МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«СТВОРЕННЯ NPC-ПЕРСОНАЖІВ ІЗ ЗАСТОСУВАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ»**

Виконав: студент 4 курсу, групи 6.1221-з
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
І.О Караулов
(ініціали та прізвище)
Керівник професор кафедри комп'ютерних наук, доцент, д.е.н. Полуектова Н.Р.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А. О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., професор

_____ Шило Г.В.
(підпис)

" ____ " _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Караулову Іллі Олеговичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Створення NPC-персонажів із застосуванням штучного інтелекту

керівник роботи Полуектова Наталія Робертівна, д.е.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » 12 2024 року № 2002-с

2. Строк подання студентом роботи 15.05.2005 р

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Спроєктувати та розробити NPC-персонаж з використанням одного з методів Штучного інтелекту

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 10.01.2025 р

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	01.02.2025	
2.	Збір вихідних даних.	01.03.2025	
3.	Обробка методичних та теоретичних	25.03.2025	
	джерел.		
4.	Розробка першого та другого розділу.	10.04.2025	
5.	Розробка третього розділу.	05.05.2025	
6.	Оформлення та нормоконтроль	14.05.2025	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	29.05.2025	

Студент _____ І.О Караулов
(підпис) (ініціали та прізвище)

Керівник роботи _____ Н.Р. Полуектова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Створення NPC-персонажів із застосуванням штучного інтелекту»: 34 с., 10 рис., 2 табл., 9 джерел.

КІНЦЕВІ АВТОМАТИ, ШТУЧНИЙ ІНТЕЛЕКТ, FSM, NPC, UNITY.

Об’єкт дослідження – непрограмовані ігрові персонажі (NPC) у відеоіграх, їхня поведінка, взаємодія з гравцем та навколишнім середовищем за допомогою методів штучного інтелекту.

Мета роботи: Реалізувати модель поведінки NPC із застосуванням алгоритмів штучного інтелекту для підвищення реалістичності, адаптивності та варіативності взаємодії з гравцем у відеоіграх.

Методи дослідження – огляд, аналіз та синтез, порівняння, експеримент.

В результаті в середовищі Unity реалізовано модель неігрового персонажа, поведінка якого керується за допомогою метода кінцевих автоматів.

SUMMARY

Bachelor’s Qualification Thesis: "Creation of NPC Characters Using Artificial Intelligence": 34 pages, 10 figures, 2 tables, 9 references.

ARTIFICIAL INTELLIGENCE, FINITE AUTOMATA, FSM, NPC, UNITY.

Object of the study: non-playable characters (npcs) in video games, their behavior, and their interaction with the player and the environment using artificial intelligence methods.

Objective: to implement a model of npc behavior using artificial intelligence algorithms to enhance realism, adaptability, and variability in interaction with the player in video games.

Research methods: review, analysis and synthesis, comparison, experiment.

Result: an npc behavior model was implemented in the unity environment, where the behavior is controlled using the finite state machine (fsm) method.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 7
1 Теоретичні основи створення NPC-персонажів 9
1.1 Поняття NPC у комп'ютерних іграх та проблеми розробки 9
1.2 Роль NPC у сучасних інтерактивних системах 10
1.3 Методи побудови та реалізації штучного інтелекту в NPC 12
1.4 Аналіз сучасних ігрових рушіїв та AI-фреймворків для NPC 13
2 Проектування системи NPC з використанням ШІ 16
2.1 Постановка задачі та вибір інструментів 16
2.2 Вибір методу реалізації 16
2.3 Обгрунтування інструментів реалізації 17
2.4 Архітектура системи NPC 18
3 Реалізація та тестування npc-персонажів 21
3.1 Розробка NPC на базі вибраної моделі 21
3.2 Реалізація сценаріїв поведінки 24
3.3 Інтеграція NPC у прототип гри або симуляційне середовище 25
3.4 Тестування системи NPC: сценарії, результати, проблеми 27
3.5 Аналіз ефективності реалізованих рішень 30
Висновки 31
Перелік посилань 33

ВСТУП

У сучасному світі відеоігри є однією з найдинамічніших і найінноваційніших галузей цифрових технологій. Із розвитком апаратного забезпечення, графічних рушіїв та інструментів розробки зростають очікування гравців щодо якості, реалістичності та глибини ігрового процесу. Одним із ключових факторів, що впливають на ігровий досвід, є поведінка неігрових персонажів (NPC, англ. non-playable characters), які взаємодіють з гравцем, створюють атмосферу, виступають союзниками або противниками, формують сюжетні лінії та ігрові виклики.

Традиційні NPC у простих іграх функціонували за заздалегідь визначеним набором скриптів або шаблонів поведінки, що обмежувало їхню адаптивність і робило поведінку передбачуваною. Проте з розвитком методів штучного інтелекту відкрилися нові можливості для створення більш складних, гнучких і реалістичних моделей поведінки персонажів. Сучасні підходи дозволяють NPC аналізувати середовище, реагувати на дії гравця, змінювати свою стратегію залежно від обставин, і тим самим створювати більш глибокий і захоплюючий ігровий процес.

У рамках цієї кваліфікаційної роботи розглядається підхід до моделювання поведінки NPC із використанням методу кінцевих автоматів (FSM – Finite State Machine), що дозволяє структуровано та ефективно організувати логіку поведінки персонажа. Кінцеві автомати є одним із найпоширеніших підходів у геймдизайні для реалізації станів, у яких може перебувати NPC, а також переходів між цими станами на основі певних умов або подій.

Роботу реалізовано в середовищі Unity – популярному ігровому рушії, що широко використовується для створення 2D- та 3D-ігор завдяки своїй гнучкості, підтримці C# як основної мови програмування та розвиненій екосистемі інструментів. У рамках реалізації було створено демонстраційну

сцену з неігровим персонажем, який демонструє зміну поведінки відповідно до внутрішнього стану, що керується кінцевим автоматом. Наприклад, NPC може перебувати у станах патрулювання, переслідування або атаки залежно від дій гравця та конфігурації середовища.

Актуальність теми зумовлена потребою у створенні адаптивних та інтелектуально керованих персонажів у сучасних відеоіграх, що сприяє підвищенню рівня занурення гравця у віртуальний світ. Використання методів штучного інтелекту дозволяє створити поведінку, яка є менш шаблонною та більш природною, що в цілому покращує сприйняття гри.

