
func _physics_process(delta):
	replan_timer += delta
	# Гравітація
	if not is_on_floor():
		velocity.y -= 20.0 * delta
	else:
		velocity.y = 0.0
	# Оновлюємо Blackboard про нашу позицію та готовність
	blackboard.update_type3_zombie(zombie_id, global_position, is_ready_to_attack)
	# Оновлюємо Blackboard інформацією про гравця
	if player and is_instance_valid(player):
		var player_facing = -player.global_transform.basis.z
		blackboard.update_player_info(player.global_position, player_facing)
	# Оновлюємо стан світу
	update_world_state()
	# Перевіряємо чи потрібне перепланування
	if replan_timer >= replan_interval or goap_agent.needs_replan():
		goap_agent.replan()
		replan_timer = 0.0
	# Отримуємо поточну дію з плану
	current_goap_action = goap_agent.update()
	# ВАЖЛИВЕ ЛОГУВАННЯ: що повертає GOAP?
	if current_goap_action:
		GameLogger.write_log("[TYPE III] >>> EXECUTING ACTION: " +
current_goap_action.name)
		execute_goap_action(current_goap_action, delta)
	else:
		# КРИТИЧНО: якщо немає дії - перепланувати ЗАРАЗ!
		GameLogger.write_log("[TYPE III] !!! NO ACTION FROM GOAP! Replanning...")
		goap_agent.replan()
		current_goap_action = goap_agent.update()
		if current_goap_action:
			GameLogger.write_log("[TYPE III] >>> AFTER REPLAN: " +
current_goap_action.name)
			execute_goap_action(current_goap_action, delta)
	move_and_slide()
	distance_traveled += velocity.length() * delta
	# Оновлюємо HP бар
	update_hp_bar()
func update_hp_bar():
	if not hp_label:
		return
	hp_label.text = "HP: %d" % hp
	var hp_percent = hp / max_hp
	if hp_percent > 0.6:
		hp_label.modulate = Color(0.5, 0, 0.5)
	elif hp_percent > 0.3:
		hp_label.modulate = Color.YELLOW
	else:
		hp_label.modulate = Color.RED
# ========================================
# GOAP - ІНІЦІАЛІЗАЦІЯ ТА ОНОВЛЕННЯ
# ========================================
func initialize_goap_actions():
	"""
	ІНІЦІАЛІЗАЦІЯ GOAP - визначення НАБОРУ ДІЙ для планера
	МЕТОД ШТУЧНОГО ІНТЕЛЕКТУ: GOAP Action Set
	Кожна дія має:
	  1) NAME - назва дії
	  2) COST - вартість виконання (для A* алгоритму)
	  3) PRECONDITIONS - передумови (що має бути TRUE перед виконанням)
	  4) EFFECTS - ефекти (що стане TRUE після виконання)
	ГРАФ ДІЙ для A* планера:
	  START
	    ↓
```

```
        [gathered: false, surrounding: false, swarm_ready: ?, attacking: false]
          ↓
        ДІЯ 1: "Gather Near Cover" (cost=1.0)
          Preconditions: gathered == false
          Effects: gathered = true
          ↓
        [gathered: TRUE, surrounding: false, swarm_ready: ?, attacking: false]
          ↓
        ДІЯ 2: "Swarm Surround" (cost=1.0)
          Preconditions: gathered == true, surrounding == false
          Effects: surrounding = true
          ↓
        [gathered: true, surrounding: TRUE, swarm_ready: ?, attacking: false]
          ↓
        ДІЯ 3: "Swarm Attack" (cost=1.0)
          Preconditions: surrounding == true, swarm_ready == true
          Effects: attacking = true
          ↓
        GOAL: [attacking: TRUE] ← ДОСЯГНУТО!
      A* обирає план: [Gather → Surround → Attack] (total cost: 3.0)
      """
      # ========== ДІЯ 1: Збір біля укриття ==========
      var gather_action = GOAP.GOAPAction.new("Gather Near Cover", 1.0)
      gather_action.add_precondition("gathered", false)  # Можна виконати якщо НЕ
зібрані
      gather_action.add_effect("gathered", true)          # Після виконання - зібрані
      goap_agent.add_action(gather_action)
      # ========== ДІЯ 2: Оточення гравця ==========
      var surround_action = GOAP.GOAPAction.new("Swarm Surround", 1.0)
      surround_action.add_precondition("gathered", true)      # Потрібно спочатку
зібратись
      surround_action.add_precondition("surrounding", false)  # Ще не оточуємо
      surround_action.add_effect("surrounding", true)         # Після виконання -
оточили
      goap_agent.add_action(surround_action)
      # ========== ДІЯ 3: Атака толпою ==========
      var swarm_attack = GOAP.GOAPAction.new("Swarm Attack", 1.0)
      swarm_attack.add_precondition("surrounding", true)   # Потрібно спочатку оточити
      swarm_attack.add_precondition("swarm_ready", true)   # І толпа має бути готова
(>= 2)
      swarm_attack.add_effect("attacking", true)           # Після виконання -
атакуємо
      goap_agent.add_action(swarm_attack)
func update_world_state():
      """СПРОЩЕНИЙ стан світу для ТОЛПИ"""
      if not player or not is_instance_valid(player):
            return
      # Скільки Type III зомбі є всього?
      var swarm_count = blackboard.get_type3_count()
      var swarm_ready = swarm_count >= 2  # Толпа готова якщо >= 2
      # Стани для простого циклу
      goap_agent.set_world_state("gathered", is_at_position)  # Зібралися?
      goap_agent.set_world_state("surrounding", assigned_surround_position !=
Vector3.ZERO)  # Є позиція окружения?
      goap_agent.set_world_state("swarm_ready", swarm_ready)  # Толпа готова?
      goap_agent.set_world_state("attacking", false)  # Не атакуємо поки (скидається)
      # Логуємо рідко
      var current_time = Time.get_ticks_msec() / 1000.0
      if current_time - last_log_time >= log_interval:
            GameLogger.write_log("[TYPE III] Swarm: " + str(swarm_count) + " zombies.
Gathered: " + str(is_at_position) + " Ready: " + str(swarm_ready))
            last_log_time = current_time
      # Оновлюємо Blackboard
      blackboard.update_type3_zombie(zombie_id, global_position, is_at_position)
func execute_goap_action(action: GOAP.GOAPAction, delta: float):
      """Виконання СПРОЩЕНИХ дій ТОЛПИ"""
      match action.name:
            "Gather Near Cover":
```

```
                execute_gather_near_cover(delta)
            "Swarm Surround":
                execute_swarm_surround(delta)
            "Swarm Attack":
                execute_swarm_attack(delta)
# ========================================
# НОВЫЕ СПРОЩЕНІ ДІЇ - ТОЛПА з УКРИТТЯМИ
# ========================================
func execute_gather_near_cover(delta: float):
      """
      ВИКОРИСТАННЯ УКРИТТІВ - збір толпи біля перешкод
      МЕТОД ШТУЧНОГО ІНТЕЛЕКТУ: TACTICAL POSITIONING (тактичне позиціонування)
      СТРАТЕГІЯ:
        1) Знайти найближчу перешкоду (obstacle) ДО ГРАВЦЯ (не до зомбі!)
        2) Зібратись ЗА перешкодою (з ПРОТИЛЕЖНОГО боку від гравця)
        3) Якщо перешкод немає - зібратись НАВКОЛО гравця
      ГЕОМЕТРІЯ УКРИТТЯ:
        - Вектор від гравця ДО перешкоди: V = (obstacle_pos - player_pos)
        - Нормалізуємо: V_norm = V / |V|
        - Позиція ЗА перешкодою: gather_pos = obstacle_pos + V_norm * 3м
      ВІЗУАЛІЗАЦІЯ:
             [Player]
                ↓
                ↓ (вектор погляду)
                ↓
            [Obstacle] ← перешкода
                ↓
                ↓ (вектор to_obstacle)
                ↓
           [GATHER] ← зомбі збираються ТУТ (за перешкодою)
      BLACKBOARD КООРДИНАЦІЯ:
        - Всі Type III зомбі бачать одну й ту саму перешкоду
        - Кожен обирає СВОЮ позицію біля перешкоди (через індекс)
        - Створює ефект "зграї яка ховається перед атакою"
      """
      if not player:
            goap_agent.action_completed()
            return
      # ========== ЕТАП 1: Пошук найближчої перешкоди ДО ГРАВЦЯ ==========
      var obstacles = get_tree().get_nodes_in_group("obstacles")
      var nearest_obstacle: Node3D = null
      var min_dist = INF
      for obstacle in obstacles:
            if obstacle is StaticBody3D:
                  # Відстань від ГРАВЦЯ до перешкоди (не від зомбі!)
                  var dist =
player.global_position.distance_to(obstacle.global_position)
                  if dist < min_dist and dist < 15.0:  # В межах 15м від гравця
                        nearest_obstacle = obstacle
                        min_dist = dist
      # ========== ЕТАП 2: Обчислюємо точку збору ==========
      var gather_point: Vector3
      if nearest_obstacle:
            # ========== ВАРІАНТ А: Є перешкода - ховаємось ЗА НЕЮ ==========
            # Вектор від гравця ДО перешкоди (нормалізований)
            var to_obstacle = (nearest_obstacle.global_position -
player.global_position).normalized()
            # Позиція ЗА перешкодою = перешкода + напрямок * 3м
            gather_point = nearest_obstacle.global_position + to_obstacle * 3.0
            GameLogger.write_log("[TYPE III] GATHERING behind obstacle at: " +
str(gather_point))
      else:
            # ========== ВАРІАНТ Б: Немає перешкод - збираємось НАВКОЛО ==========
            # Використовуємо CIRCLE FORMATION (як в surround, але радіус 8м)
            var swarm_count = blackboard.get_type3_count()
            var angle_step = 360.0 / max(swarm_count, 1)
            var sorted_ids = blackboard.type3_zombies.keys()
            sorted_ids.sort()
```

