Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Модель штучної імунної мережі для керування поведінкою ботів у RTS-іграх

(тема)

Виконав:
студент II курсу, групи КСМм-23-1
Жукова А.С.
(прізвище, ініціали)

Спеціальність
123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

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

Керівник: ст. викл. Фомічов О.О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерні системи та мережі
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Жуковій Анастасії Сергіївні
(прізвище, ім’я, по батькові)

1. Тема роботи Модель штучної імунної мережі для керування поведінкою ботів у RTS-іграх

затверджена наказом по університету від “ 22 ” листопада 2024 р. № 1237 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 20 січня 2025 р.

3. Вхідні дані до роботи
   1) документація Unity;
   2) мережа aiNET;
   3) графічні ресурси;
   4) звукові ресурси;
   5) документація C#;
   6) документація Microsoft Visual Studio.

4. Перелік питань, що потрібно опрацювати у роботі
   1) аналіз предметної області;
   2) огляд використаних технологій та ігровий дизайн;
   3) програмна реалізація;
   4) інструкція користувача;
   5) висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Презентація – 18 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області	26.11.24-16.12.24	
2	Вибір технологій розробки та програмного	17.12.24-18.12.24	
	забезпечення		
4	Розробка ігрових сцен	19.12.24-23.12.24	
5	Розробка aiNET	23.12.24-25.12.24	
6	Розробка MVC	26.12.24-28.12.24	
7	Офрмлення матеріалів кваліфікаційної роботи	30.12.24-05.12.24	
8	Подання кваліфікаційної роботи керівникові	15.01.24-16.01.24	
9	Подання кваліфікаційної роботи на рецензу-	19.01.24.-20.01.25	
	вання		

Дата видачі завдання 25 листопада 2024 р.

Студент _____
(підпис)

Керівник роботи _____ ст.викл. Фомічов О.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 85 с., 27 рис., 1 табл., 1 дод., 31 джерело.

C#, UNITY, КЛАСИФІКАЦІЯ, ШТУЧНА ІМУННА МЕРЕЖА АФІННІСТЬ, aiNeT, MVC, КЛОНУВАННЯ, МУТАЦІЯ, СУПРЕСІЯ, ІГРОВИЙ ПЕРСОНАЖ.

Метою кваліфікаційної роботи є розробка, реалізація та випробування імунного методу aiNET, керування поведінкою ігрових персонажів у комп'ютерній грі. Розроблений додаток має жанр RTS, керування за допомогою клавіш, ворогів, вежі для окупації та команду гравціф, яких потрібно купляти за монети.

У ході виконання кваліфікаційної роботи проаналізовано різні жанри мобільних ігор, детально розглянуто найпопулярніші середовища для гейм-розробки. Розглянуто та вибрано мову програмування.

Досліджено та виконано архітектуру MVC. Реалізація патерна MVC зроблена в скрипті «DragHandler», який відповідає за драг-операції елементів гри.

Для досягнення мети проєкту використовується середовище розробки Microsoft Visual Studio 2019 та ігровий рушій Unity 3D 2020. Написання скриптів виконувалось мовою програмування C#.

ABSTRACT

Master’s thesis: 85 pages, 27 figures, 1 table, 1 appendix, 31 sources.

C#, UNITY, CLASSIFICATION, ARTIFICIAL IMMUNE NETWORK AFFINITY, aiNeT, MVC, CLONING, MUTATION, SUPRESSION, GAME CHARACTER.

The purpose of the qualification work is to develop, implement and test the aiNET immune method for controlling the behavior of game characters in a computer game. The developed application has an RTS genre, control with keys, enemies, towers for occupation, and a team of players to be bought for coins.

In the course of the qualification work, various genres of mobile games were analyzed, the most popular environments for game development were considered in detail. A programming language was considered and selected.

The MVC architecture was researched and implemented. The MVC pattern is implemented in the “DragHandler” script, which is responsible for drag operations of game elements.

To achieve the project goal, we used the Microsoft Visual Studio 2019 development environment and the Unity 3D 2020 game engine. The scripts were written in the C# programming language.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ІА – імунний алгоритм

ІМ – імунна модель

МУТАЦІЯ – випадкові зміни в деякому антитілі; у штучній імунній системі відповідає випадковим змінам у можливому рішенні.

ПЗ – програмне забезпечення

ШІС – штучна імунна система

aiNET – Artificial Immune Network

MVC – архітектура модель-вид-контроль (англ., Model-View-Controller)

RTS – стратегія реального часу (англ., Real-time Strategy)

TPS – шутери від третьої особи

ВСТУП

У сучасному суспільстві відеоігри відіграють важливу роль у культурі, економіці та технологіях. Вони є не лише розвагою, а й інструментом для навчання нових навичок, розвитку креативності та вирішення складних технічних завдань. Одним з найперспективніших напрямків розробки відеоігор є використання штучного інтелекту (ШІ), особливо моделей, натхненних природними процесами, такими як штучні імунні системи. Ці системи засновані на біологічних принципах імунної відповіді і мають високий потенціал у задачах адаптації, класифікації та оптимізації, що робить їх придатними для управління поведінкою ботів у стратегіях реального часу (RTS). Метою цього дослідження є побудова моделі штучної імунної мережі, яка контролює поведінку ігрових ботів і покращує їхню адаптивність та реалістичність.

Стрімкий розвиток інформаційних технологій створює нові можливості для розробників і видавців ігрового ПЗ, а також спрощує і вдосконалює наявні технології. У результаті в сфері розроблення ігрових додатків з'явилося безліч платформ, шаблонів для розроблення архітектур ігрових додатків та ігрових рушіїв. Крім того, дедалі більшої популярності набувають системи штучного інтелекту, здатні автоматизувати розв'язання складних практичних завдань, пов'язаних з обробленням та аналізом інтелектуальних даних. Сьогодні штучний інтелект можна використовувати як повноцінних інтерв'юерів, викладачів, юристів та адвокатів. Розвиток систем розробки штучного інтелекту справив значний вплив і на ринок ігрового програмного забезпечення.