Мета цієї роботи полягає в розробці та реалізації моделі NPC, яка використовує метод кінцевих автоматів у поєднанні з елементами штучного інтелекту для створення більш реалістичної взаємодії з гравцем. Основна увага приділяється логіці прийняття рішень, ефективності реалізації та можливості масштабування розробленої моделі для більш складних сценаріїв.

Таким чином, дана робота є прикладом застосування класичних моделей керування поведінкою у поєднанні з сучасними інструментами ігрової розробки та має практичну цінність для розробників, які прагнуть створювати більш переконливих та адаптивних NPC у своїх іграх.

1 ТЕОРЕТИЧНІ ОСНОВИ СТВОРЕННЯ NPC-ПЕРСОНАЖІВ

1.1 Поняття NPC у комп'ютерних іграх та проблеми розробки

Неігровий персонаж (NPC) – це будь-який персонаж у грі, який не контролюється безпосередньо гравцем. Натомість, його дії, поведінка та взаємодія з ігровим світом і гравцем визначаються програмним кодом гри, часто з використанням елементів штучного інтелекту або заздалегідь визначених скриптів.

Термін виник із традиційних настільних ігор, відомих як рольові ігри (RPG), у яких дії персонажів контролювалися через наратив, створений гравцем, що адміністрував гру. У цифрових іграх роль NPC полягає в імітації поведінки та раціональності справжніх гравців [1], діючи з мінімальним або повністю відсутнім людським втручанням. Такі агенти контролюються штучним інтелектом (AI) – однією з галузей інформатики, що має десятилітній досвід у наукових дослідженнях, пов'язаних із цифровими іграми. У той час як індустрія прагне задовольнити вимоги клієнтів, дотримуючись критеріїв якості, наука зосереджується на розробці нових моделей, технік і методів, які роблять NPC більш переконливими в запропонованому середовищі, тестуючи людське сприйняття штучного та реального.

Технологічна еволюція як у сфері програмного, так і апаратного забезпечення дала змогу зробити поведінку NPC інтуїтивнішою, застосовуючи різноманітні процеси, що традиційно використовуються провідними ігровими компаніями. Серед них прийняття рішень – це процес, що визначає, як персонаж реагує на ситуації у віртуальному середовищі. У психологічному розумінні цей процес передбачає суб'єкта, який виявляє та оцінює стимули навколо себе, відтворюючи відповідь або ланцюг реакцій як поведінку. Штучний інтелект NPC ґрунтується на заздалегідь визначеному наборі правил,

який намагається імітувати найбільш відповідну поведінку згідно з даними, отриманими з навколишнього середовища.

Водночас заздалегідь визначений набір правил може відтворювати неприродну поведінку, яка здається нелогічною або передбачуваною з погляду гравця, що послаблює ефект занурення в цифрову гру. З іншого боку, наукові дослідження постійно спрямовані на перенесення людського сприйняття на рівень програмного забезпечення – завдання, що потребує вивчення когнітивної психології. Дослідження Пікар [2] підтверджують гіпотезу, що якщо мета – зробити штучних агентів по-справжньому розумнішими та забезпечити природнішу реакцію, то дослідники мають надати їм здатність розпізнавати, розуміти та навіть виражати емоції.

Згідно з [3], розробники ігор зазначають, що деякі середні або великі компанії не виділяють достатньо часу та ресурсів для розробки складного набору правил для прийняття рішень NPC. Крім того, [4] згадує компанії, які досі дотримуються застарілого підходу жорсткого програмування, тобто частого редагування та перекомпіляції вихідного коду замість використання підходу, що не потребує втручання програміста. Тому слід надавати перевагу технікам, які легко реалізуються, здатні продукувати правдоподібні результати з низькими обчислювальними витратами, особливо тим підходам, що гарантують «жорсткий контроль над можливими результатами, запобігаючи відповідям поза контекстом».

1.2 Роль NPC у сучасних інтерактивних системах

Сьогодні NPC є невіддільною частиною ігрового досвіду, відіграючи багатогранні ролі, які сприяють зануренню гравця, розвитку сюжету та створенню живого та динамічного світу.

NPC можуть виконувати різні функції в ігровому процесі:

- союзники: допомагають гравцеві в бою, виконанні завдань або наданні інформації (наприклад, напарники у стрілялках або торговці у рольових іграх);
- вороги: атакують гравця, створюючи перешкоди та виклики (наприклад, монстри, солдати, бандити);
- нейтральні персонажі: взаємодіють з гравцем лише в певних умовах або слугують частиною навколишнього середовища (наприклад, мешканці міста, тварини, торговці, що не впливають на сюжет);
- квестодавці: надають гравцеві завдання, сприяючи просуванню по сюжету;
- інформаційні персонажі: забезпечують гравця необхідною інформацією щодо сюжету або навколишнього світу.

NPC надають грі відчуття реальності та наповненості. Вони ходять містами, виконують повсякденні справи, взаємодіють один з одним, що створює ілюзію живого світу (наприклад, серія *The Elder Scrolls* або *GTA*).

NPC можуть давати завдання, розповідати історії або надавати підказки для подальшого проходження. Наприклад, у *The Witcher 3* персонажі, що зустрічаються у тавернах або на шляхах, часто стають джерелом додаткових квестів.

У деяких іграх NPC можуть змінювати своє ставлення до гравця залежно від його дій (наприклад, *Mass Effect*, *Dragon Age*). Вони можуть вступати в діалоги, ображатися, радіти, проявляти підтримку або агресію.

NPC виконують функції торговців, ремісників та посередників у світі гри. Вони купують і продають предмети, що дозволяє гравцеві взаємодіяти з економікою світу (*Skyrim*, *Path of Exile*).

NPC часто відіграють важливу роль у розвитку сюжету. Вони можуть бути головними персонажами в історії, або ж слугувати мостом між різними частинами сюжету (*Cyberpunk 2077*, *Red Dead Redemption 2*).

1.3 Методи побудови та реалізації штучного інтелекту в NPC

Для створення інтелектуальної поведінки NPC-персонажів зараз використовуються різні підходи. Нижче наведені основні з них.

Кінцеві Автомати (Finite State Machines, FSM)

NPC може перебувати лише в одному з **кінцевого числа станів** у будь-який момент часу (наприклад, "Патрулювання", "Атака", "Бездіяльність"). Переходи між станами відбуваються за чітко визначеними умовами або подіями.

Зазвичай використовується проста логіка if-else або switch-case для перевірки поточного стану та умов переходу. Об’єктно-орієнтоване програмування може застосовувати шаблон "Стан" (State Pattern), де кожен стан є окремим класом, що робить систему більш модульною.