```
            var my_index = sorted_ids.find(zombie_id)
            var my_angle = deg_to_rad(angle_step * my_index)
            # Позиція по колу (радіус 8м)
            gather_point = player.global_position + Vector3(cos(my_angle), 0,
sin(my_angle)) * 8.0
            GameLogger.write_log("[TYPE III] GATHERING around player at: " +
str(gather_point))
      # ========== ЕТАП 3: Рух до точки збору ==========
      var dist = global_position.distance_to(gather_point)
      if dist < 3.0:
            # Досягли точки збору - СТОП!
            is_at_position = true
            assigned_surround_position = gather_point
            var saved_y = velocity.y
            velocity = Vector3.ZERO
            velocity.y = saved_y
            play_animation("IDLE")
            goap_agent.action_completed()  # Повідомляємо GOAP що дія завершена
            return
      # Ще не досягли - рухаємось
      var direction = global_position.direction_to(gather_point)
      velocity.x = direction.x * speed
      velocity.z = direction.z * speed
      play_animation("WALK")
func execute_swarm_surround(delta: float):
      """
      КРУГОВА ФОРМАЦІЯ - оточення гравця ТОЛПОЮ
      МЕТОД ШТУЧНОГО ІНТЕЛЕКТУ: CIRCLE FORMATION (геометричний розподіл)
      МАТЕМАТИКА:
        N зомбі мають оточити гравця РІВНОМІРНО по колу (радіус 5м)
        1) Обчислюємо КУТ для кожного зомбі:
           angle_step = 360° / N
           my_angle = angle_step * my_index
           Приклад для N=4:
             Зомбі #0: 0°   (попереду гравця)
             Зомбі #1: 90°  (справа)
             Зомбі #2: 180° (ззаду)
             Зомбі #3: 270° (зліва)
        2) Обчислюємо ПОЗИЦІЮ на колі (ПОЛЯРНІ → ДЕКАРТОВІ координати):
           x = center_x + radius * cos(angle)
           z = center_z + radius * sin(angle)
           В 3D просторі (Y=0 - площина):
           Position = Player_Pos + Vector3(cos(θ) * R, 0, sin(θ) * R)
        3) Кожен зомбі рухається до СВОЄЇ позиції на колі
      РЕЗУЛЬТАТ:
        - Зомбі СИНХРОННО оточують гравця з УСІХ сторін
        - Неможливо втекти - зомбі блокують всі напрямки
        - BLACKBOARD координує: всі знають скільки зомбі і свій індекс
      """
      if not player:
            goap_agent.action_completed()
            return
      # ========== ЕТАП 1: Визначаємо кількість зомбі та індекс ==========
      var swarm_count = blackboard.get_type3_count()  # Скільки Type III зомбі?
      var angle_step = 360.0 / max(swarm_count, 1)    # Крок кута для рівномірного
розподілу
      # Знаходимо свій ІНДЕКС серед зомбі (через Blackboard)
      var sorted_ids = blackboard.type3_zombies.keys()
      sorted_ids.sort()  # Сортуємо щоб порядок був однаковий для всіх
      var my_index = sorted_ids.find(zombie_id)
      if my_index == -1:
            my_index = 0
      # ========== ЕТАП 2: Обчислюємо КУТ для цього зомбі ==========
      var my_angle = deg_to_rad(angle_step * my_index)
      # Приклад: index=0 → 0°, index=1 → 90°, index=2 → 180°, index=3 → 270°
      # ========== ЕТАП 3: Обчислюємо ПОЗИЦІЮ на колі (ПОЛЯРНІ → ДЕКАРТОВІ) ==========
      # Формула: P = Center + (cos(θ), 0, sin(θ)) * radius
      assigned_surround_position = player.global_position + Vector3(
```

```
                cos(my_angle) * 5.0,  # X координата на колі
                0,                    # Y = 0 (площина)
                sin(my_angle) * 5.0   # Z координата на колі
        )
        GameLogger.write_log("[TYPE III] SURROUNDING player at angle: " +
str(rad_to_deg(my_angle)) + " deg")
        # ========== ЕТАП 4: Рухаємось до позиції ==========
        var dist = global_position.distance_to(assigned_surround_position)
        if dist < 2.0:
                # Досягли позиції окружения - СТОП!
                is_at_position = true
                var saved_y = velocity.y
                velocity = Vector3.ZERO
                velocity.y = saved_y
                play_animation("IDLE")
                goap_agent.action_completed()  # Повідомляємо GOAP що дія завершена
                return
        # Ще не досягли - рухаємось до позиції
        var direction = global_position.direction_to(assigned_surround_position)
        velocity.x = direction.x * speed * 1.2  # Швидше при окруженні
        velocity.z = direction.z * speed * 1.2
        play_animation("WALK")
func execute_swarm_attack(delta: float):
        """АТАКА ТОЛПОЮ - всі зомбі одночасно наступають на гравця"""
        if not player:
                reset_goap_cycle()
                return
        var distance = global_position.distance_to(player.global_position)
        if distance < attack_range:
                # АТАКУЄМО!
                var saved_y = velocity.y
                velocity = Vector3.ZERO
                velocity.y = saved_y
                play_animation("ATK")
                var dmg = damage * delta * 1.5  # Більший урон при атаці толпою
                if player and is_instance_valid(player) and
player.has_method("take_damage"):
                        player.take_damage(dmg)
                        damage_dealt_to_player += dmg
                        GameLogger.write_log("[TYPE III] SWARM ATTACK! Damage: " +
str(dmg))
                # Відслідковуємо тривалість атаки
                attack_duration += delta
                # ЦИКЛІЧНІСТЬ: Після 10 сек - нове окружения!
                if attack_duration >= max_attack_duration:
                        GameLogger.write_log("[TYPE III] Attack duration exceeded!
Resetting...")
                        attack_duration = 0.0
                        reset_goap_cycle()
                        return
        else:
                # Гравець далеко - наступаємо ТОЛПОЮ (зберігаючи напрямок окружения)
                attack_duration = 0.0
                # Рухаємось від своєї позиції окружения до гравця
                var target = player.global_position
                if assigned_surround_position != Vector3.ZERO:
                        # Зберігаємо напрямок (з якого боку я окружав)
                        var angle_offset = assigned_surround_position -
player.global_position
                        target = player.global_position + angle_offset.normalized() * 2.0
                var direction = global_position.direction_to(target)
                velocity.x = direction.x * speed * 1.4  # Агресивний наступ
                velocity.z = direction.z * speed * 1.4
                play_animation("WALK")
func take_damage(amount: float):
        hp -= amount
        play_animation("HIT")
        print("Зомбі Type III отримав урон: ", amount, " HP: ", hp)
```

```
	if hp <= 0:
		die()
func reset_goap_cycle():
	"""СКИДАННЯ ЦИКЛУ - починаємо збиратися заново"""
	GameLogger.write_log("[TYPE III] RESETTING CYCLE - gathering again!")
	# Скидаємо всі стани
	goap_agent.set_world_state("gathered", false)
	goap_agent.set_world_state("surrounding", false)
	goap_agent.set_world_state("attacking", false)
	# Скидаємо внутрішні змінні
	assigned_surround_position = Vector3.ZERO
	is_at_position = false
	is_ready_to_attack = false
	attack_duration = 0.0
	# Завершуємо поточну дію
	goap_agent.action_completed()
	# ФОРСУЄМО перепланування
	goap_agent.replan()
func die():
	play_animation("DEAD")
	# Повідомляємо Blackboard про смерть
	blackboard.unregister_type3_zombie(zombie_id)
	blackboard.record_death_location(global_position)
	await get_tree().create_timer(2.0).timeout  # Чекаємо поки анімація смерті
завершиться
	var game_manager = get_tree().get_first_node_in_group("game_manager")
	if game_manager:
		game_manager.on_zombie_death(self)
	queue_free()
# ============================================
# АНІМАЦІЇ
# ============================================
func find_animation_player(node: Node) -> AnimationPlayer:
	"""Рекурсивно шукає AnimationPlayer в дереві"""
	if node is AnimationPlayer:
		return node
	for child in node.get_children():
		var result = find_animation_player(child)
		if result:
			return result
	return null
func play_animation(anim_name: String):
	"""Програє анімацію якщо є AnimationPlayer"""
	if animation_player and animation_player.has_animation(anim_name):
		animation_player.play(anim_name)
func apply_color_to_model(color: Color):
	"""Застосовує колір до всіх mesh в моделі"""
	for child in get_children():
		apply_color_recursive(child, color)
func apply_color_recursive(node: Node, color: Color):
	"""Рекурсивно шукає MeshInstance3D і застосовує колір"""
	if node is MeshInstance3D:
		# Створюємо новий матеріал з модуляцією
		var material = StandardMaterial3D.new()
		material.albedo_color = color
		node.set_surface_override_material(0, material)
	for child in node.get_children():
		apply_color_recursive(child, color)
```

base.gd

```
extends StaticBody3D
@export var max_hp: float = 1000.0
var hp: float = 1000.0
func _ready():
	add_to_group("base")
	print("Штаб готовий! HP: ", hp)
func take_damage(amount: float):
```

```
	hp -= amount
	print("Штаб отримав урон: ", amount, " HP: ", hp)
	if hp <= 0:
		hp = 0
		die()
func die():
	print("Штаб знищено!")
```

bullet.gd

```
extends RigidBody3D
var speed: float = 20.0
var damage: float = 25.0
var lifetime: float = 5.0
var time_alive: float = 0.0
var direction: Vector3 = Vector3.FORWARD
var shooter: Node = null  # Хто випустив кулю
# Липка пуля
var magnet_range: float = 3.0  # Радіус притягування
var magnet_strength: float = 50.0  # Сила притягування
var target_zombie: Node = null
var knockback_force: float = 5.0  # Сила відштовхування
func _ready():
	# Підключення до сигналу зіткнення
	body_entered.connect(_on_body_entered)
	# Встановлюємо швидкість
	linear_velocity = direction * speed
func _physics_process(delta):
	time_alive += delta
	if time_alive >= lifetime:
		queue_free()
	# Шукаємо найближчого зомбі для притягування
	find_nearest_zombie()
	if target_zombie and is_instance_valid(target_zombie):
		# Притягуємось до зомбі
		var to_zombie = target_zombie.global_position - global_position
		var distance = to_zombie.length()
		if distance < magnet_range and distance > 0.5:
			var magnet_force = to_zombie.normalized() * magnet_strength
			apply_central_force(magnet_force)
func find_nearest_zombie():
	var zombies = get_tree().get_nodes_in_group("zombies")
	var nearest_distance = magnet_range
	target_zombie = null
	for zombie in zombies:
		if zombie and is_instance_valid(zombie):
			var dist = global_position.distance_to(zombie.global_position)
			if dist < nearest_distance:
				nearest_distance = dist
				target_zombie = zombie
func _on_body_entered(body):
	print("Куля зіткнулась з: ", body.name, " Тип: ", body.get_class())
	# НЕ атакуємо того, хто випустив кулю!
	if body == shooter:
		print("  -> Це стрілець, ігноруємо")
		return
	# Перевірка чи це зомбі
	var has_damage_method = body.has_method("take_damage")
	var is_zombie = body.is_in_group("zombies")
	print("  -> has_method('take_damage'): ", has_damage_method)
	print("  -> is_in_group('zombies'): ", is_zombie)
	if has_damage_method and is_zombie:
		print("  -> ВЛУЧИВ! Наносимо урон ", damage)
		# Наносимо урон
		body.take_damage(damage)
		# Відштовхуємо зомбі назад
		if body is CharacterBody3D:
			var knockback_direction = (body.global_position -
```