У цій роботі проаналізовано існуючі методи та алгоритми, модифіковано імунний оператор для реалізації адаптивного управління та досліджено архітектуру ігрового додатку, що реалізує розроблену модель.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд ігрових жанрів

Жанр відеоігор використовується для класифікації відеоігор відповідно до інтерактивного ігрового процесу гравця. На відміну від літературних чи кінематографічних творів, жанри відеоігор не залежать від сценарію чи змісту вигаданого ігрового світу. Оскільки не існує єдиної жанрової класифікації відеоігор, одна й та сама відеогра може бути класифікована по-різному залежно від джерела інформації. Також можливе змішування жанрів, якщо певну гру не можна віднести до одного жанру. [1]

Одну з перших детальних класифікацій відеоігор зробив у 1982 році геймдизайнер Кріс Кроуфорд у своїй книзі «Мистецтво дизайну комп'ютерних ігор»; у 1986 році Ральф Коен вказав на те, що неможливо дати чітке визначення жанру. Він заявив, що «жанр – це відкрита категорія, і будь-хто може модифікувати її, додаючи, змінюючи або модифікуючи її компоненти».

Розглянемо найбільш поширені жанри:

1.1.1 Шутери від першої або третьої особи (FPS/ TPS)

Шутер від першого лиця (FPS) – це популярний жанр ігор, в якому геймер бачe весь світ очима свого персонажа, який на болі бою використовує вогнепальну зброю. Тобто, гравець бачить руки, зброю та інші предмети свого персонажа, що дозволяє глибше зануритись у світ гри. В цьому жанрі гравці повинні використовувати свої навички прицілювання, щоб стріляти у ворогів і виконувати місії в ігровому світі. FPS-ігри можуть бути однокористувацькими, де гравці протистоять штучному інтелекту, або багатокористувацькими, де гравці грають один проти одного в Інтернеті.

Жанр FPS зазвичай включає елементи швидкості, рефлексів, стратегії та командної гри. Гравці використовують різноманітну зброю, гранати, прийоми та навички, щоб перемогти ворогів і досягти мети.

Однією з найвідоміших відеоігор жанру FPS є Counter-Strike.

Рисунок 1.1 – фрагмент гри «Counter-Strike»

Шутери від третьої особи (TPS: Third-Person Shooter) – вирізняється унікальним поєднанням динамічного бою та захопливого візуального ряду. На відміну від шутерів від першої особи, які є позиційованими, гравець бачить свого персонажа ззаду або трохи збоку. TPS зосереджені на тактичному та статегічному захисті. Більшість ігор використовує механізм «укриття» – гравці можуть ховатися за об’єктами, стріляти ззаду або перестрибувати через перешкоди. Зовнішній вид персонажів дозволяє розробникам приділяти більше уваги на візуальні деталі, анімацію, костюми та інше.

GTA: San Andreas – гра з відкритим світом, розроблений Rockstar North і випущений Rockstar Games у 2004 році. Це одна з найвідоміших і найпопулярніших частин серії Grand Theft Auto (GTA). Пропонуючи

величезний відкритий світ і надаючи гравцям свободу пересування, гра стала справжньою класикою і вплинула на багато наступних ігор. [2]

Рисунок 1.2 – фрагмент гри «GTA: San Andreas»

1.1.2 Рольові ігри (RPG)

Рольові ігри набули популярності за останні кілька десятиліть, еволюціонуючи від нішевих настільних ігор, таких як Dungeons & Dragons, до масових розваг, таких як багатокористувацькі онлайн-ігри. У рольових іграх гравці беруть на себе роль вигаданого персонажа і вирушають у пригоди, зумовлені сюжетом.

Однією з найбільших переваг рольових ігор є те, що вони дозволяють гравцям робити вибір, який впливає на їхній світ. Незалежно від того, чи це моральна дилема, чи В іграх можна побачити відображення реальних проблем і моральних дилем. Тут ти відчуваєш себе сильним, а баланс складності такий, що гра не є неймовірно складною, але й не буває легкою.

Гра «Gothic» стала культовою завдяки своєму унікальному світу, реалістичній атмосфері та глибокій сюжетній лінії, що переносить гравців у

похмурий фентезійний світ, сповнений небезпек та політичних інтриг. Дія гри відбувається в тюремній колонії, ізольованій магічним бар'єром, де панує безжальна ієрархія та постійна боротьба за виживання. [3]

Рисунок 1.3 – фрагмент гри «Gothic»

1.1.3 Мобільні «казуальні» ігри

Казуальні ігри приваблють гравців своєю простотою і швидким ігровим процесом, який не вимагає витрат часу або складних навичок. Ідеально підходять для коротких періодів відпочинку. Ігри розроблені так, що не потрібно запам’ятовувати складні комбінації. Керування часто обмежується дотиком, свайпом або клацанням. Наприклад, у Flappy Bird гравці просто натискають на екран, щоб змусити пташку полетіти. Багато ігор зв'язані з соціальними мережами, що дозволяє гравцям змагатися з іншими, ділитися досягненями та результатими.

Flappy Bird – проста, але надзвичайно популярна мобільна гра. Вона швидко завоювала величезну популярність завдяки своїй захоплюючій складній механіці та мінімалістичному підходу до дизайну. Незважаючи на те, що вона здається примітивною за своєю концепцією, вона стала

справжнім феноменом в індустрії мобільних ігор. Гравці торкаються екрану, щоб підняти пташку, а якщо не вдається, то вона падає під дією сили тяжіння. Мета – пролетіти між зеленими трубами різної висоти, не зачепивши їх. [4]

Рисунок 1.4 – фрагмент гри «Flappy Bird»

1.1.4 Платформери

Однією з класичних комп'ютерних ігор є платформер, який зародився в 1980-х і залишається популярним і сьогодні. Основна увага приділяється переміщенню персонажа різними рівнями за допомогою стрибків, уникаючи перешкод і ворогів. Часто атмосфера гри стилізується під мультфільм або ретро-вигляд, який робиться за допомогою піксельною графіки. Платформні ігри мають різноманітні пастки, такі як шипи, щілини та рухомі риштування, які вимагають від гравців ретельного розрахунку своїх рухів. Проходження рівня зазвичай приносить нагороди, такі як додаткові життя або нові здібності персонажа. Багато платформерів використовують піксельну графіку, однак новіші платформери часто використовують високоякісну анімацію та 3D-візуалізацію.