Дерева поведінки (Behavior Trees, BT)

Цей метод дозволяє описати складну поведінку NPC у вигляді ієрархічної структури, що включає умови, завдання та переходи між ними. Поведінка розбивається на підзадачі, які перевіряються послідовно або паралельно.

Кожен вузол є об’єктом, який повертає один із трьох результатів: Успіх, Невдача або Виконання. Композитні вузли (Селектори, Послідовності) керують потоком виконання своїх дочірніх вузлів. Листові вузли (Дії, Умови) виконують ігрову логіку.

Нейронні Мережі

Нейронні мережі (НМ) – це обчислювальні моделі, натхненні структурою та функціонуванням людського мозку. Вони складаються зі взаємопов’язаних вузлів (нейронів), організованих у шари. НМ "навчаються" розпізнавати закономірності та приймати рішення на основі великих обсягів даних. Для ШІ NPC це означає, що замість того, щоб програмувати кожну реакцію, ми навчаємо мережу бажаній поведінці.

Для навчання нейронної мережі потрібні дані. Це можуть бути записи дій гравців (для імітаційного навчання) або результати взаємодії NPC з ігровим світом (для навчання з підкріпленням).

Агентні Системи (Багатоагентні Системи)

Агентні системи моделюють поведінку NPC як сукупність незалежних, автономних агентів, кожен з яких має власні правила, цілі та здатність взаємодіяти з навколишнім середовищем та іншими агентами. Складна колективна поведінка виникає внаслідок взаємодії цих індивідуальних, часто простих, правил:

- кожен NPC розглядається як окремий агент;
- кожен агент має свій власний набір правил (які можуть бути реалізовані за допомогою FSM, BT, систем правил або навіть маленьких нейронних мереж);
- агенти можуть спілкуватися між собою (наприклад, передавати інформацію про ворогів), співпрацювати або конкурувати;
- агенти сприймають ігровий світ через сенсори (зір, слух) і впливають на нього через актори (рух, атака);
- загальна поведінка системи не програмується безпосередньо, а виникає з взаємодії агентів.

1.4 Аналіз сучасних ігрових рушіїв та AI-фреймворків для NPC

У розробці комп’ютерних ігор особливу роль відіграє вибір ігрового рушія та інструментів для реалізації штучного інтелекту неігрових персонажів (NPC). Від цього вибору залежить не лише якість взаємодії гравця з NPC, а й швидкість розробки, зручність налагодження та можливості масштабування проекту.

Unreal Engine (особливо версія 5) – потужний рушій для створення ігор із високим рівнем графіки. Часто використовується у AAA-проєктах (високобюджетні ігри).

Unity - це кросплатформовий ігровий рушій, створений компанією Unity Technologies в 2005 році. З його допомогою роблять одиночні та розраховані на багато користувачів ігри з сучасною 2D-і 3D-графікою для таких платформ, як ПК, PlayStation, Xbox, Nintendo Switch.

Godot Engine - відкритий кросплатформовий ігровий рушій для розробки 2D/3D-відеоігор і програм для ПК, мобільних пристроїв, веб-платформ.

CryEngine – потужний рушій, відомий завдяки серії ігор *Crysis*. Має високий рівень графіки та фізики, але менш популярний серед інді-розробників через складність і вузьку спеціалізацію.

Далі наведемо огляд деяких спеціалізованих фреймворків.

Unity Machine Learning Agents Toolkit (ML-Agents Toolkit) – проект з відкритим вихідним кодом, що дозволяє іграм і симуляціям служити середами для навчання інтелектуальних агентів. Агенти можуть навчатися з використанням навчання з підкріпленням, навчання імітацією, нейроеволюції або інших методів машинного навчання через простий використання API Python.

PyTorch - це програмний фреймворк глибокого навчання з відкритим вихідним кодом, що використовується для побудови нейронних мереж, що поєднує бібліотеку машинного навчання (ML) Torch з API високого рівня на основі Python. Його гнучкість та простота використання, серед інших переваг, зробили його провідним фреймворком ML для академічних та дослідницьких спільнот.

Kythera - це потужне рішення AI, розроблене для надання комплексного рішення для розробників ігор. Воно пропонує широкий спектр повністю інтегрованих компонентів, які працюють разом без проблем, щоб пожвавити ваших ігрових персонажів. З Kythera можна створюввати високопродуктивний

та масштабований AI, який налаштовується, розширюється та написаний на C++.

Таким чином, сучасні ігрові рушії пропонують потужні вбудовані інструменти для розробки AI, особливо це стосується Behavior Trees та систем навігації. Для більш складних, адаптивних та динамічних поведінок розробники все частіше звертаються до передових фреймворків, таких як Unity ML-Agents, або інтегрують власні рішення на базі GOAP, Utility AI, а також експериментують з машиною навчання. Вибір конкретного підходу залежить від розміру проекту, доступних ресурсів та бажаного рівня реалізму та інтелектуальності NPC.

2 ПРОЕКТУВАННЯ СИСТЕМИ NPC З ВИКОРИСТАННЯМ ШІ

2.1 Постановка задачі та вибір інструментів

Цей розділ присвячений розробці інтелектуальної системи керування NPC, яка легко інтегрується в інтерактивне ігрове середовище. Головна мета – створити модель поведінки NPC, що ефективно реагує на дії гравця, автономно пересувається ігровим світом та приймає обґрунтовані рішення залежно від поточної ситуації.

Цілі NPC:

- переміщення між заданими точками маршруту (патрулювання);
- виявлення гравця в радіусі (зона бачення);
- перехід до режиму переслідування;
- повернення до патрулювання, якщо гравець зник.

2.2 Вибір методу реалізації

У рамках проєкту було проаналізовано різні моделі побудови поведінки NPC, зокрема Behavior Trees, агентні системи, нейронні мережі та Finite State Machines (FSM). Враховуючи характер поставленої задачі, обмеженість функціоналу NPC та необхідність простої реалізації – було обрано модель FSM, яка є найбільш ефективною для створення базової, але надійної поведінки персонажа.

FSM (кінцевий автомат) – це модель, яка описує поведінку системи через набір чітко визначених станів. У будь-який момент NPC перебуває лише в одному стані, і переходить між ними за певними умовами (тригерами). Такий підхід дозволяє реалізувати логіку, яка добре масштабується і легко підтримується.

У межах реалізації NPC були визначені такі стани:

– patrol – NPC переміщується між заданими точками на карті;
– chase – NPC переслідує гравця, якщо він потрапив у зону виявлення;
– return – NPC повертається до маршруту патрулювання, якщо втратив гравця з поля зору.