```
global_position).normalized()
                  knockback_direction.y = 0  # Без вертикального відштовхування
                  body.velocity += knockback_direction * knockback_force
                  print("  -> Відштовхуємо зомбі!")
            queue_free()
            return
      # Якщо влучили в щось інше - зникаємо
      print("  -> Влучив в щось інше, зникаємо")
      queue_free()
```

obstacle.gd

```
extends StaticBody3D
func _ready():
      add_to_group("obstacles")
```

player.gd

```
extends CharacterBody3D
@export var max_hp: float = 100.0
var hp: float = 100.0
@export var speed: float = 5.0
@export var mouse_sensitivity: float = 0.002
@export var bullet_scene: PackedScene
@export var bullet_speed: float = 30.0  # Збільшено швидкість
@export var bullet_damage: float = 75.0  # Збільшено урон з 25 до 75
@export var fire_rate: float = 0.2  # УВЕЛИЧЕНА скорость стрельбы (5 выстрелов/сек)
var shoot_point: Marker3D
var camera: Camera3D
var next_fire_time: float = 0.0
var is_shooting: bool = false
# Camera - просто привязана к игроку
# Камера - дочерний узел, настраивается в редакторе
# HP Bar
var hp_bar: ProgressBar
var hp_label: Label3D
func _ready():
      add_to_group("player")
      shoot_point = $ShootPoint
      camera = $Camera3D
      # Камера уже привязана как дочерний узел - ничего не делаем
      # Створюємо HP бар над гравцем
      create_hp_bar()
      # Захоплюємо мишу
      Input.mouse_mode = Input.MOUSE_MODE_CAPTURED
func create_hp_bar():
      # Створюємо Label3D для відображення HP
      hp_label = Label3D.new()
      hp_label.text = "HP: %d/%d" % [hp, max_hp]
      hp_label.modulate = Color.GREEN
      hp_label.font_size = 32
      hp_label.billboard = BaseMaterial3D.BILLBOARD_ENABLED
      hp_label.position = Vector3(0, 2, 0)
      add_child(hp_label)
func _input(event):
      # Поворот гравця мишею
      if event is InputEventMouseMotion:
            rotate_y(-event.relative.x * mouse_sensitivity)
func _physics_process(delta):
      # Гравітація
      if not is_on_floor():
            velocity.y -= 20.0 * delta
      else:
            velocity.y = 0.0
      # Рух
      var input_dir = Input.get_vector("ui_left", "ui_right", "ui_up", "ui_down")
      var direction = (transform.basis * Vector3(input_dir.x, 0,
input_dir.y)).normalized()
      velocity.x = direction.x * speed
```

```
	velocity.z = direction.z * speed
	move_and_slide()
	# Стрільба
	if Input.is_action_pressed("shoot") and Time.get_ticks_msec() / 1000.0 >=
next_fire_time:
		shoot()
		next_fire_time = Time.get_ticks_msec() / 1000.0 + fire_rate
	# Оновлюємо HP бар
	if hp_label:
		hp_label.text = "HP: %d/%d" % [hp, max_hp]
		var hp_percent = hp / max_hp
		if hp_percent > 0.6:
			hp_label.modulate = Color.GREEN
		elif hp_percent > 0.3:
			hp_label.modulate = Color.YELLOW
		else:
			hp_label.modulate = Color.RED
func shoot():
	if bullet_scene == null:
		print("ПОМИЛКА: Bullet scene не призначено!")
		return
	if shoot_point == null:
		print("ПОМИЛКА: ShootPoint не знайдено!")
		return
	is_shooting = true
	var bullet = bullet_scene.instantiate()
	bullet.global_position = shoot_point.global_position
	# Напрямок стрільби - вперед від гравця
	bullet.direction = -global_transform.basis.z
	bullet.damage = bullet_damage
	bullet.speed = bullet_speed
	bullet.shooter = self  # Зберігаємо стрілка
	get_parent().add_child(bullet)
	print("Постріл! Напрямок: ", bullet.direction, " Позиція: ",
bullet.global_position)
	await get_tree().create_timer(0.1).timeout
	is_shooting = false
func take_damage(amount: float):
	hp -= amount
	print("Гравець отримав урон: ", amount, " HP: ", hp)
	if hp <= 0:
		hp = 0
		die()
func die():
	print("Гравець загинув!")
	queue_free()
```

hud.gd

```
extends CanvasLayer
@onready var player_health_bar = $Panel/VBoxContainer/PlayerHealthBar
@onready var player_health_label = $Panel/VBoxContainer/PlayerHealthLabel
@onready var wave_label = $Panel/VBoxContainer/WaveLabel
@onready var zombies_left_label = $Panel/VBoxContainer/ZombiesLeftLabel
# Створюємо програмно
var timer_label: Label
var game_over_label: Label
var game_over_bg: ColorRect  # Фон для Game Over
var player: CharacterBody3D
var game_manager: Node3D
var initialized: bool = false
var game_time: float = 0.0  # Секундомер
var game_over: bool = false
func _ready():
	# Відкладаємо ініціалізацію на 1 фрейм
	call_deferred("initialize")
func initialize():
	player = get_tree().get_first_node_in_group("player")
```

```
	game_manager = get_tree().get_first_node_in_group("game_manager")
	print("HUD готовий!")
	print("  Player: ", player)
	print("  GameManager: ", game_manager)
	if game_manager:
		print("  GM current_wave: ", game_manager.current_wave)
		print("  GM zombies_alive: ", game_manager.zombies_alive)
	# Створюємо секундомер
	create_timer_label()
	# Створюємо Game Over екран
	create_game_over_label()
	initialized = true
func create_timer_label():
	"""Створити секундомер СПРАВА ВВЕРХУ"""
	timer_label = Label.new()
	timer_label.text = "Час: 00:00"
	timer_label.add_theme_font_size_override("font_size", 28)
	# Прикріплюємо до ПРАВОГО верхнього кута
	timer_label.anchor_left = 1.0
	timer_label.anchor_top = 0.0
	timer_label.anchor_right = 1.0
	timer_label.anchor_bottom = 0.0
	# Зміщення від правого краю
	timer_label.offset_left = -180  # 180 пікселів від правого краю
	timer_label.offset_top = 10
	timer_label.offset_right = -10
	timer_label.offset_bottom = 50
	timer_label.horizontal_alignment = HORIZONTAL_ALIGNMENT_RIGHT
	add_child(timer_label)
func create_game_over_label():
	"""Створити ПОВНОЕКРАННИЙ Game Over з фоном"""
	# Створюємо темний напівпрозорий фон
	game_over_bg = ColorRect.new()
	game_over_bg.color = Color(0, 0, 0, 0.8)  # Чорний 80% прозорості
	# ПОВНОЕКРАННИЙ фон
	game_over_bg.anchor_left = 0.0
	game_over_bg.anchor_top = 0.0
	game_over_bg.anchor_right = 1.0
	game_over_bg.anchor_bottom = 1.0
	game_over_bg.visible = false
	game_over_bg.z_index = 99
	add_child(game_over_bg)
	# Текст Game Over
	game_over_label = Label.new()
	game_over_label.text = "GAME OVER"
	game_over_label.add_theme_font_size_override("font_size", 80)
	game_over_label.add_theme_color_override("font_color", Color.RED)
	game_over_label.horizontal_alignment = HORIZONTAL_ALIGNMENT_CENTER
	game_over_label.vertical_alignment = VERTICAL_ALIGNMENT_CENTER
	# ПОВНОЕКРАННИЙ - заповнює весь екран
	game_over_label.anchor_left = 0.0
	game_over_label.anchor_top = 0.0
	game_over_label.anchor_right = 1.0
	game_over_label.anchor_bottom = 1.0
	game_over_label.offset_left = 0
	game_over_label.offset_top = 0
	game_over_label.offset_right = 0
	game_over_label.offset_bottom = 0
	# Додаємо обводку тексту
	game_over_label.add_theme_color_override("font_outline_color", Color.BLACK)
	game_over_label.add_theme_constant_override("outline_size", 8)
	game_over_label.visible = false  # Спочатку приховано
	game_over_label.z_index = 100  # Поверх всього!
	add_child(game_over_label)
func _process(delta):
	if not initialized:
		return
	# Секундомер (тільки якщо гра не закінчена)
```

```
	if not game_over:
		game_time += delta
		var minutes = int(game_time) / 60
		var seconds = int(game_time) % 60
		if timer_label:
			timer_label.text = "Час: %02d:%02d" % [minutes, seconds]
	# Здоров'я гравця
	if player and is_instance_valid(player):
		player_health_bar.value = player.hp
		player_health_label.text = "Гравець: %d / %d" % [player.hp,
player.max_hp]
	else:
		# Гравець помер - GAME OVER!
		if not game_over:
			game_over = true
			show_game_over()
	# Інформація про хвилі
	if game_manager and is_instance_valid(game_manager):
		wave_label.text = "Хвиля: %d" % game_manager.current_wave
		zombies_left_label.text = "Зомбі: %d" % game_manager.zombies_alive
func show_game_over():
	"""Показати екран Game Over"""
	# Показуємо фон
	if game_over_bg:
		game_over_bg.visible = true
	# Показуємо текст
	if game_over_label:
		game_over_label.visible = true
		var minutes = int(game_time) / 60
		var seconds = int(game_time) % 60
		game_over_label.text = "GAME OVER\n\nВи продержались:
%02d:%02d\n\nНатисніть ESC для виходу" % [minutes, seconds]
		# Звільняємо мишу
		Input.mouse_mode = Input.MOUSE_MODE_VISIBLE
```

behavior_tree.gd