Один з найвпливовіших платформерів, на якому побудований жанр «Super Mario Bros». У Super Mario Bros гравець керує Маріо та його братом Луїджі в режимі для двох гравців. Мета гри – пройти серію рівнів і врятувати принцесу Персик від злого Баузера. [5]

Рисунок 1.5 – фрагмент гри «Super Mario Bros»

1.1.5 Стратегії (RTS)

Стратегія в реальному часі (RTS) – це жанр стратегічних відеоігор, в яких, на відміну від покрокових стратегій, ігровий процес не розділений на нормалізовані ходи і всі гравці можуть рухатися одночасно. RTS-ігри сприяли появі багатьох інших жанрів і стали основною силою, що спричинила появу кіберспорту в світі, вплинувши на кіберспорт в усьому світі. Поширеною помилкою є думка, що гра може називатися RTS лише тоді, коли вона включає такі елементи, як будівництво бази та збір ресурсів. RTS часто мають просунуті багатокористувацькі компоненти, які дозволяють гравцям змагатися з іншими гравцями та вдосконалювати свої навички.

Warcraft III: шедевр свого жанру, що поєднує сильний сюжет, добре продуманий мультиплеєр та інноваційні для свого часу механіки. Ігри всесвіту зосереджені на конфлікті між Альянсом та Ордою, Вогняним Легіоном, Батогом та іншими силами, що загрожують Азероту. Завдання гравця - керувати економікою та вести війну проти супротивників. [6]

Рисунок 1.6 – фрагмент гри «Warcraft III»

1.2 Огляд ігрових двигунів

Ігровий рушій – це готова архітектура, яку розробники використовують для запуску своїх ігор. Вони забезпечують основу для розробки, поєднуючи інструменти, пов'язані з графікою, фізикою, анімацією, звуком, штучним інтелектом та різними іншими аспектами гри. Ігрові рушії визначають можливості та функціональність гри, впливаючи на її якість, продуктивність, крос-платформну сумісність і навіть на бюджет розробки. [7]

1.2.1 Unreal Engine

Потужний ігровий рушій, розроблений компанією Epic Games у 1998 році. Спочатку розроблений для шутера від першої особи Unreal, рушій відтоді став універсальним інструментом для розробки ігор різних жанрів, зокрема шутерів, рольових ігор, платформерів, симуляторів і навіть додатків віртуальної та доповненої реальності. Unreal Engine відомий своїми потужними можливостями фотореалістичного рендерингу, включаючи високоякісні текстури, реалістичні відображення і тіні, а також підтримку освітлення в реальному часі.

Unreal Engine включає потужні інструменти для створення штучного інтелекту, такі як система AI Behaviour Trees, яка дозволяє налаштовувати поведінку неігрових персонажів (NPC) у складних ігрових сценаріях. Вона також підтримує навігацію та пошук шляхів, щоб допомогти NPC переміщатися в просторі та уникати перешкод. Доступний для різних платформ, включаючи ПК, консолі (PS, Xbox), мобільні пристрої (iOS, Android) і навіть веб-браузери.

Unreal Engine має нову візуальну мову сценаріїв, але традиційне програмування все ще може знадобитися. Epic Games також випустила вихідний код для цього рушія. Цей код написаний на C++ і його можна безкоштовно завантажити з Github. Це дозволяє користувачам вчитися на цьому коді, редагувати його та розробляти власні інструменти. Таким чином, користувачі мають повний контроль над рушієм, над яким вони працюють. Небагато компаній роблять свої файли доступними для громадськості для запуску власних продуктів. Це свідчить про те, що Epic Games хоче зробити всі інструменти, необхідні розробникам для створення того, що вони хочуть, доступними для них без обмежень. [8]

Рисунок 1.7 – фрагмент з рушія «Unreal Engine»

1.2.2 Amazon Lumberyard

Lumberyard – ще один варіант для тих, хто хоче створювати власні оригінальні та креативні відеоігри без особливих витрат, і компанія Amazon створила графічне програмне забезпечення, яке підтримує PlayStation 4, Xbox One і популярний сервіс Twitch.tv. Нещодавно Amazon придбала права на використання переваг дуже потужного та якісного рушія Cry Engine. На його основі було розроблено дещо менший додаток під назвою Lumberyard, який щойно вийшов. Це повністю безкоштовне програмне забезпечення для людей, які люблять програмувати і створювати власні відеоігри. Amazon вже підписав партнерські угоди з Sony та Microsoft, щоб полегшити випуск завершених проектів. Варто також зазначити, що програма повністю підтримує Twitch та хмарні додатки. Рушій активно розвивається Open 3D Foundation (спільнота з відкритим вихідним кодом), що робить його більш адаптивним і гнучким у налаштуванні, особливо для інді-студій і великих компаній, які потребують високого ступеня кастомізації.

Lumberyard можна безкоштовно завантажити та використовувати; Amazon не стягує роялті з доходів від гри. Однак, якщо гра використовує

сервіси AWS, розробники платять лише за використані хмарні ресурси. Безшовна інтеграція рушія з AWS робить його привабливим варіантом для розробників багатокористувацьких та онлайн-ігор.

Через свою складну архітектуру та можливості, цей рушій є складним для вивчення, особливо для початківців. У порівнянні з Unity та Unreal Engine, Lumberyard має меншу базу користувачів та менш розвинену спільноту, що ускладнює пошук документації та навчальних посібників. Незважаючи на свої потужні можливості, Lumberyard не часто використовується великими студіями. Однак рушій використовується в експериментальних та комерційних проектах, таких як «Star Citizen». [9]