Кожен стан реалізовано як окрему гілку логіки всередині конструкції switch-case у скрипті (рис. 2.1).

```
switch (currentState)
{
    case NPCState.Patrol:
        Patrol();
        break;
    case NPCState.Chase:
        Chase();
        break;
    case NPCState.Return:
        ReturnToPatrol();
        break;
}
```

Рисунок 2.1 – Логіка всередині конструкції switch-case у скрипті

FSM дозволяє ефективно організувати поведінку NPC без складної математики або навчання, що і робить її оптимальним вибором у рамках поставленого проєкту.

2.3 Обгрунтування інструментів реалізації

Для реалізації NPC було обрано рушій Unity через його простоту у використанні, підтримку 2D/3D-середовищ, розвинені інструменти для роботи з навігацією та анімаціями, а також зручне середовище для скриптової логіки.

Версія 2021+ забезпечує:

– покращену продуктивність та оптимізацію рендерингу;

- підтримку URP і HDRP, що дає змогу адаптувати якість графіки під різні платформи;
- розширену систему DOTS (Data-Oriented Technology Stack) – корисну для великих кількостей NPC;
- вбудовані засоби відладки, профілювання і тестування ігрової логіки.

Таким чином, вибір нової версії Unity сприяє стабільній та масштабованій реалізації поведінки NPC.

Мова програмування, що використовується в Unity це C#. Вона є сучасною об’єктно-орієнтованою мовою, що забезпечує зручний синтаксис, високу читабельність коду та велику кількість бібліотек для роботи зі станами, подіями, таймерами тощо. Таким чином, комбінація Unity + C# дозволяє ефективно реалізувати інтелектуальну поведінку NPC з урахуванням сучасних вимог до геймдизайну та гнучкості коду.

Animator Controller це графічний інструмент Unity, який дозволяє задавати переходи між анімаційними станами, що надає можливість пов’язати анімаційні стани з логікою FSM.

Основні переваги:

- підтримка умовних переходів, параметрів (bool, int, trigger);
- можливість створити анімаційні дерева поведінки, що відповідають діям NPC (ходьба, біг, атака, смерть).

Animator Controller дозволяє створити візуально узгоджену та логічну поведінку, де кожна дія NPC супроводжується відповідною анімацією, синхронізованою з внутрішньою логікою.

2.4 Архітектура системи NPC

Архітектура системи NPC є основою для реалізації його інтелекту та взаємодії з ігровим світом. Вона повинна бути модульною, масштабованою та

легко розширюваною, щоб підтримувати різноманітні типи поведінки та складність ігрових сценаріїв. Зазвичай, архітектура системи NPC включає наступні ключові компоненти, які працюють разом (рис. 2.2).

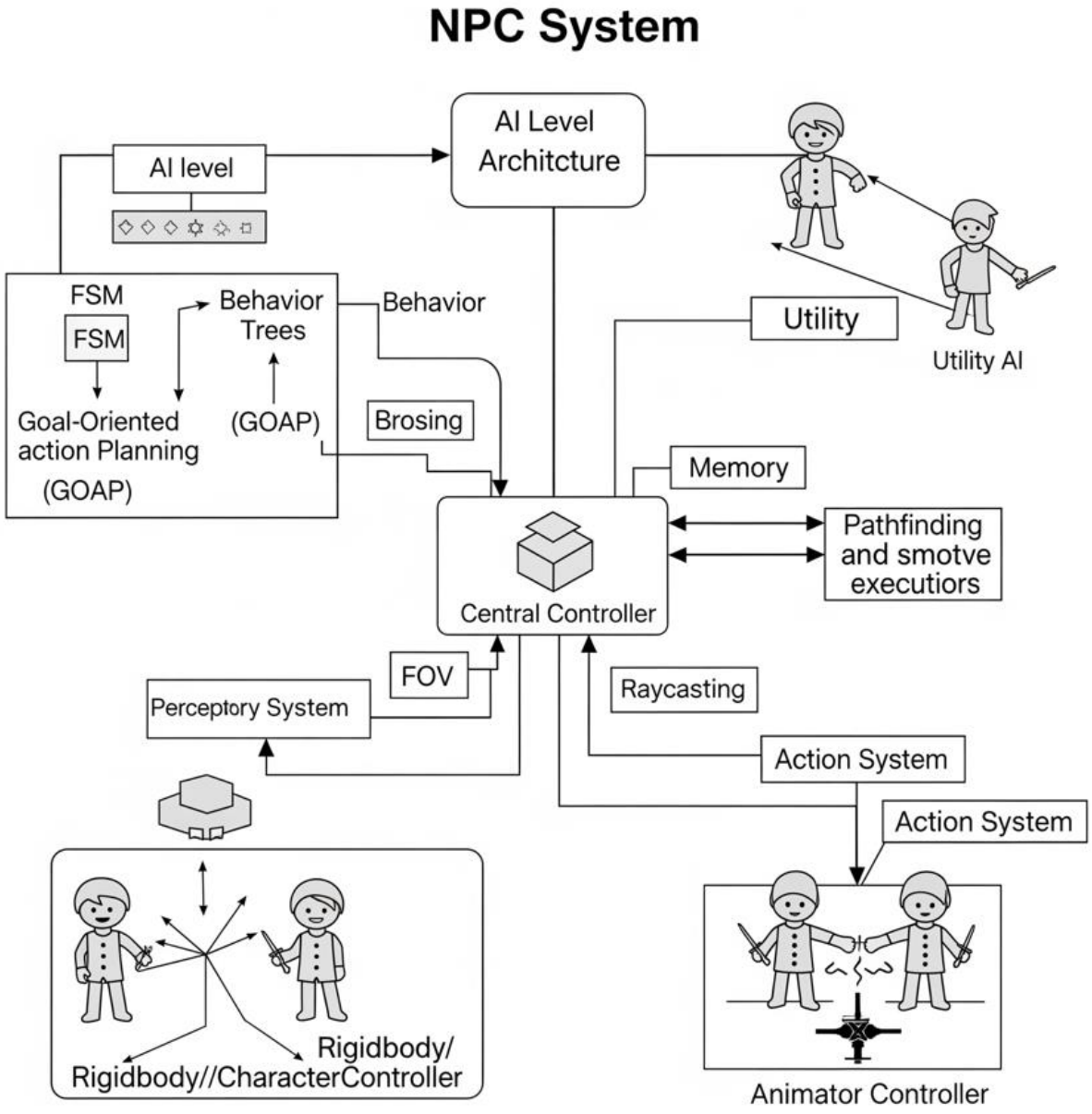


Рисунок 2.2 – Схема архітектури NPC

Контролер NPC – це централізований компонент, який координує всі модулі. Він обробляє поточний стан NPC, події ззовні (наприклад, «гравець побачений») і активує відповідну логіку.