```
extends RefCounted
class_name BTNode
enum Status {
	SUCCESS,
	FAILURE,
	RUNNING
}
func tick() -> Status:
	return Status.FAILURE
# BTSelector - вибирає перший успішний дочірній вузол
class BTSelector extends BTNode:
	var children: Array[BTNode] = []
	var current_child: int = 0
	func _init(child_nodes: Array):
		for child in child_nodes:
			children.append(child)
	func tick() -> Status:
		for i in range(current_child, children.size()):
			var status = children[i].tick()
			if status == Status.RUNNING:
				current_child = i
				return Status.RUNNING
			elif status == Status.SUCCESS:
				current_child = 0
				return Status.SUCCESS
		current_child = 0
		return Status.FAILURE
# BTSequence - виконує всі дочірні вузли по черзі
class BTSequence extends BTNode:
	var children: Array[BTNode] = []
	var current_child: int = 0
```

```
	func _init(child_nodes: Array):
		for child in child_nodes:
			children.append(child)
	func tick() -> Status:
		for i in range(current_child, children.size()):
			var status = children[i].tick()
			if status == Status.RUNNING:
				current_child = i
				return Status.RUNNING
			elif status == Status.FAILURE:
				current_child = 0
				return Status.FAILURE
		current_child = 0
		return Status.SUCCESS
# BTCondition - перевіряє умову
class BTCondition extends BTNode:
	var condition_func: Callable
	func _init(func_ref: Callable):
		condition_func = func_ref
	func tick() -> Status:
		if condition_func.call():
			return Status.SUCCESS
		return Status.FAILURE
# BTAction - виконує дію
class BTAction extends BTNode:
	var action_func: Callable
	func _init(func_ref: Callable):
		action_func = func_ref
	func tick() -> Status:
		return action_func.call()
# BTInverter - інвертує результат дочірнього вузла
class BTInverter extends BTNode:
	var child: BTNode
	func _init(child_node: BTNode):
		child = child_node
	func tick() -> Status:
		var status = child.tick()
		if status == Status.SUCCESS:
			return Status.FAILURE
		elif status == Status.FAILURE:
			return Status.SUCCESS
		return Status.RUNNING
# BTRepeater - повторює дочірній вузол N разів
class BTRepeater extends BTNode:
	var child: BTNode
	var repetitions: int
	var current_rep: int = 0
	func _init(child_node: BTNode, reps: int = -1):
		child = child_node
		repetitions = reps
	func tick() -> Status:
		while repetitions == -1 or current_rep < repetitions:
			var status = child.tick()
			if status == Status.RUNNING:
				return Status.RUNNING
			elif status == Status.FAILURE:
				current_rep = 0
				return Status.FAILURE
			current_rep += 1
		current_rep = 0
		return Status.SUCCESS
```

config.gd

```
extends Node
class_name Config
# Константи гри
const OBSTACLE_COUNT = 18
```

```
const TYPE1_COUNT = 10
const TYPE2_COUNT = 7
const TYPE3_COUNT = 3
# Параметри арени
const ARENA_SIZE = 50.0
const SPAWN_MARGIN = 23.0
# Параметри гравця
const PLAYER_MAX_HP = 100.0
const PLAYER_SPEED = 5.0
const PLAYER_DAMAGE = 25.0
const PLAYER_FIRE_RATE = 1.0
const BULLET_SPEED = 20.0
# Параметри штабу
const BASE_MAX_HP = 1000.0
# Параметри зомбі
const ZOMBIE_TYPE1_HP = 100.0
const ZOMBIE_TYPE1_SPEED = 2.0
const ZOMBIE_TYPE1_DAMAGE = 10.0
const ZOMBIE_TYPE2_HP = 150.0
const ZOMBIE_TYPE2_SPEED = 3.0
const ZOMBIE_TYPE2_DAMAGE = 15.0
const ZOMBIE_TYPE3_HP = 200.0
const ZOMBIE_TYPE3_SPEED = 2.5
const ZOMBIE_TYPE3_DAMAGE = 12.0
const ZOMBIE_ATTACK_RANGE = 3.0
```

logger.gd

```
extends Node
class_name GameLogger
static var log_file: FileAccess = null
static var log_path: String = "user://game_log.txt"
static var is_initialized: bool = false
static func init():
	if is_initialized:
		return
	# Відкриваємо файл для запису (перезаписуємо старий)
	log_file = FileAccess.open(log_path, FileAccess.WRITE)
	if log_file:
		is_initialized = true
		write_log("=== GAME LOG STARTED ===")
		write_log("Time: " + Time.get_datetime_string_from_system())
		write_log("========================")
	else:
		print("ПОМИЛКА: Не вдалося відкрити файл логу!")
static func write_log(message: String):
	if not is_initialized:
		init()
	if log_file:
		var timestamp = "[%.3f] " % (Time.get_ticks_msec() / 1000.0)
		log_file.store_line(timestamp + message)
		log_file.flush()  # Записуємо відразу на диск
	# Також виводимо в консоль
	print(message)
static func close():
	if log_file:
		write_log("=== GAME LOG ENDED ===")
		log_file.close()
		log_file = null
		is_initialized = false
static func get_log_path() -> String:
	# Повертаємо реальний шлях до файлу
	return ProjectSettings.globalize_path(log_path)
```

metrics.gd

```
extends RefCounted
class_name MetricsCollector
class ZombieMetrics:
```

```
	var zombie_type: String
	var survival_time: float
	var distance_traveled: float
	var damage_to_player: float
	var cause_of_death: String
var metrics: Array[ZombieMetrics] = []
func record_zombie_death(zombie):
	var m = ZombieMetrics.new()
	m.zombie_type = zombie.get_class()
	m.survival_time = (Time.get_ticks_msec() / 1000.0) - zombie.spawn_time
	m.distance_traveled = zombie.distance_traveled
	m.damage_to_player = zombie.damage_dealt_to_player
	m.cause_of_death = "Player" if zombie.hp <= 0 else "Survived"
	metrics.append(m)
func save_to_file(path: String):
	var file = FileAccess.open(path, FileAccess.WRITE)
	if file:
		file.store_line("Type,SurvivalTime,Distance,DmgPlayer,Death")
		for m in metrics:
			file.store_line("%s,%.2f,%.2f,%.2f,%s" % [
				m.zombie_type,
				m.survival_time,
				m.distance_traveled,
				m.damage_to_player,
				m.cause_of_death
			])
		file.close()
		print("Метрики збережено: ", path)
```

vision_cone.gd

```
extends RefCounted
class_name VisionCone
var owner_node: Node3D
var cone_angle: float  # В градусах
var cone_range: float
var can_see: bool = false
func _init(owner: Node3D, angle: float, vision_range: float):
	owner_node = owner
	cone_angle = angle
	cone_range = vision_range
func update(delta: float, target: Node3D, obstacles: Array) -> bool:
	if not target or not is_instance_valid(target):
		can_see = false
		return false
	var dist = owner_node.global_position.distance_to(target.global_position)
	# Перевірка відстані
	if dist > cone_range:
		can_see = false
		return false
	# Перевірка кута
	var forward = -owner_node.global_transform.basis.z
	var to_target = owner_node.global_position.direction_to(target.global_position)
	var angle = rad_to_deg(acos(forward.dot(to_target)))
	if angle > cone_angle / 2.0:
		can_see = false
		return false
	# Raycast перевірка на перешкоди
	var space_state = owner_node.get_world_3d().direct_space_state
	var query = PhysicsRayQueryParameters3D.create(
		owner_node.global_position + Vector3(0, 0.5, 0),
		target.global_position + Vector3(0, 0.5, 0)
	)
	query.exclude = [owner_node]
	var result = space_state.intersect_ray(query)
	if result and result.collider != target:
		can_see = false
		return false
```

```
	can_see = true
	return true
```

zombie_memory.gd

```
extends RefCounted
class_name ZombieMemory
var memory_duration: float
var last_seen_time: float = -999.0
var last_known_position: Vector3 = Vector3.ZERO
var has_seen_player: bool = false
func _init(duration: float):
	memory_duration = duration
func update(current_time: float, can_see: bool, player_pos: Vector3):
	if can_see:
		last_seen_time = current_time
		last_known_position = player_pos
		has_seen_player = true
func has_memory(current_time: float) -> bool:
	if not has_seen_player:
		return false
	return (current_time - last_seen_time) < memory_duration
func get_last_known_position() -> Vector3:
	return last_known_position
func get_time_since_last_seen(current_time: float) -> float:
	return current_time - last_seen_time
```

game_manager.gd

```
extends Node3D
# Prefabs
@export var obstacle_scene: PackedScene
@export var zombie_type1_scene: PackedScene
@export var zombie_type2_scene: PackedScene
@export var zombie_type3_scene: PackedScene
# Config
const OBSTACLE_COUNT = 18
# References
var player: CharacterBody3D
var zombies_container: Node3D
var obstacles_container: Node3D
var zombies_alive: int = 0
var current_wave: int = 0
var total_waves: int = 10  # Збільшено з 7 до 10 хвиль
var game_over: bool = false
var victory: bool = false
# Wave management
var spawn_timer: float = 0.0
var spawn_interval: float = 15.0  # Спавн кожні 15 секунд
var zombies_per_spawn: int = 2  # Кількість зомбі за раз
var can_spawn: bool = true
# Spawn positions (3 точки равноудалені від штабу)
var spawn_positions: Array[Vector3] = []
# Metrics
var metrics_collector: MetricsCollector
func _ready():
	add_to_group("game_manager")
	# Ініціалізуємо логгер
	GameLogger.init()
	GameLogger.write_log("=== GAME MANAGER STARTED ===")
	player = get_node("Player")
	zombies_container = get_node("Zombies")
	obstacles_container = get_node("Obstacles")
	GameLogger.write_log("Player found: " + str(player != null))
	GameLogger.write_log("Zombies container: " + str(zombies_container != null))
	metrics_collector = MetricsCollector.new()
	# Створюємо 3 спавнери навколо ігрового поля
	setup_spawn_positions()
	# Bake navigation після spawn перешкод
```