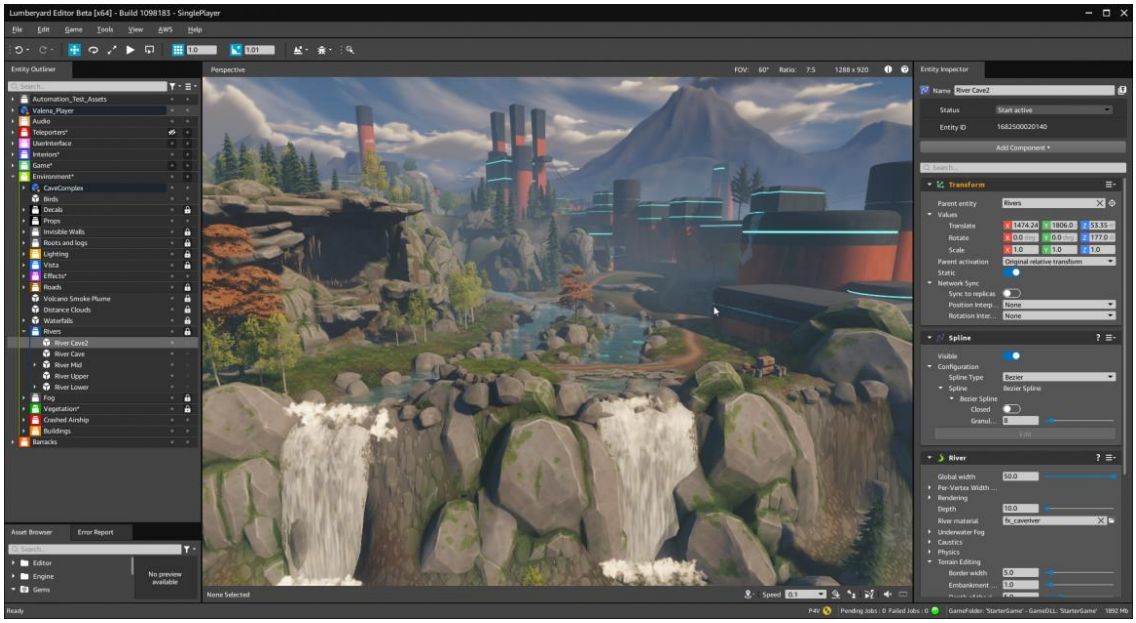

Рисунок 1.7 – фрагмент з рушія «Amazon Lumberyard»

1.2.3 LibGDX

LibGDX – це фреймворк з відкритим вихідним кодом на основі Java, що підтримує останню версію OpenGL. Додатки можна створювати для різних платформ, використовуючи ту саму кодову базу, а завдяки простоті використання, високій продуктивності, низькому споживанню ресурсів і

дуже ретельній документації LibGDX був прийнятий багатьма проектами, як аматорськими, так і професійними.

Наразі підтримуються Windows, Linux, Mac OS X, Android, iOS та HTML5. Фреймворк використовується багатьма незалежними розробниками ігор, а також великими компаніями (наприклад, гра Ingress, розроблена Google).

Документація LibGDX доступна, але може бути менш повною та актуальною, ніж у інших популярних фреймворків, таких як Unreal Engine та Unity.

Рисунок 1.8 – фрагмент з рушія «LibGDX»

1.2.4 Unity

Unity – один з найпопулярніших у світі рушіїв для розробки ігор, який використовують як аматори, так і професійні AAA-студії та кінематографісти. Хоча Unity вважається насамперед 3D-інструментом, він має багато спеціальних можливостей, включаючи підтримку 2D-ігор, віртуальної реальності, інструментів постпродакшину та крос-платформних додатків. Хоча розробники цінують Unity за інтерфейс та вбудовані функції,

унікальним середовищем Unity робить можливість писати власні скрипти C# для реалізації поведінки та керування ігровою механікою.

Рушій Unity важливий завдяки своїм крос-платформним можливостям, які гарантують можливість створювати та розвивати ігри незалежно від використовуваної операційної системи. На кожній платформі Unity пропонує можливість створювати повноцінні ігри, що характеризуються захоплюючою графікою, повною функціональністю та впровадженням професійного штучного інтелекту для побічних персонажів. Залежно від ваших потреб, ви можете обмежити розробку гри функціями, доступними в середовищі Unity, або скористатися магазином ресурсів, де доступні тисячі компонентів, які допоможуть вам у розробці гри. [10]

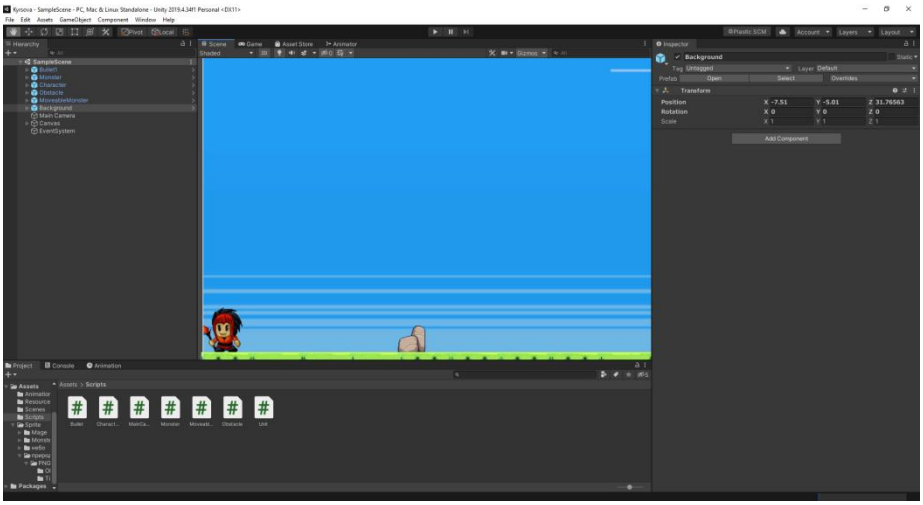

Рисунок 1.9 – Інтерфейс рушія Uhity 2019

1.3 Вибір мови та засобів розробки

Сьогодні різноманітність мов програмування, фреймворків та рушіїв, доступних для розробки ігор, надає розробникам різні можливості для створення цікавих та захоплюючих проектів. Одним з найпопулярніших рішень у сучасній ігровій індустрії є використання комбінації мови C# та ігрового рушія Unity. Ця комбінація є однією з найзручніших для

розробників, незалежно від досвіду, оскільки пропонує зрозумілий синтаксис, гнучкість та велику екосистему інструментів та ресурсів.

Розглянувши таблицю 1.1 можна зробити такі висновки:

у випадку інтерпретованої мови створення та виконання коду – це двоетапний процес: програміст створює код та щоразу, коли гравець грає в гру, код перетворюється в реальному часі на комп'ютері гравця з авторської мови на машинну. Перевага такого рішення полягає в тому, що програма може бути інтерпретована відповідно до типу комп'ютера, на якому вона працює, що дозволяє переносити код. Наприклад, JavaScript-код веб-сторінки, незалежно від того, чи працює вона під управлінням macOS, Windows, Linux або однієї з багатьох мобільних операційних систем, таких як iOS, Android або Windows Phone, може працювати майже на будь-якому сучасному комп'ютерах, на яких він працює. Однак така універсальність означає, що код потребує часу для інтерпретації на комп'ютері гравця, що уповільнює виконання коду і означає, що мова програмування не оптимізована для пристрою, на якому запускається додаток. Оскільки на всіх пристроях виконується один і той самий інтерпретований код, стає неможливим оптимізувати його для конкретної системи, на якій він виконується.