Рівень AI (ШІ) може містити:

- FSM (Finite State Machine) – простий автомат станів (наприклад, патрулювання → атака → втеча);
- Behavior Tree – дерево поведінки, зручне для складної ієрархії дій;
- GOAP (Goal-Oriented Action Planning) – дозволяє NPC планувати послідовність дій для досягнення цілі;
- Utility AI – дозволяє вибирати дії на основі ваг/корисності.

Сенсорна система (Perception) - визначає, що бачить або чує NPC. Представляє собою модуль виявлення гравця, перешкод, звуків, подій. Вона зазвичай включає:

- поле зору (FOV);
- raycast до цілі;
- таймери «бачив/чув останнім часом».

Система пам’яті (Memory System) зберігає інформацію про минулі події. Дозволяє NPC ухвалювати обґрунтовані рішення на основі попереднього досвіду.

Виконання дій (Action System) обробляє таймінги, анімації, рух. Для навігації у просторі використовується алгоритм пошуку шляху – наприклад, A* або вбудована система Unity NavMesh, яка дозволяє NPC ефективно знаходити маршрут у складному середовищі з перешкодами. Ця система діє як виконавчий механізм, що точно реалізує заплановані дії (стрільба, патрулювання, хування, втеча тощо) у відповідному контексті гри.

Модуль Анімації та фізика забезпечує візуальну частину дій та відпоідає за зв’язок з Animator Controller (Unity), Rigidbody/CharacterController.

Архітектура NPC – це компроміс між гнучкістю, продуктивністю та складністю реалізації. Вибір підходу залежить від жанру гри, технічних обмежень і рівня деталізації, який хоче досягти розробник. Сучасні ігри (наприклад, *The Elder Scrolls*, *Red Dead Redemption 2*) використовують гібридні системи, де NPC мають складну поведінку, але з оптимізацією для продуктивності.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ NPC-ПЕРСОНАЖІВ

3.1 Розробка NPC на базі вибраної моделі

Для реалізації поведінки NPC обрана модель кінцевого автомата станів (FSM), інтегрована з аніматором Unity через клас StateMachineBehaviour. Такий підхід дозволяє розділити логіку поведінки на окремі стани, що відповідають різним режимам роботи NPC, та управляти переходами між ними на основі параметрів.

Створення базового класу FSM

Базовий клас NPCBaseFSM успадковується від StateMachineBehaviour і містить загальні змінні та методи, які використовуються в усіх станах NPC. У методі OnStateEnter відбувається ініціалізація посилань на об’єкт NPC та гравця (опонента), що дозволяє всім станам працювати з ними (рис. 3.1).

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
    0 references
5   public class NPCBaseFSM : StateMachineBehaviour
6   {
        2 references
7       public GameObject NPC;
        1 reference
8       public GameObject opponent;
        0 references
9       public float speed = 2.0f;
        0 references
10      public float rotSpeed = 1.0f;
        0 references
11      public float accuracy = 3.0f;
        0 references
12      public override void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
13      {
14          NPC = animator.gameObject;
15          opponent = NPC.GetComponent<CarAi>().GetPlayer();
16      }
17  }
```

Рисунок 3.1 – Базовий клас NPCBaseFSM

Реалізація конкретних станів

Для кожного стану поведінки створено окремий клас-нащадок базового FSM.

Клас Patrol організує те, що NPC послідовно пересувається між заданими точками (waypoints). У методі OnStateUpdate здійснюється рух до поточного пункту патрулювання, а при досягненні – вибір наступного (рис. 3.2).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
0 references
public class Patrol : NPCBaseFSM
{
    5 references
    GameObject[] waypoints;
    6 references
    int currentWP;
    0 references
    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        base.OnStateEnter(animator, stateInfo, layerIndex);
        waypoints = GameObject.FindGameObjectsWithTag("Waypoints");
        currentWP = 0;
    }
    0 references
    override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        if (waypoints.Length == 0) return;

        if (Vector3.Distance(waypoints[currentWP].transform.position,
                             NPC.transform.position) < accuracy)
        {
            currentWP++;
            if (currentWP >= waypoints.Length)
            {
                currentWP = 0;
            }
        }
        Vector3 direction = waypoints[currentWP].transform.position - NPC.transform.position;
        NPC.transform.rotation = Quaternion.Slerp(NPC.transform.rotation,
                                                  Quaternion.LookRotation(direction),
                                                  rotSpeed * Time.deltaTime);
        NPC.transform.Translate(0, 0, Time.deltaTime * speed);
    }
    0 references
    override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
    }
}
```

Рисунок 3.2 – Базовий клас Patrol

За рахунок класу Chase NPC переслідує гравця, повільно повертаючи голову у його напрямку та рухаючись вперед (рис. 3.3).

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

0 references
public class Chase : NPCBaseFSM
{
    0 references
    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        base.OnStateEnter(animator, stateInfo,layerIndex);
    }

    0 references
    override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        var direction = opponent.transform.position - NPC.transform.position;
        NPC.transform.rotation = Quaternion.Slerp(NPC.transform.rotation,
                                 Quaternion.LookRotation(direction),
                                 rotSpeed * Time.deltaTime);
        NPC.transform.Translate(0, 0, Time.deltaTime * speed);
    }

    0 references
    override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {

    }
}
```

Рисунок 3.3 – Базовий клас Chase

Для управління переходами між станами використовується параметр distance, який передається в Animator через скрипт CarAi (рис. 3.4).

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

0 references
public class CarAi : MonoBehaviour
{
    2 references
    Animator anim;
    2 references
    public GameObject player;
    0 references
    public GameObject GetPlayer()
    {
        return player;
    }

    0 references
    void Start()
    {
        anim = GetComponent<Animator>();
    }
    0 references
    void Update()
    {
        anim.SetFloat("distance", Vector3.Distance(transform.position,player.transform.position));
    }
}
```

Рисунок 3.4 – Базовий клас CarAi

Він обчислює відстань між NPC та гравцем щокадру і встановлює значення параметра. В Animator Controller налаштовані переходи між станами Patrol і Chase на основі значення distance (наприклад, перехід у стан Chase при відстані менше 20 одиниць).