```
	call_deferred("spawn_obstacles")
	call_deferred("bake_navigation")
	call_deferred("start_first_wave")
	GameLogger.write_log("Log file location: " + GameLogger.get_log_path())
func setup_spawn_positions():
	# 6 точок на краях карти по периметру (БОЛЬШЕ ДИНАМИКИ!)
	var radius = 35.0
	# 6 точок під кутами 0°, 60°, 120°, 180°, 240°, 300°
	for i in range(6):
		var angle = deg_to_rad(i * 60)
		var pos = Vector3(
			cos(angle) * radius,
			1,
			sin(angle) * radius
		)
		spawn_positions.append(pos)
func spawn_obstacles():
	for i in range(OBSTACLE_COUNT):
		var obstacle = obstacle_scene.instantiate()
		var pos = Vector3(
			randf_range(-20, 20),
			1.5,
			randf_range(-20, 20)
		)
		# Перевірка що не спавнимо на гравці
		if pos.distance_to(Vector3(0, 1.5, -15)) > 5:
			obstacle.global_position = pos
			obstacles_container.add_child(obstacle)
func bake_navigation():
	var nav_region = get_node("NavigationRegion3D")
	if nav_region:
		GameLogger.write_log("Baking navigation mesh...")
		nav_region.bake_navigation_mesh()
		GameLogger.write_log("Navigation mesh baked!")
	else:
		GameLogger.write_log("ERROR: NavigationRegion3D not found!")
func start_first_wave():
	current_wave = 1
	# Спавним першу групу зомбі відразу
	spawn_zombies()
	can_spawn = true
func spawn_zombies():
	GameLogger.write_log("=== SPAWNING ZOMBIES ===")
	# Спавнимо зомбі
	for i in range(zombies_per_spawn):
		var zombie_type = get_zombie_type_for_wave()
		var zombie = zombie_type.instantiate()
		var type_name = "UNKNOWN"
		if zombie_type == zombie_type1_scene:
			type_name = "TYPE I (Green)"
		elif zombie_type == zombie_type2_scene:
			type_name = "TYPE II (Orange)"
		elif zombie_type == zombie_type3_scene:
			type_name = "TYPE III (Purple)"
		# ВАЖЛИВО: спочатку додаємо в дерево, ПОТІМ встановлюємо позицію
		zombies_container.add_child(zombie)
		zombie.add_to_group("zombies")
		# Тепер можна безпечно встановлювати global_position
		var spawn_pos = spawn_positions[i % spawn_positions.size()]
		# Додаємо випадкове зміщення щоб не спавнилися на голову
		var random_offset = Vector3(randf_range(-3, 3), 0, randf_range(-3, 3))
		zombie.global_position = spawn_pos + random_offset
		zombies_alive += 1
		GameLogger.write_log("Spawned " + type_name + " at position " +
str(spawn_pos))
	GameLogger.write_log("Wave " + str(current_wave) + ": Spawned " +
str(zombies_per_spawn) + " zombies | Total alive: " + str(zombies_alive))
func get_zombie_type_for_wave() -> PackedScene:
```

```
	# Визначаємо тип зомбі залежно від хвилі
	var rand_val = randf()
	if current_wave <= 2:
		# Перші 2 хвилі - тільки Type I
		return zombie_type1_scene
	elif current_wave <= 4:
		# Хвилі 3-4 - Type I та II
		return zombie_type1_scene if rand_val < 0.6 else zombie_type2_scene
	else:
		# Хвилі 5+ - всі типи
		if rand_val < 0.4:
			return zombie_type1_scene
		elif rand_val < 0.7:
			return zombie_type2_scene
		else:
			return zombie_type3_scene
func _process(delta):
	if game_over:
		return
	# Перевірка умов поразки
	if player.hp <= 0:
		game_over = true
		print("ПОРАЗКА! Гравець загинув!")
		metrics_collector.save_to_file("res://metrics.csv")
		return
	# Постійний спавн зомбі кожні 15 секунд
	if can_spawn:
		spawn_timer += delta
		if spawn_timer >= spawn_interval:
			spawn_timer = 0.0
			current_wave += 1
			# Збільшуємо кількість зомбі з часом
			if current_wave % 5 == 0:
				zombies_per_spawn = min(zombies_per_spawn + 1, 5)  # Максимум
5 за раз
			spawn_zombies()
			print("Хвиля ", current_wave, " - спавнимо ", zombies_per_spawn, "
зомбі")
func on_zombie_death(zombie):
	zombies_alive -= 1
	GameLogger.write_log("Zombie died! Remaining: " + str(zombies_alive))
	metrics_collector.record_zombie_death(zombie)
func _exit_tree():
	GameLogger.write_log("=== GAME ENDED ===")
	GameLogger.close()
```

export_presets.cfg

```
[preset.0]
name="Windows Desktop"
platform="Windows Desktop"
runnable=true
advanced_options=false
dedicated_server=false
custom_features=""
export_filter="all_resources"
include_filter=""
exclude_filter=""
export_path=""
patches=PackedStringArray()
encryption_include_filters=""
encryption_exclude_filters=""
seed=0
encrypt_pck=false
encrypt_directory=false
script_export_mode=2
[preset.0.options]
custom_template/debug=""
```

```
custom_template/release=""
debug/export_console_wrapper=1
binary_format/embed_pck=false
texture_format/s3tc_bptc=true
texture_format/etc2_astc=false
shader_baker/enabled=false
binary_format/architecture="x86_64"
codesign/enable=false
codesign/timestamp=true
codesign/timestamp_server_url=""
codesign/digest_algorithm=1
codesign/description=""
codesign/custom_options=PackedStringArray()
application/modify_resources=true
application/icon=""
application/console_wrapper_icon=""
application/icon_interpolation=4
application/file_version=""
application/product_version=""
application/company_name=""
application/product_name=""
application/file_description=""
application/copyright=""
application/trademarks=""
application/export_angle=0
application/export_d3d12=0
application/d3d12_agility_sdk_multiarch=true
ssh_remote_deploy/enabled=false
ssh_remote_deploy/host="user@host_ip"
ssh_remote_deploy/port="22"
ssh_remote_deploy/extra_args_ssh=""
ssh_remote_deploy/extra_args_scp=""
ssh_remote_deploy/run_script="Expand-Archive -LiteralPath '{temp_dir}\\{archive_name}'
-DestinationPath '{temp_dir}'
$action = New-ScheduledTaskAction -Execute '{temp_dir}\\{exe_name}' -Argument
'{cmd_args}'
$trigger = New-ScheduledTaskTrigger -Once -At 00:00
$settings = New-ScheduledTaskSettingsSet -AllowStartIfOnBatteries -
DontStopIfGoingOnBatteries
$task = New-ScheduledTask -Action $action -Trigger $trigger -Settings $settings
Register-ScheduledTask godot_remote_debug -InputObject $task -Force:$true
Start-ScheduledTask -TaskName godot_remote_debug
while (Get-ScheduledTask -TaskName godot_remote_debug | ? State -eq running) { Start-
Sleep -Milliseconds 100 }
Unregister-ScheduledTask -TaskName godot_remote_debug -Confirm:$false -
ErrorAction:SilentlyContinue"
ssh_remote_deploy/cleanup_script="Stop-ScheduledTask -TaskName godot_remote_debug -
ErrorAction:SilentlyContinue
Unregister-ScheduledTask -TaskName godot_remote_debug -Confirm:$false -
ErrorAction:SilentlyContinue
Remove-Item -Recurse -Force '{temp_dir}'"
```

EXPORT.bat

```
@echo off
echo ZOMBIE DEFENSE 3D - Exporter
echo.
set BUILD_DIR=%~dp0BUILD
if not exist "%BUILD_DIR%" mkdir "%BUILD_DIR%"
set GODOT_PATH=
where godot.exe >nul 2>&1
if %ERRORLEVEL% EQU 0 (
    for /f "delims=" %%i in ('where godot.exe') do set GODOT_PATH=%%i
    goto :export
)
if exist "C:\Program Files\Godot\Godot_v4.2-stable_win64.exe" (
    set GODOT_PATH=C:\Program Files\Godot\Godot_v4.2-stable_win64.exe
```

```
    goto :export
)
echo ERROR: Godot not found
set /p GODOT_PATH=Path to Godot.exe:
:export
if not exist "%~dp0export_presets.cfg" (
    echo ERROR: export_presets.cfg not found
    echo Create export preset in Godot Editor first
    pause
    exit /b 1
)
echo Exporting...
"%GODOT_PATH%" --headless --export-release "Windows Desktop"
"%BUILD_DIR%\ZombieDefense3D.exe"
echo.
echo Done! File: %BUILD_DIR%\ZombieDefense3D.exe
pause
```

PLAY.bat

```
@echo off
set GAME_EXE=%~dp0BUILD\ZombieDefense3D.exe
if not exist "%GAME_EXE%" (
    echo ERROR: Game not found at %GAME_EXE%
    echo Run EXPORT.bat first
    pause
    exit /b 1
)
start "" "%GAME_EXE%"
```

START.bat