У скомпільованій мові C# процес програмування поділяється на три етапи: програміст створює код мовою програмування C# далі компілятор перетворює код з авторської мови в скомпільований додаток, який потім перетворюється в додаток на машинній мові для конкретного типу машини. Комп'ютер запускає скомпільовану програму. Цей додатковий процес компіляції перетворює код з вихідної мови на щось придатне для використання (тобто програму), яка може виконуватися безпосередньо комп'ютером без використання інтерпретатора. Компілятор має повне розуміння програми та платформи виконавця, на якій вона працює, тому ви можете включити багато оптимізацій у свій процес. [11]

Таблиця 1.1 – Мови програмування

<table>
<tr><td colspan="3">Мови програмування</td></tr>
<tr><td>Мови придатні для читання комп’ютером</td><td colspan="2">Мови придатні для читання програмістом</td></tr>
<tr><td rowspan="2">Машинна мова</td><td>Компільовані</td><td>Інтерпретовані</td></tr>
<tr><td>BASIC, C++, C#, Java</td><td>JavaScript, PHP, Python</td></tr>
</table>

C# – це найновіша об'єктно-орієнтована мова високого рівня, розроблена Microsoft у період 1998-2001 років. З точки зору синтаксису, його часто порівнюють з такими мовами, як Object Pascal, C++ та Java. У середовищі розробки він вважається простим, звичним і зрозумілим. C# тісно пов'язаний з платформою. Net є для нього одночасно фреймворком і середовищем виконання. Довгий час ця залежність вважалася найбільшим недоліком мови, оскільки обмежувала її використання лише в системах Windows. Microsoft вирішила цю проблему в 2016 році. . Net Core сумісний з іншими операційними системами. З тих пір C# використовується для створення додатків для всіх операційних систем.

C# можна охарактеризувати як універсальну і всюдисущу мову. Мова використовується для створення мобільних додатків для Android, iOS і Windows 10 Mobile, а також великих систем, адаптованих до потреб підприємств За допомогою фреймворку ASP.NET її також можна використовувати для розробки веб-додатків Системи на основі C# використовують банки, логістичні та транспортні компанії, індустрія розваг і навіть управління безпілотниками.

Часто мотивацією для початку вивчення C# є мрія про кар'єру розробника ігор. Це пов'язано з тим, що на цій мові заснована одна з двох

кращих рушійних сил для розробки ігор.1 звичайно ж, мова йде про Unity, яка дозволяє створювати ігри з нуля для різних операційних систем (Windows, macOS, Linux), платформ, консолей (PlayStation, Xbox, Nintendo) і мобільних пристроїв, і навіть смарт-телевізорів. Такі ігри, як PokemonGo, Hearthstone та Angry Birds, були розроблені в Unity (і, отже, у багатьох випадках на C#). Інші дуже перспективні галузі IT-індустрії також охоче тягнуться до C#. За допомогою таких фреймворків, як ML.NET та Bot Framework (обидва під прапором Microsoft), наприклад, створюються боти та розробляється штучний інтелект.

```
using System;

class Hello
{
    static void Main()
    {
        Console.WriteLine("Hello, World");
    }
}
```

Рисунок 1.13 – Приклад написання коду на C#

Як ви можете здогадатися, в певному сенсі C# пов'язаний з C і c++. Синтаксис нагадує Java, він більш дружній, ніж C++, і набагато простіше в повсякденній роботі, ніж C. від останнього його відрізняє хоча б те, що програміст може забути про управління пам'яттю, так як середовище виконання подбає про це за нього. C# є сильно набраною мовою, що полегшує запобігання помилкам, оскільки програма сама повідомляє про них – на відміну від погано набраних мов (наприклад, C), які дозволяють повертати непередбачувані результати.

Проаналізувавши ігрові двигуни (пункт 1.1), вирішила використовувати ігровий движок Unity, оскільки цей движок не пред'являє високих вимог до навантаження.

Поєднання C# і Unity дає розробникам можливість писати простий для розуміння код і використовувати інтегроване середовище, яка значно спрощує роботу з ресурсами, анімацією, фізикою та іншими важливими аспектами гри, роблячи створення гри швидким, зручним і ефективним. Таке поєднання відкриває великі можливості для розробників всіх рівнів підготовки, від новачків, які роблять перші кроки в розробці ігор, до досвідчених професіоналів, що створюють якісні продукти для різних платформ.

1.4 Складання вимог та постановка завдання

Метою є розробка моделі штучної імунної мережі aiNET для керування ігровими персонажами у комп’ютерній грі. Для досягнення мети дослідження необхідно зробити наступні пункти:

- зробити глибокий аналіз теоретичних відомостей;
- проаналізувати всі можливі вирішення розробки моделі aiNET;
- реалізувати архітектуру MVC;
- створити гру поєднавши aiNET та MVC;
- запроектувати все на платформі Unity 3D, використовуючи мову програмування C#;
- протестувати гру.

2 ОСНОВНІ ПОЛОЖЕННЯ ТЕОРІЇ ШТУЧНИХ ІМУННИХ СИСТЕМ ТА МОДЕЛЬ ШТУЧНОЇ ІМУННОЇ МЕРЕЖІ

2.1 Теорія штучних імунних об’єктів

Штучні імунні системи (Artificial Immune System (AIS)) – це тип інтелектуальної обчислювальної системи на основі правил, натхненної принципами та процесами імунної системи хребетних тварин. Вона належить до піддисциплін біологічних і натуралістичних обчислень і пов'язана з машинним навчанням, яке є частиною ширшої галузі штучного інтелекту.

Концепція AIS почала формуватися у 1980-х роках. Він використовує інженерні методи, характерні для біологічних імунних систем. Це включає математичне моделювання імунної системи, комп'ютерну симуляцію та абстрагування імунологічних принципів в алгоритми.

Системи штучного інтелекту – це адаптивні системи, засновані на теоретичній імунології та спостережуваних функціях, принципах і моделях імунної системи, які використовуються для вирішення проблем. Оскільки це обчислювальний підхід, натхненний біологічною імунною системою, його класифікують як різновид метаевристик, натхненних природою, поряд з генетичними алгоритмами, оптимізацією мурашиних колоній, оптимізацією рою частинок тощо.

Чотири основні алгоритми ШІС постійно розвиваються: алгоритм негативного відбору (NSA), штучна імунна мережа (AINE), алгоритм клонального відбору (CSA) та алгоритм дендритних клітин (DCA).