Логіка руху NPC реалізована через зміну позиції та обертання об’єкта за допомогою методів Transform.Translate та Quaternion.Slerp. Це дозволяє плавно змінювати напрямок руху і швидкість NPC.

3.2 Реалізація сценаріїв поведінки

У проєкті реалізовано три основні сценарії поведінки NPC: патрулювання, переслідування гравця та повернення до патрулювання, які керуються за допомогою моделі кінцевого автомата (FSM), вбудованої у Animator Controller.

Патрулювання – у стані Patrol NPC пересувається між точками, які задані в сцені як об’єкти з тегом Waypoints. NPC переходить до наступної точки, коли наближається на задану відстань (accuracy). Рух NPC реалізується плавним обертанням у напрямку точки та поступовим переміщенням вперед.

Переслідування – якщо гравець наближається до NPC на відстань менше певного порогу (наприклад, 20 одиниць), в Animator активується перехід у стан Chase. У цьому стані NPC повертається в бік гравця і починає рух у його напрямку з заданою швидкістю.

Повернення до патрулювання – Коли гравець виходить за межі встановленої дистанції (наприклад, >20 одиниць), NPC автоматично повертається до стану Patrol. Перехід назад виконується через параметр distance, який задається в скрипті CarAi:

```
anim.SetFloat("distance", Vector3.Distance(transform.position,
player.transform.position));
```

На основі цього параметра в Animator встановлені умови переходу:

– distance < 20 → Patrol → Chase;

– distance > 20 → Chase → Patrol.

Таким чином, NPC динамічно змінює свою поведінку залежно від розташування гравця, реагуючи на наближення або віддалення.

3.3 Інтеграція NPC у прототип гри або симуляційне середовище

Для перевірки роботи моделі поведінки NPC була створена тестова сцена в середовищі Unity. У ній реалізовано взаємодію між керованим гравцем і NPC на основі FSM (Finite State Machine).

На сцені розміщено такі об’єкти (рис. 3.5):

– гравець – об’єкт, яким можна керувати за допомогою клавіш WASD або стрілочок;

– NPC – персонаж із прив’язаним Animator і реалізованими станами патрулювання та переслідування;

– точки маршруту (waypoints) – об’єкти з тегом "Waypoints", між якими NPC переміщується в режимі патрулювання.

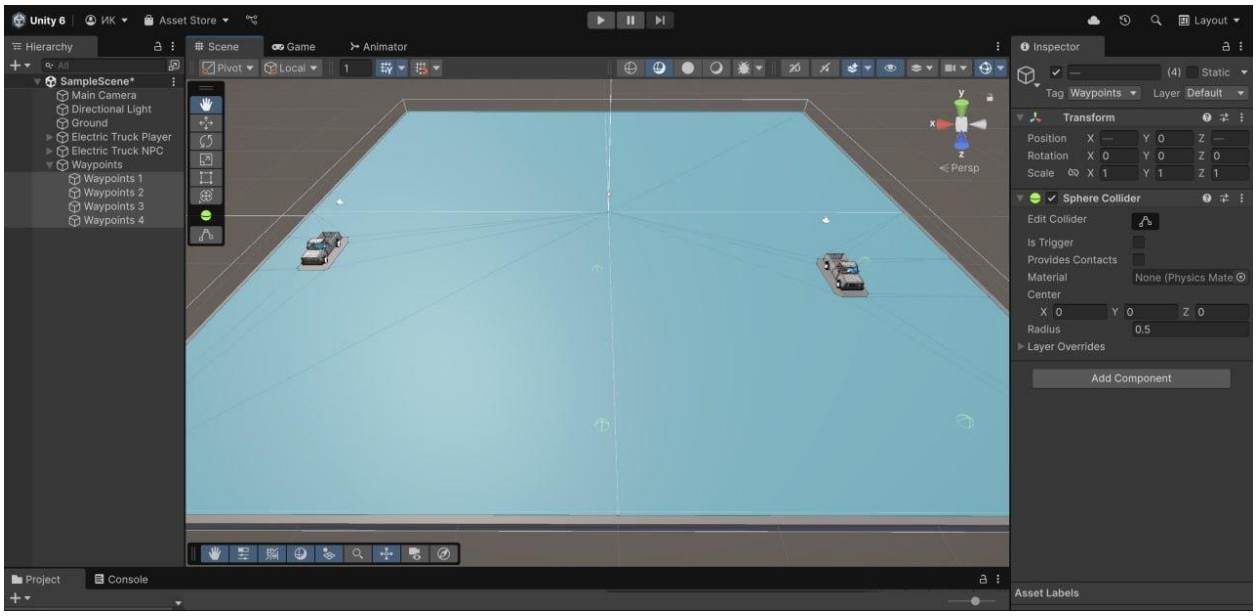


Рисунок 3.5 – Загальний вигляд сцени в Unity

Логіка переходу між станами реалізована в Animator за допомогою FSM. Визначено два основні стани (ри. 3.6):

– Patrol – NPC рухається між точками маршруту.
– Chase – NPC переслідує гравця, якщо відстань менша за певний поріг.