```
@echo off
echo ZOMBIE DEFENSE 3D - Launcher
echo.
set GODOT_PATH=
where godot.exe >nul 2>&1
if %ERRORLEVEL% EQU 0 (
    for /f "delims=" %%i in ('where godot.exe') do set GODOT_PATH=%%i
    goto :run
)
if exist "C:\Program Files\Godot\Godot_v4.2-stable_win64.exe" (
    set GODOT_PATH=C:\Program Files\Godot\Godot_v4.2-stable_win64.exe
    goto :run
)
if exist "%USERPROFILE%\Downloads\Godot_v4.2-stable_win64.exe" (
    set GODOT_PATH=%USERPROFILE%\Downloads\Godot_v4.2-stable_win64.exe
    goto :run
)
echo ERROR: Godot not found
echo Please enter path to Godot.exe
set /p GODOT_PATH=Path:
:run
echo Starting game...
"%GODOT_PATH%" --path "%~dp0" res://scenes/main.tscn
Pause
```

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інформаційних систем та технологій

«Методи штучного інтелекту для моделювання поведінки NPC у процесі розробки відеоігор»

Виконав: студент групи САДМ-61 Костянтин КОЛОСОВ

Науковий керівник: доцент кафедри ІСТ Ровіл НАФЄЄВ

м. Київ, 2026

Мета роботи

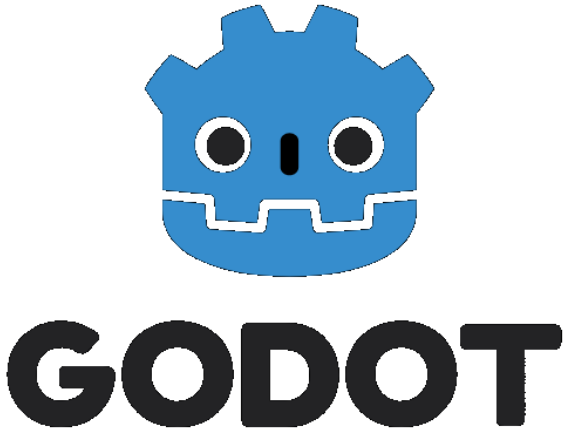
Розробити та реалізувати три принципово різні системи штучного інтелекту (FSM з нечіткою логікою, Behavior Tree з геометричним позиціонуванням, GOAP з Blackboard-координацією) для моделювання поведінки NPC у 3D-шутері Zombie Defense 3D на рушії Godot 4 та провести їх порівняльний аналіз за ефективністю, ресурсоспоживанням і впливом на геймплейне сприйняття гравця.

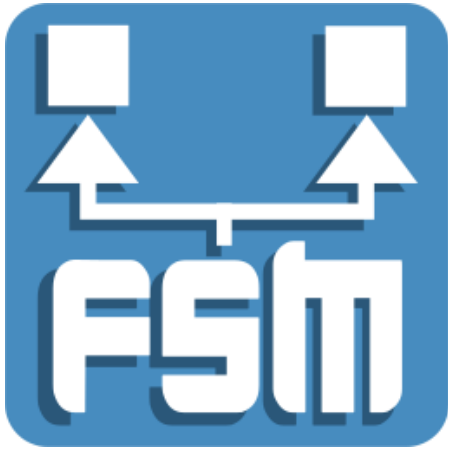
Об’єкт та предмет дослідження

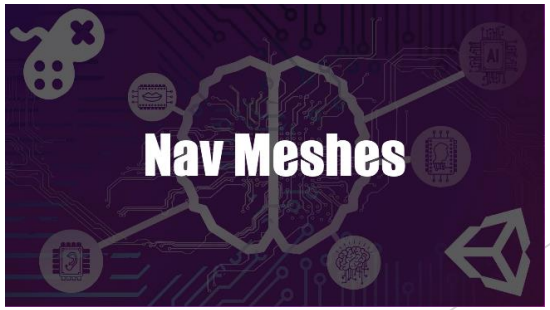
Об’єкт дослідження – процес моделювання інтелектуальної поведінки неігрових персонажів у тривимірних іграх жанру survival shooter з використанням відкритих рушіїв

Предмет дослідження – практичне застосування та порівняння реактивного (FSM+Fuzzy Logic), деліберативного (Behavior Tree) та гібридного (GOAP+Blackboard) підходів у єдиному прототипі Zombie Defense 3D

Використані технології







Порівняння основних парадигм AI в іграх

Парадигма	Роки масового використання	Складність реалізації	Гнучкість	Приклад ігор	Основна проблема
Скрипти/Hard-coded	1970–1990	Дуже низька	Низька	Pong, Space Invaders	Неможливо масштабувати
FSM/HFSM	1980–2010	Низька-Середня	Середня	Pac-Man, Half-Life, RE4	Вибух станів при складанні
Behavior Trees	2004–дотепер	Середня	Висока	Halo 3, The Last of Us, Gears	Складно налаштовувати баланс
GOAP	2005–дотепер	Висока	Дуже висока	F.E.A.R., STALKER, Cyberpunk	Вимагає ретельного дизайну дій
Utility AI	2010–дотепер	Середня-висока	Найвища	The Sims 3, RDR2, Starfield	Потребує тонкого тюнінгу ваг