AIS можна використовувати в різних сферах, таких як розподілене зондування, комп'ютерна безпека (виявлення вірусів, моніторинг процесів UNIX), виявлення аномалій у часових рядах, діагностика несправностей, розпізнавання образів, оптимізація та зберігання пам'яті. Таким чином, AI – це складна система, яка використовує принципи біологічних імунних систем

для вирішення складних обчислювальних завдань. AIS – це перспективний напрямок, який продовжує розвиватися і знаходити нові застосування в різних сферах. [12]

У чому різниця між штучною імунною системою та вродженою імунною системою?

Штучна імунна система (ШІС) і вроджена імунна система відрізняються за кількома параметрами, включаючи структуру, тип відповіді, адаптивність і спосіб набуття.

Структура. Вроджена імунна система – це складна мережа клітин, тканин і органів, яка присутня в організмі людини від народження. Вона включає зовнішні захисні механізми, такі як сльози і шлункова кислота, і внутрішні захисні механізми, як вроджені, так і адаптивні, у відповідь на патогени. AIS – це штучна система, яка імітує вроджену імунну систему. Це система машинного навчання на основі правил, яка використовує алгоритми та математичні моделі для виявлення та реагування на загрози.

Тип реакції. Вроджена імунна система виробляє біологічну реакцію, наприклад, виробляє антитіла для знищення патогенів. Системи штучного інтелекту натомість виробляють обчислювальну реакцію, наприклад, оповіщення системи безпеки для усунення загрози.

Адаптивність. Вроджена імунна система є адаптивною і може реагувати на широкий спектр патогенів. Може навчатися і розвиватися з часом; ШІС обмежений алгоритмами і математичними моделями, що використовуються при його розробці. Він може адаптуватися до нових моделей даних, але його адаптивність не така всеосяжна і динамічна, як у вродженої імунної системи.

Спосіб набуття. Вроджений імунітет набувається внаслідок контакту з патогенами під час інфікування або передається від матері до дитини при народженні чи через грудне вигодовування. Штучний імунітет може бути активним або пасивним. Активний штучний імунітет набувається шляхом введення мертвої або ослабленої форми збудника, зазвичай через

вакцинацію. Пасивний штучний імунітет набувається шляхом введення антитіл проти патогенів, наприклад, через препарати крові, що містять антитіла.

Штучні імунітети надихаються вродженою імунною системою і намагаються імітувати її функції, але їхня структура, типи відповіді, адаптивність і спосіб набуття принципово відрізняються. У той час як природна імунна система захищає організми від хвороб, ШІ вирішує складні обчислювальні завдання.

Існує кілька основних моделей і алгоритмів для штучних імунних систем:

Негативний відбір (Negative Selection): Цей алгоритм імітує процес навчання в імунній системі, завдяки чому створюється детектор, здатний розпізнавати аномалії та сторонні об'єкти. Він використовується для виявлення аномалій, таких як кібербезпека та моніторинг процесів.

Клональна селекція (Clonal Selection): Згідно з цим підходом, антитіла (агенти, які борються з чужорідними речовинами) клонуються і мутуються для покращення результатів. У штучних системах цей алгоритм часто використовується для оптимізації та адаптації.

Імунна мережа (Immune Network): модель імунної мережі, яка описує, як антитіла взаємодіють, щоб підтримувати імунну систему в рівновазі; в AIC вона допомагає уникнути перенасичення детекторів.

Алгоритми дендритних клітин (Dendritic Cell Algorithm) використовуються для виявлення аномалій у складних середовищах, таких як комп'ютерні мережі, і можуть ефективно відрізняти безпечні від небезпечних.

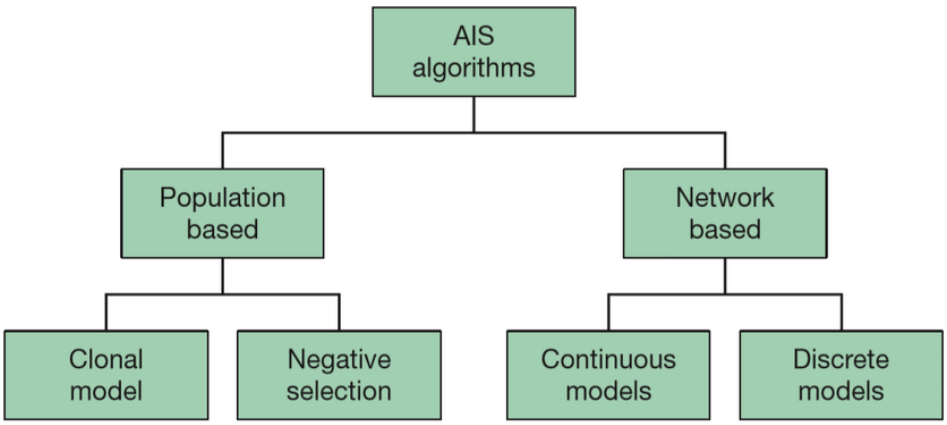

Рисунок 2.1 – Схема AIS

Іншою важливою сферою застосування AIC є оптимізація складних процесів. Алгоритми клонального відбору можна використовувати для вирішення проблем оптимізації, знаходячи оптимальне рішення для задачі з великою кількістю параметрів. Наприклад, AIC використовується в логістиці для оптимізації маршрутів доставки, розподілу ресурсів і планування виробництва. AIS також успішно використовується для моніторингу та контролю виробничого процесу. Виробництво зазвичай вимагає постійного контролю якості продукції, а виявлення аномалій у процесі може зменшити дефекти та забезпечити стабільність виробництва. Використання штучних імунних систем в цій області дозволяє швидко виявляти зміни в процесах, які можуть привести до дефектів. [12]

2.2 Модель штучної імунної мережі aiNET та імунні оператори

Інженерія штучної імунної системи (ШІС) – це напрям досліджень у галузі штучного інтелекту, що базується на моделюванні принципів роботи біологічної імунної системи для розв'язання обчислювальних задач. Серед багатьох моделей AIC виділяється модель aiNET (штучна імунна мережа). Розроблена для вирішення завдань класифікації, оптимізації та кластеризації даних, aiNET використовує методи комп'ютерного навчання та принципи

імунної відповіді, що дозволяє системам виявляти нові загрози, адаптуватися та реагувати на зміни.