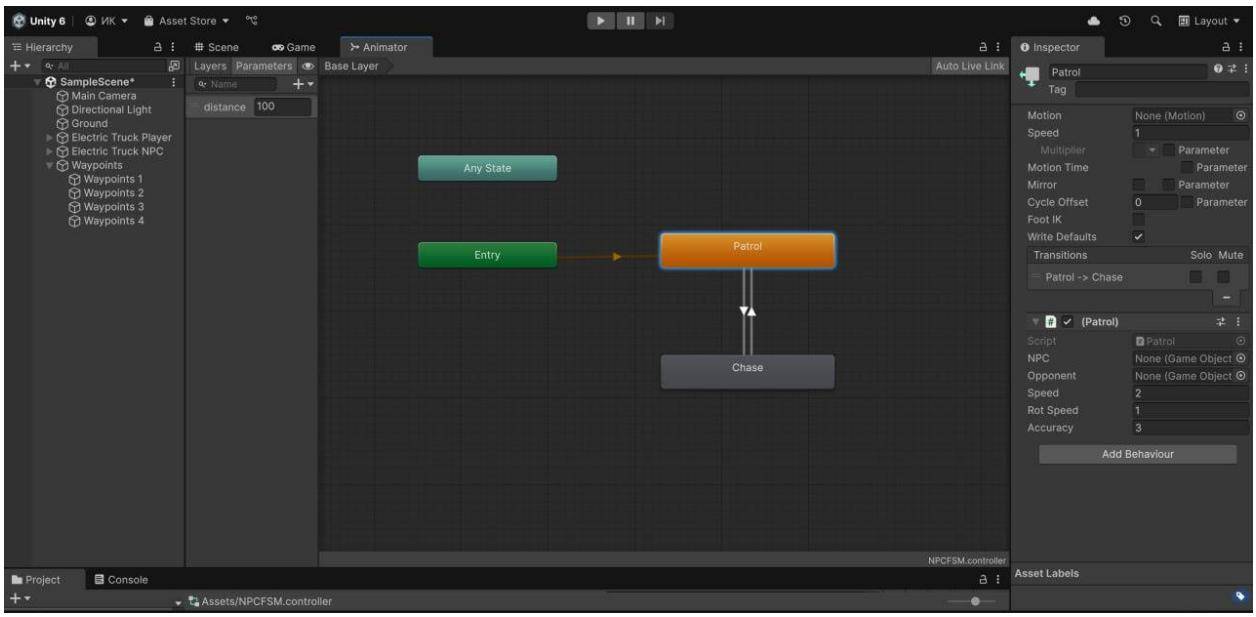


Рисунок 3.6 – Patrol і Chase, а також параметр distance

Параметр distance контролює, коли відбувається перехід між станами.

Поведінка NPC визначається через наслідувані стани FSM (рис. 3.7, 3.8):

– NPCBaseFSM.cs – базовий клас, у якому визначено доступ до NPC та гравця;
– Patrol.cs – NPC пересувається між точками маршруту;
– Chase.cs – NPC наближається до гравця при досягненні певної дистанції.

```
override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
{
    if (waypoints.Length == 0) return;

    if (Vector3.Distance(waypoints[currentWP].transform.position,
                         NPC.transform.position) < accuracy)
    {
        currentWP++;
        if (currentWP >= waypoints.Length)
        {
            currentWP = 0;
        }
    }
    Vector3 direction = waypoints[currentWP].transform.position - NPC.transform.position;
    NPC.transform.rotation = Quaternion.Slerp(NPC.transform.rotation,
                                              Quaternion.LookRotation(direction),
                                              rotSpeed * Time.deltaTime);
    NPC.transform.Translate(0, 0, Time.deltaTime * speed);
}
```

Рисунок 3.7 – OnStateUpdate у Patrol.cs

```
override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
{
    var direction = opponent.transform.position - NPC.transform.position;
    NPC.transform.rotation = Quaternion.Slerp(NPC.transform.rotation,
                                              Quaternion.LookRotation(direction),
                                              rotSpeed * Time.deltaTime);
    NPC.transform.Translate(0, 0, Time.deltaTime * speed);
}
```

Рисунок 3.8 – OnStateUpdate у Chase.cs

У процесі симуляції:

– NPC патрулює за маршрутними точками;
– якщо гравець наближається – NPC переслідує його;
– якщо гравець віддаляється – NPC повертається до патрулювання (рис.3.9).

3.4 Тестування системи NPC: сценарії, результати, проблеми

Для перевірки коректності роботи реалізованої FSM-системи NPC було проведено тестування за сценаріями, представленими в таблиці 3.1.

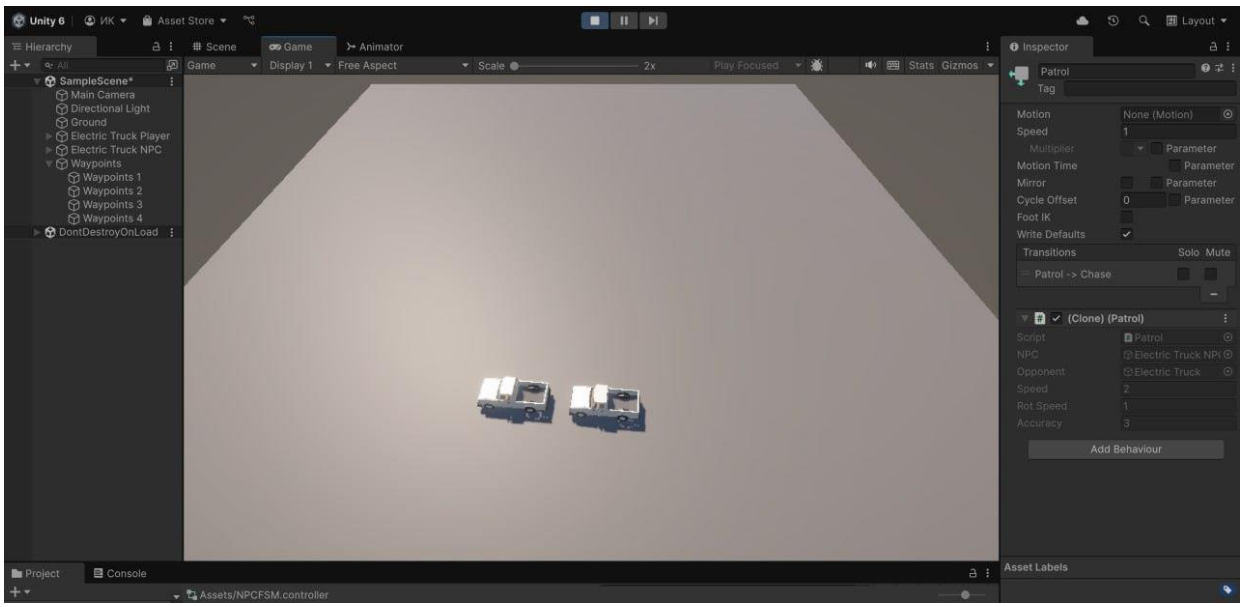


Рисунок 3.9 – Переслідування

Таблиця 3.1 – Тестування

№	**Сценарій**	**Очікувана поведінка NPC**
1	Гравець відсутній поруч	NPC патрулює між точками Waypoint
2	Гравець наближається на відстань < 10 одиниць	NPC переходить у стан переслідування
3	Гравець віддаляється > 20 одиниць	NPC повертається до патрулювання
4	Гравець наближається й віддаляється кілька разів	NPC коректно перемикається між станами

В ході тестування було встановлено:

– NPC коректно починає з патрулювання, рухаючись між заданими точками;

– при наближенні гравця (менше 20 одиниць) – плавно переходить у стан переслідування;

- після втрати контакту (дистанція > 20 одиниць) – повертається до попереднього маршруту патрулювання;
- анімаційні стани в Animator’і перемикаються згідно з логічними параметрами (distance);
- FSM реалізована через StateMachineBehaviour, кожен стан реалізує власну логіку.

Під час тестування були виявлені такі труднощі, представлені в таблиці 3.2.