Порівняння скінчених автоматів станів та дерев поведінки

Критерій	FSM (Finite State Machine)	Behavior Trees
Складність реалізації	Дуже низька для простих NPC	Середня, потребує розуміння композиції вузлів
Масштабованість	Погана (експоненційний ріст переходів)	Висока (модульність, легке додавання поведінки)
Зручність для дизайнерів	Низька (потрібен код для нових переходів)	Висока (візуальні редактори)
Пріоритетність поведінки	Складно (потрібні складні умови)	Природно (через Selector)
Пам’ять стану	Явна (поточний стан)	Неявна (позиція в дереві)
Продуктивність	Відмінна	Відмінна (трохи вища витрата через рекурсію)
Типові проєкти	Класичні 2D/3D шутери, платформери	AAA-проєкти, open-world, стелс ігри

Порівняльна характеристика сучасних ML-підходів до адаптивної поведінки NPC

Підхід	Час навчання	Адаптація під час гри	Потреба в даних/симуляціях	Ризик «катастрофічного забування»	Найкраще застосування
Чисте RL (PPO, SAC)	Середній-високий	Висока (онлайн)	Висока (мільйони кроків)	Середній	Динамічні PvE, horror, stealth
NeuroEvolution (NEAT, CMA-ES)	Дуже високий (офлайн)	Низька (офлайн)	Висока (паралельні симуляції)	Низький	Різноманітність поведінки в одному типі NPC
Гібрид RL + Еволюція	Високий (офлайн + онлайн)	Дуже висока	Середня	Низький	Довготривалі кампанії, MMO
Imitation + RL (GAIL, BC+PPO)	Середній	Висока	Середня (демонстрації)	Високий	NPC, що копіюють стиль гравця

Вимоги до реалістичності поведінки NPC у різних жанрах ігор

Жанр	Основна роль NPC	Ключова вимога до AI	Приклади ігор	Тип AI, що переважає
Аркадні шутери	Мішені для стрільби	Швидка реакція, прості шаблони поведінки	Doom (2016), Doom Eternal	Реактивний (FSM)
Survival Shooter	Створення постійного тиску	Динамічна адаптація, варіативність атак	Left 4 Dead, Zombie Defense 3D	Гібридний (FSM + BT + GOAP)
Тактичні шутери	Координація, фланкування	Командна тактика, прийняття рішень	Rainbow Six Siege, Ready or Not	Behavior Trees + Utility AI
RPG / Open World	Соціальна взаємодія, правдоподібність	Реалістичні діалоги, симуляція повсякденного життя	The Witcher 3, Baldur’s Gate 3	GOAP, діалогова система
Survival Horror	Психологічний тиск, непередбачуваність	Несподівані реакції, складна сенсорна модель	Resident Evil Village, Alien: Isolation	Складні FSM + сенсорні системи
Стратегії	Ефективне управління ресурсами	Планування, оптимізація дій	StarCraft II, Total War	Utility AI + планування

UML-діаграми

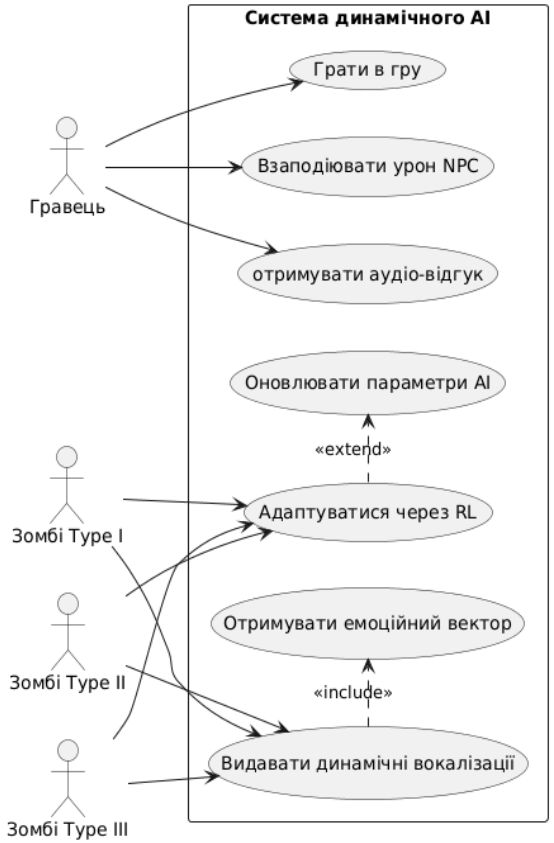


Рис. 3.1 Діаграма варіантів використання інтегрованої моделі