Модель aiNET застосовує концепцію імунної мережі, в якій антитіла та антигени взаємодіють, утворюючи складну мережу зв'язків. Одним з основних компонентів моделі є імунний оператор, який виконує функції клонування, мутації, відбору та пам'яті, а також забезпечує адаптацію системи. Розглянемо модель aiNET, її основні принципи роботи та імунні оператори, які визначають ефективність цієї технології в сучасних інформаційних системах. [13]

aiNET – це модель штучної імунної мережі, призначена для вирішення завдань класифікації, кластеризації та виявлення аномалій. aiNET використовує принципи біологічної імунної системи, включаючи її здатність до розпізнавання, адаптації та саморегуляції. У моделі aiNET антитіла представляють можливі рішення проблеми, а антигени є мішенями, які система повинна розпізнати або класифікувати.

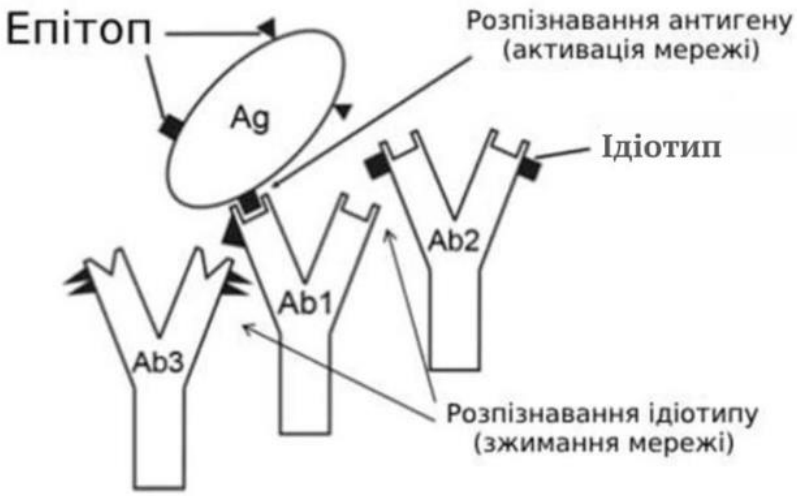

Рисунок 2.2 – Зображення принципу антитіл та антигенів в моделі aiNET

Основні етапи роботи aiNET такі:

Ініціація антитіл: в систему вводиться початкова популяція антитіл, які використовуються для розпізнавання антигенів.

Взаємодія антиген-антитіло: як тільки антиген вводиться в систему, антитіла починають розпізнавати антиген і взаємодіяти з ним для визначення подібностей і відмінностей.

Клонування та мутація антитіл: антитіла, які успішно розпізнають антигени, клонуються та мутуються. Це дозволяє антитілам адаптуватися до антигену і підвищує точність розпізнавання.

Відбір антитіл: шляхом відбору антитіл, які краще розпізнають антигени, система може усунути неефективні елементи і залишити лише найефективніші антитіла.

Видалення непотрібних антитіл: aiNET також використовує механізм видалення антитіл, функції яких перетинаються з іншими елементами або які були неактивними протягом певного часу. Це підтримує ефективність та оптимізує використання ресурсів.

Пам'ять: aiNET запам'ятовує найважливіші антитіла і може швидко реагувати на відомі загрози. [14] [15]

Оператор представлення є першим кроком у процесі роботи з імунними об'єктами в моделі aiNET. У цьому операторі кожен імунний об'єкт, що представляє певний елемент або частину проблеми, представляється у вигляді вектора або іншої структури даних, що відображає властивості цього об'єкта. Таке представлення здійснюється шляхом розміщення цього об'єкта в просторі, де система може взаємодіяти з різними імунними об'єктами, полегшуючи пошук відповідних рішень і правил класифікації.

Оператор клонування є важливою частиною адаптації в штучних імунних системах: в моделі aiNET цей оператор виконує процес клонування імунних об'єктів, які показують хороші результати в процесі вирішення проблеми або мають високу «схожість» з об'єктом, який потрібно класифікувати. Клонування створює нову копію імунного об'єкта, і кожен клон має кілька варіацій, що покращує можливість пошуку найкращого рішення. Це дозволяє «зберігати» корисні рішення, випробовуючи їхні варіації для отримання кращих результатів.

Оператор мутації використовується для створення нових варіацій об'єкта імунітету, збільшуючи різноманітність пошуку рішення і запобігаючи застряганню в локальних мінімумах. У контексті aiNET цей оператор може бути використаний для підвищення продуктивності та адаптації до нових умов, таких як вхідні вектори або певні параметри чи властивості об'єкта імунітету, наприклад, правила класифікації. Мутації допомагають системі залишатися гнучкою та адаптивною і створюють нові можливості для пошуку ефективних рішень.

Оператор супресії мережі aiNET використовується для обмеження активності об'єктів імунітету, які не вносять істотного внеску у вирішення проблеми. Суть цього оператора полягає в «придушенні» або зменшенні ваги імунних об'єктів, які не мають достатньої важливості або активності. Це зменшує розмір імунної мережі, підвищує обчислювальну ефективність і прискорює процес навчання та адаптації, оскільки система має справу лише з найбільш корисними та релевантними об'єктами.

Ці оператори надають моделі aiNET гнучкості та адаптивності, дозволяючи системі оптимізувати рішення і покращувати можливості класифікації завдяки взаємодії та еволюції імунних об'єктів. [16] [17]

2.3 Оператор презентування імунних об'єктів

У розробці ШІ використовуються численні підходи та методи, в тому числі й ті, що натхненні природними біологічними процесами. Одним з таких підходів є концепція «імунного представлення об'єкта», яка дозволяє штучним системам ефективно виявляти і реагувати на різні загрози і відхилення від норми. Цей принцип, заснований на паралелях з імунною системою живих організмів, використовується в багатьох сферах – від кібербезпеки до медицини та робототехніки. Оператори імунного представлення об'єкта в ШІ дозволяють системам адаптуватися до нових ситуацій і приймати рішення в умовах невизначеності. [18]

Щоб зрозуміти, як цей принцип стосується штучного інтелекту, необхідно коротко розглянути його застосування в біології. Імунна система людини – це складна мережа клітин, органів і молекул, які працюють разом, щоб захистити організм від інфекції. Одне з основних завдань імунної системи – розпізнавати чужорідні молекули (антигени) і реагувати на них відповідним чином. Для цього організм використовує спеціалізовані клітини, які захоплюють і обробляють антигени та презентують їх на поверхні, щоб інші клітини імунної системи могли їх ідентифікувати. Цей процес презентації важливий для того, щоб імунна система могла належним чином реагувати на загрози. У цьому процесі беруть участь молекули MHC (молекули гістосумісності), які взаємодіють з антигеном і презентують його на поверхні клітини для подальшої активації Т-лімфоцитів.