Таблиця 3.2 – Проблеми, які виявлені під час тестування

Проблема	**Опис**	**Рішення**
NPC не переходив у стан chase	Значення distance не оновлювалося	Додано оновлення параметра distance у CarAi.Update()
NPC обертався занадто повільно	Затримка в обертанні у стані chase	Підвищено rotSpeed до 2.5
NPC залишався на останній точці патруля	Потрапляв у нескінченний цикл	Додано перевірку на довжину маршруту та обнулення currentWP

FSM-система NPC показала стабільну та передбачувану поведінку. Завдяки параметричному контролю (Animator.SetFloat("distance")), NPC ефективно реагує на зміну положення гравця. Такий підхід дозволяє масштабувати поведінку NPC і додавати нові стани без значних змін у загальній архітектурі.

3.5 Аналіз ефективності реалізованих рішень

У процесі реалізації NPC на основі скінченного автомата станів (Finite State Machine, FSM) було досягнуто балансу між простотою реалізації та задовільною адаптивною поведінкою персонажа в ігровому середовищі.

Переваги реалізованого підходу:

- модульність. Кожен стан (патрулювання, переслідування) винесений в окремий скрипт, що дозволяє легко масштабувати логіку, додаючи нові стани;
- зрозуміла логіка переходів. Використання параметра distance в Animator дозволяє явно контролювати умови переходів між станами;
- простота налагодження. Завдяки візуальній системі Animator в Unity, зручно відслідковувати активний стан NPC у реальному часі;
- оптимальність. Використання FSM не потребує великої кількості ресурсів, що є важливим для мобільних або інді-ігор.

Обмеження:

- жорстка структура. FSM може бути складною для підтримки при збільшенні кількості станів і переходів (ефект "станового вибуху");
- немає адаптації. NPC не вчиться, не прогнозує дії гравця, а лише реагує за заздалегідь визначеними правилами;
- лінійна поведінка. NPC діє однаково у кожному сеансі, що може знижувати реалістичність при довготривалій грі.

Для поставлених задач (демонстрація базових сценаріїв поведінки NPC: патрулювання, переслідування, повернення) реалізоване рішення є ефективним. Воно забезпечує стабільну та передбачувану поведінку персонажа без перевантаження системи чи складної логіки.

ВИСНОВКИ

Метою даної роботи було реалізувати модель поведінки NPC із застосуванням алгоритмів штучного інтелекту для підвищення реалістичності, адаптивності та варіативності взаємодії з гравцем у відеоіграх.

Для моделювання поведінки NPC було обрано метод кінцевих автоматів (FSM - Finite State Machine), який дозволяє структуровано та ефективно організувати логіку поведінки персонажа. Цей метод є поширеним у геймдизайні для реалізації станів персонажа та переходів між ними на основі умов або подій.

Робота була реалізована в середовищі Unity, популярному ігровому рушії, який підтримує створення 2D- та 3D-ігор і використовує C# як основну мову програмування. Реалізація інтегрована з Animator Unity через клас StateMachineBehaviour, що дозволяє пов’язати логіку FSM з анімаційними станами.

Було реалізовано три основні сценарії поведінки NPC: патрулювання між заданими точками, переслідування гравця при його наближенні та повернення до патрулювання, якщо гравець віддаляється. Переходи між станами "Патрулювання" та "Переслідування" контролюються параметром "distance", який обчислює відстань до гравця та передається в Animator.

Результати Тестування: Тестування системи NPC показало її стабільну та передбачувану поведінку. NPC коректно починає з патрулювання, переходить у стан переслідування при наближенні гравця та повертається до патрулювання при його віддаленні. Виявлені та виправлені проблеми стосувалися оновлення параметра відстані та швидкості обертання NPC.

Реалізований на основі FSM підхід є ефективним для поставлених завдань завдяки своїй модульності, зрозумілій логіці переходів та простоті налагодження, а також є оптимальним з точки зору ресурсів. Однак, FSM має

обмеження, такі як жорстка структура, відсутність адаптації та лінійність поведінки при великій кількості станів.

Таким чином, у рамках кваліфікаційної роботи було успішно розроблено та реалізовано модель поведінки NPC з використанням методу кінцевих автоматів у середовищі Unity, що дозволило створити базову, але надійну та передбачувану поведінку персонажа, яка ефективно реагує на положення гравця

ПЕРЕЛІК ПОСИЛАНЬ

1. Bailey C., Katchabaw M. An emergent framework for realistic psychosocial behaviour in non player characters. *Proceedings of the 2008 Conference on Future Play : Research, Play, Share*. New York : Association for Computing Machinery, 2008. P. 17–24. URL : https://doi.org/10.1145/1496984.1496988 (дата звернення : 05.05.25).

2. Picard R. W. Affective computing. Cambridge (MA): MIT Press, 1997.

3. Pfau J., Smeddinck J. D., Malaka R. The case for usable AI: What industry professionals make of academic AI in video games. *Extended Abstracts of the 2020 Annual Symposium on Computer-Human Interaction in Play*. New York : Association for Computing Machinery, 2020. P. 330–334. URL : https://doi.org/10.1145/3383668.3419905 (дата звернення : 05.05.25).

4. Taralla D., Qiu Z., Sutera A., Fonteneau R., Ernst D. Decision making from confidence measurement on the reward growth using supervised learning: A study intended for large-scale video games. *Proceedings of the 8th International Conference on Agents and Artificial Intelligence*. Rome : SciTePress, 2016. Vol. 2. P. 264–271.

5. Барановська Т. В., Кузьменко О. В. Основи штучного інтелекту: навчальний посібник. Київ : Центр учбової літератури, 2020. 244 с.

6. Паламарчук В. Ю., Лозовий В. С. Інтелектуальні агенти в комп'ютерних іграх: структура, поведінка, архітектура. *Вісник Хмельницького національного університету. Серія: Технічні науки*. 2021. № 1. С. 120–124. URL : http://journals.khnu.km.ua/vestnik/pdf/tech/2021_1_24.pdf (дата звернення : 28.05.2025).

7. Champandard A. J. AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors. New Riders, 2003. 496 p.

8. Unity Technologies. Unity Manual : Animation State Machines. Unity Documentation. URL : https://docs.unity3d.com/Manual/AnimationState Machines.html (дата звернення : 28.05.2025).

9. Unity Technologies. Unity Manual : Navigation and Pathfinding. Unity Documentation. URL : https://docs.unity3d.com/Manual/nav-Overview.html (дата звернення : 28.05.2025).