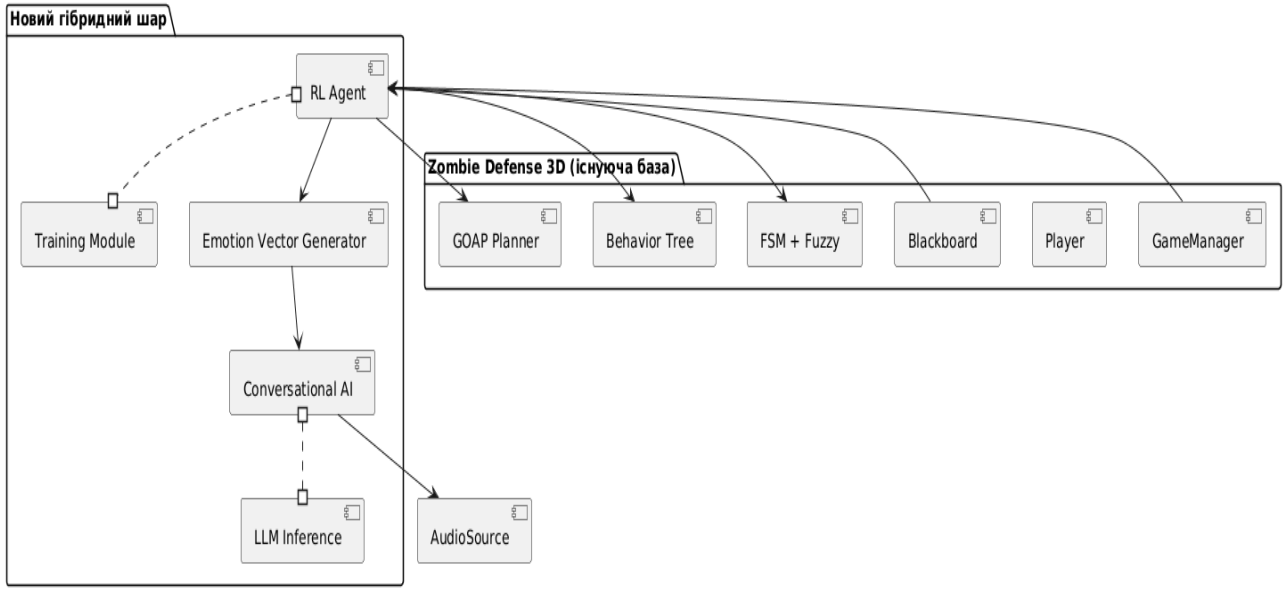


Рис. 3.2 Діаграма компонентів гібридної архітектури

UML-діаграми

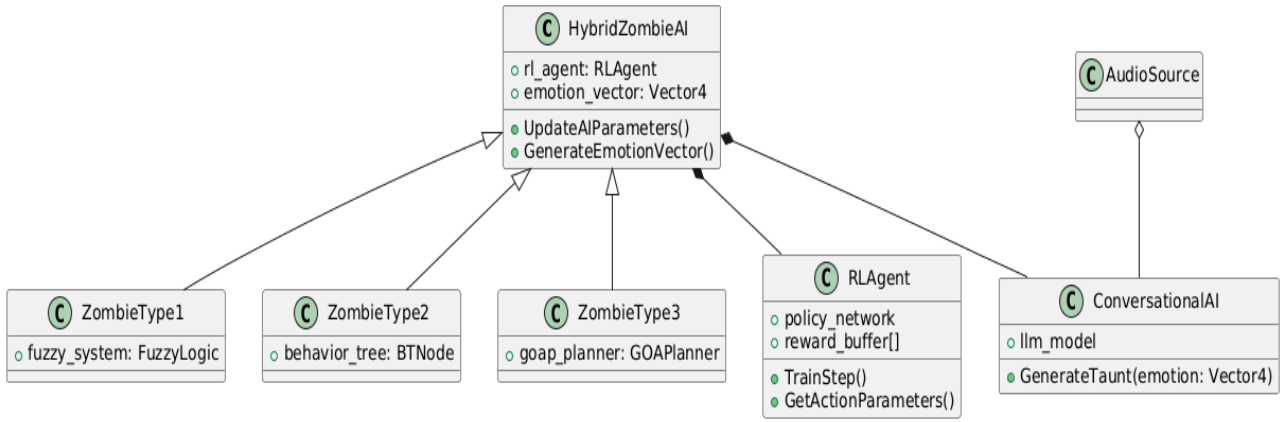


Рис. 3.3 Діаграма класів розширеної моделі

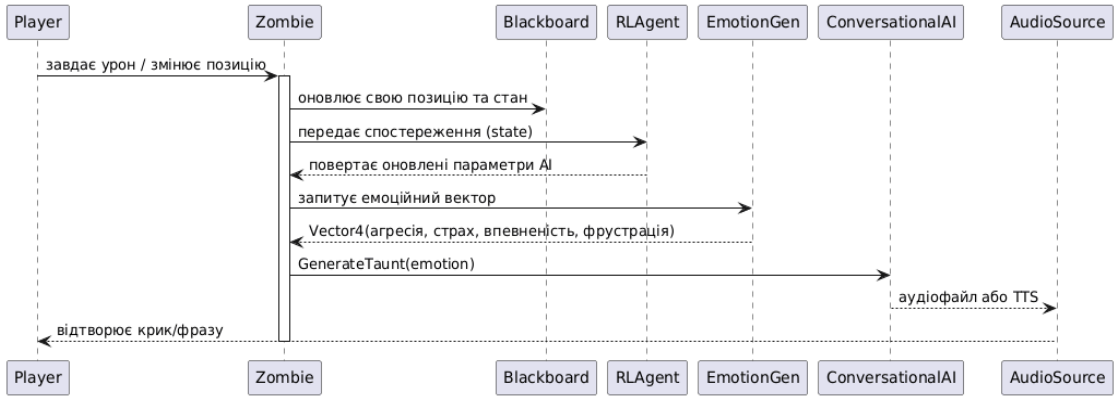


Рис. 3.4 Діаграма послідовності для одного кадру взаємодії

UML-діаграми

Рис. 3.5 Діаграма діяльності гібридного NPC

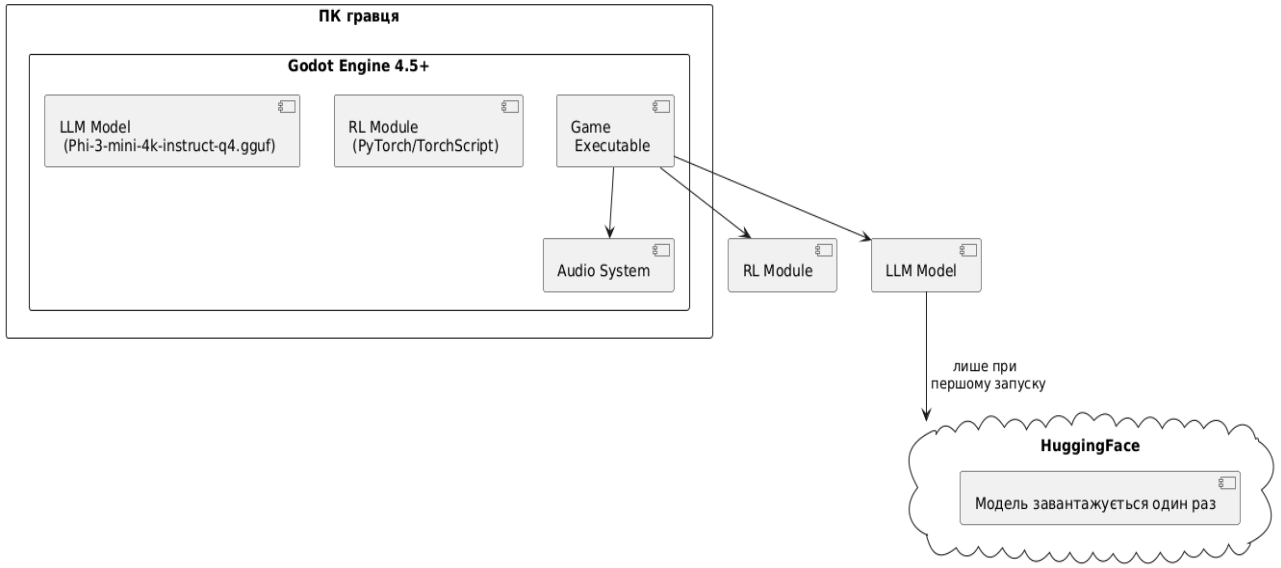


Рис. 3.6 Діаграма розгортання гібридної моделі

Реалізація прототипу

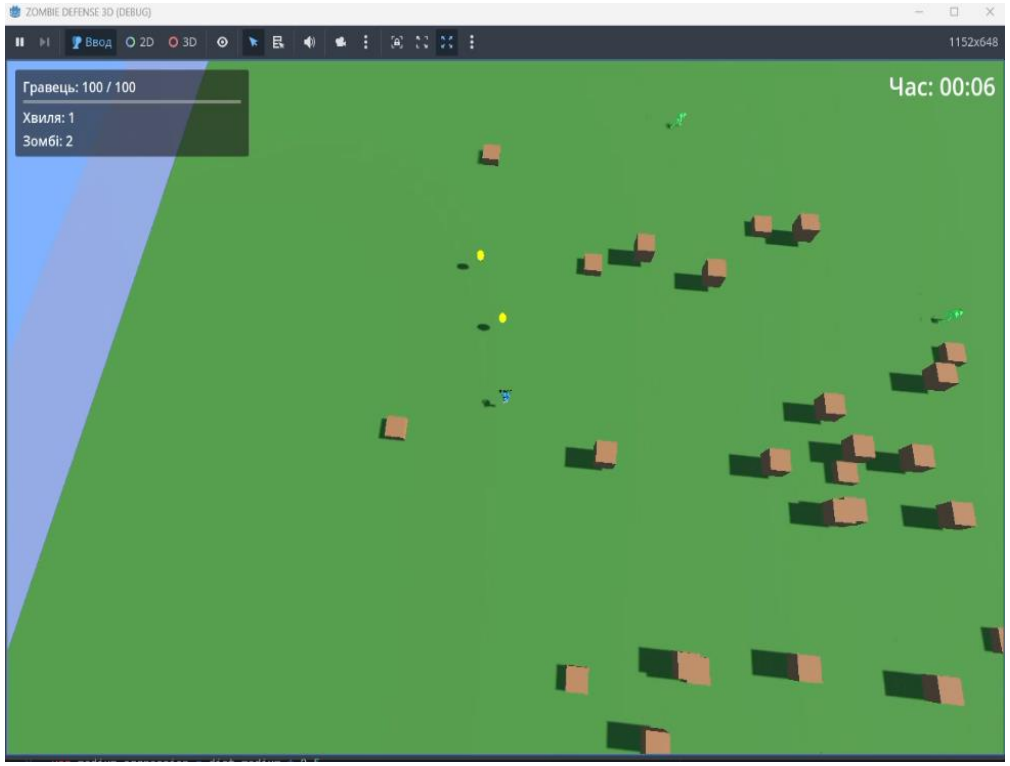


Рис. 4.1 Хвиля 1. Поява двох зелених зомбі Type I (FSM + Fuzzy Logic), гравець ще не отримав ушкоджень

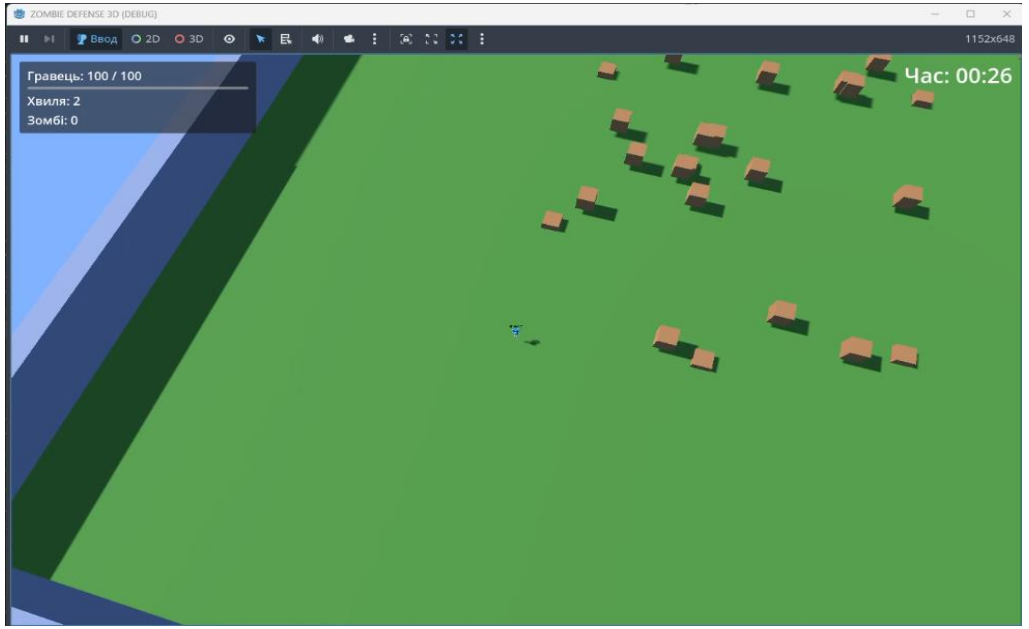


Рис. 4.2 Хвиля 2. Успішне завершення, всі зомбі знищені, лічильник «Зомбі:0»

Реалізація прототипу

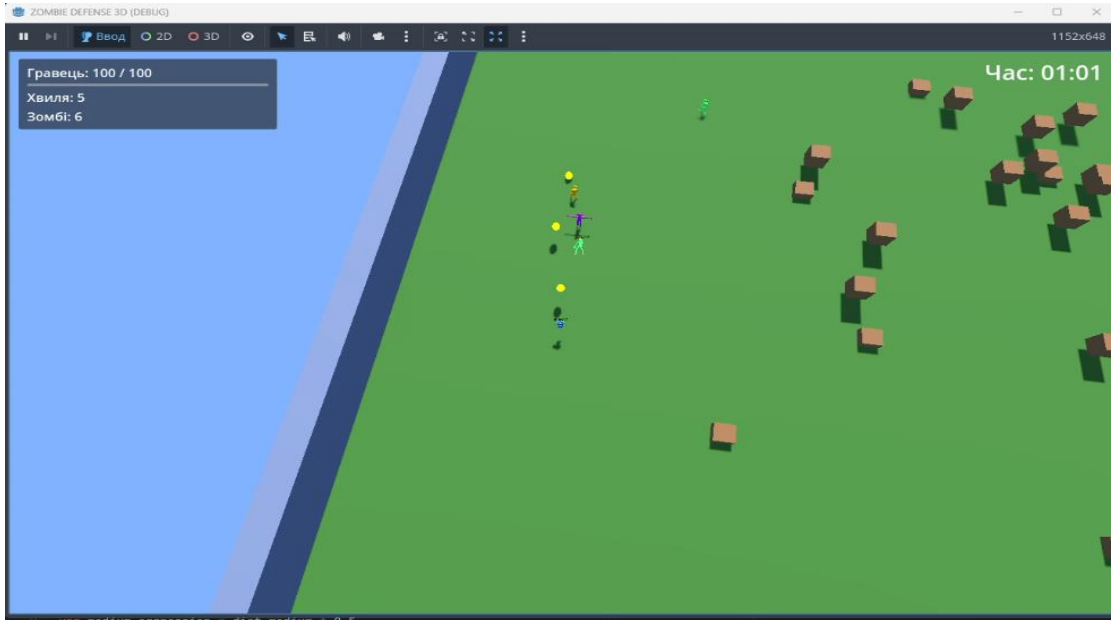


Рис. 4.3 Хвиля 5. Шість зомбі (зелені, помаранчеві та фіолетові), гравець тримає дистанцію

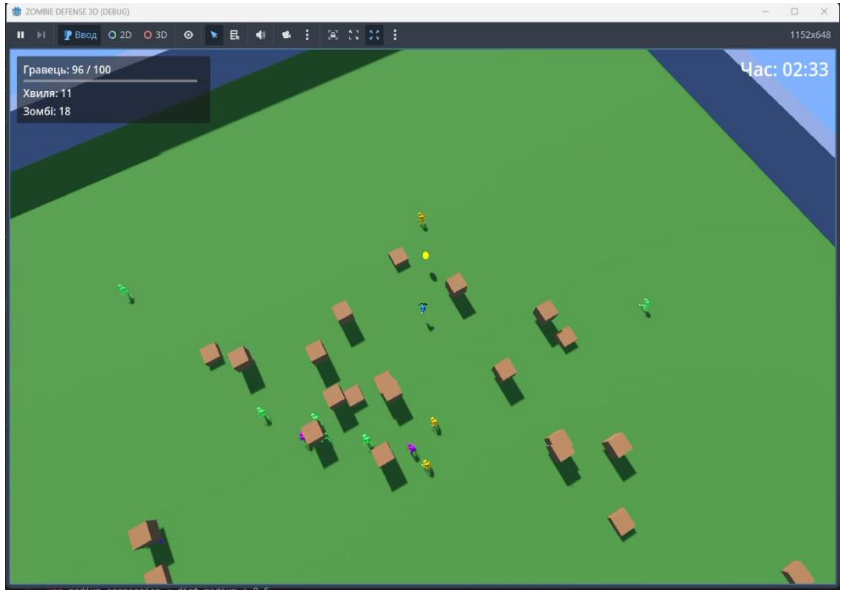


Рис. 4.4 Хвиля 11. Одинадцять зомбі усіх типів, гравця оточують з різних боків

Реалізація прототипу

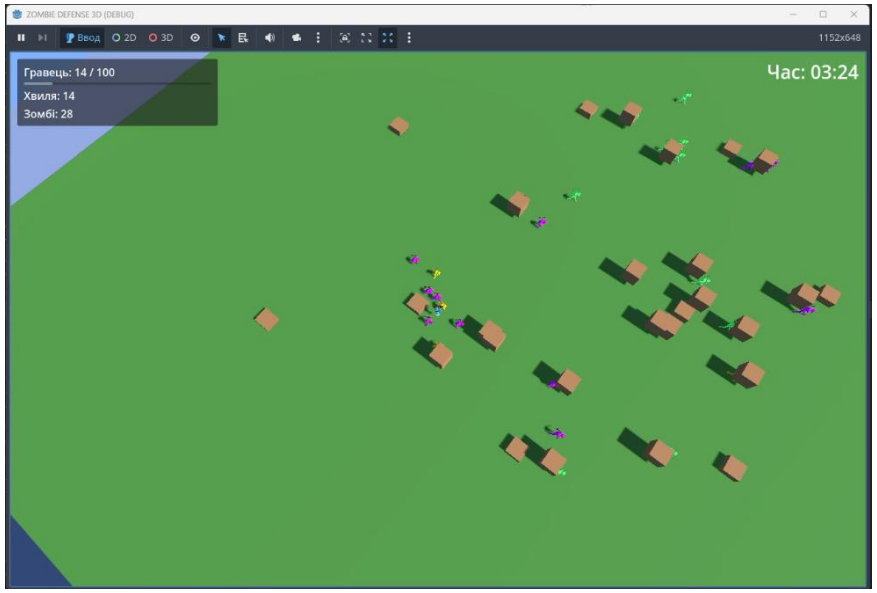


Рис. 4.5 Хвиля 14. Одинадцять зомбі, гравця щільно оточено з різних боків, рівень здоров’я гравця нижче за 30%

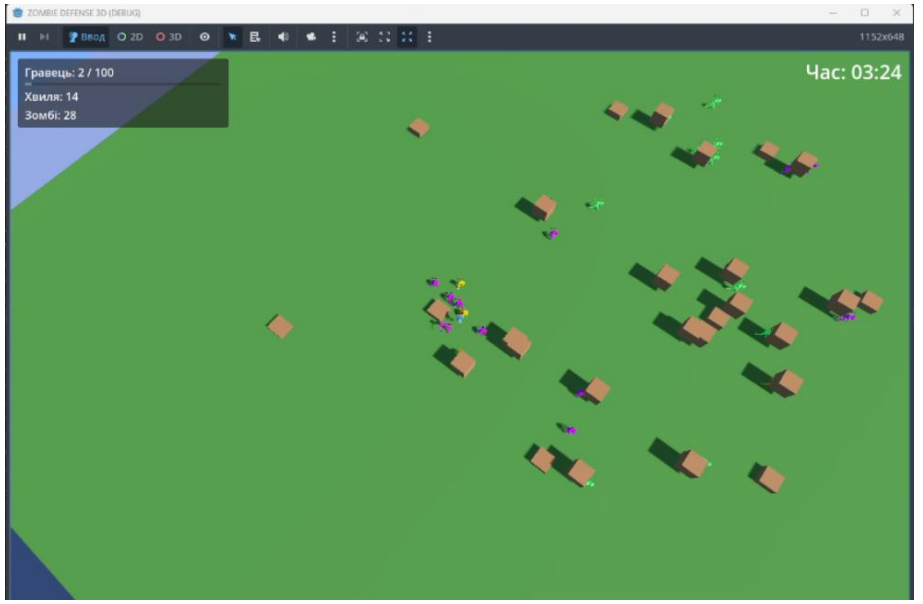


Рис. 4.6 Хвиля 14. Рекордне виживання (28 зомбі), гра завершується смертю гравця через неможливість одночасно контролювати всі напрямки

Висновки

Експериментально доведено, що комплексне застосування класичних архітектур ШІ дозволяє створити високоякісних NPC навіть в умовах обмежених ресурсів інді-розробки.

Гібридний підхід до створення ШІ суттєво підвищує сприйняття інтелектуальності противників та емоційний вплив на гравця порівняно з традиційними системами.

Модифікація реактивних моделей за допомогою нечіткої логіки є ефективним рішенням для підвищення правдоподібності поведінки без особливого навантаження та обчислювальних затрат

Реалізація складної координації NPC (GOAP) на відкритому рушії без використання професійних інструментів є практично досяжною та переносимою на інші платформи

Апробація:

1. VI Всеукраїнська науково-практична конференція «Telecommunication: problems and innovation», розділ «Інформаційні системи та технології», тези на тему «Використання штучного інтелекту для поведінки NPC при розробці відеоігор». Очікується публікація.
2. Другий розділ журналу «Зв'язок». Стаття на тему «Методи штучного інтелекту для моделювання поведінки NPC у процесі розробки відеоігор». Очікується публікація.

Дякую за увагу!