Концепція імунного представлення об'єктів була адаптована для використання в системах штучного інтелекту, включаючи алгоритми машинного навчання та системи аналізу даних. ШІ використовує цей процес для розпізнавання аномалій і виявлення шаблонів шкідливого коду та іншої небажаної поведінки. У таких системах, як і в біології, для системи важливо відбирати релевантну інформацію, щоб правильно реагувати на загрози; оператор презентації в ШІ функціонує, обробляючи вхідні дані в систему. У цьому випадку дані, схожі на антигени, проходять етап обробки, щоб визначити, чи є вони важливими для подальшої обробки, чи їх можна проігнорувати. Система аналізує ці «імунні об'єкти» і переходить до виконання конкретних дій, таких як виявлення аномалій або активація певних алгоритмів безпеки, якщо вони є підозрілими або відповідають встановленим критеріям.

Наприклад, класифікація зображень або тексту передбачає передачу «імунних мішеней» моделям ШІ шляхом представлення відповідних характеристик і особливостей даних. Алгоритми обробки даних можна порівняти з механізмами, які відбирають лише найважливіші «антигени» для подальшої обробки. Прикладом може слугувати алгоритм виявлення

аномалій у великих масивах даних. У таких випадках «антигенами» можуть бути дані, які відрізняються від нормальної поведінки системи, наприклад, аномальний мережевий трафік або незвичайні запити до бази даних тощо. Комірка, відповідальна за подання даних до ШІ, повинна виявити такі аномалії і вжити подальших заходів щодо інших компонентів системи. Дані слід надсилати на наступну адресу.

Однією з головних переваг використання концепції імунного представлення об'єктів в ШІ є адаптивна здатність системи. Важливою особливістю біологічних імунних систем є їхня здатність навчатися та еволюціонувати в умовах, що постійно змінюються. Штучні системи, що використовують імунні алгоритми, аналогічно мають здатність адаптуватися до нових загроз і змін у навколишньому середовищі. Наприклад, у системах кібербезпеки, які використовують принципи штучного імунітету, ці алгоритми можуть навчитися виявляти нові типи вірусів і шкідливих програм. Такі системи можуть автоматично оновлювати свої «антигени» та алгоритми виявлення на основі нових загроз, таким чином залишаючись ефективними у боротьбі з новими викликами. [19]

Імунні алгоритми ШІ широко використовуються в кібербезпеці для виявлення та нейтралізації шкідливих програм і хакерських атак; імунні системи ШІ можуть адаптуватися до нових типів загроз без необхідності постійно оновлювати вірусні бази даних і сигнатури. Це забезпечує більш гнучкий і швидкий захист від атак. Інші сфери застосування включають робототехніку, де системи ШІ можуть реагувати на мінливі умови навколишнього середовища, наприклад, автономні транспортні засоби, які повинні негайно реагувати на зміну дорожніх умов і поведінку інших учасників дорожнього руху.

Оператори імунного представлення об'єктів є важливим інструментом у розвитку штучного інтелекту, що дозволяє створювати адаптивні та ефективні системи для розпізнавання загроз і аномалій. Використовуючи принципи, запозичені з біології, штучний інтелект може досягти високого

ступеня самонавчання та адаптації до нових ситуацій. Очікується, що застосування таких алгоритмів у кібербезпеці, робототехніці та інших сферах дасть значні переваги для розвитку інтелектуальних систем, здатних працювати в умовах високої невизначеності.

2.4 Оператор клонування

Оператори клонування є важливим механізмом у багатьох обчислювальних системах, що використовують технології штучного інтелекту (ШІ). Вони дозволяють створювати точні або модифіковані копії об'єктів, моделей і даних для подальшого аналізу, навчання та оптимізації. Оператори клонування широко використовуються в генетичних алгоритмах, алгоритмах імунізації, нейронних мережах та еволюційних обчисленнях. У цій статті розглядаються основні принципи роботи операторів клонування, їх застосування в різних галузях штучного інтелекту та перспективи розвитку.

Клональний відбір – це назва теорії, яка пояснює, як адаптивна імунна система справляється із зовнішніми чужорідними мікроорганізмами. Основними елементами цієї системи є антитіла та антигени, де антигени - це чужорідні організми, які потрапляють в систему для подальшого розпізнавання та вивчення. В ШІ клонування – це процес створення копії інформаційного об'єкта для подальшого використання або модифікації. У генетичних алгоритмах (ГА) клонування забезпечує збереження успішних особин, а в імунологічних алгоритмах (ІА) імітує адаптивний відбір антитіл. [20]

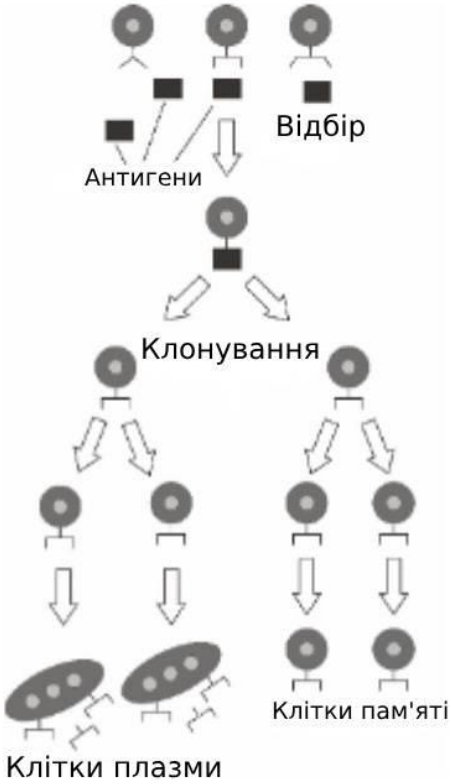

Рисунок 2.3 – Модель клонального відбору

Антитіла розпізнають чужорідні антигени з певною афінністю. Афінність – це властивість, яка вимірює силу взаємодії між антигеном і антитілом. Для клонування відбирають антитіла з високою афінністю і отримують багато клонів антитіл. Обчислюється наступним чином:

$$Af_{ij} = \left(1 + d_{ij}\right)^{-1}, \tag{2.1}$$

де Af_{ij} – афінність між об’єктами i та j, а d_{ij} – евклідова відстань між ними, значення знаходиться в проміжку (0.0; 1.0].

Клонування – це виробництво певної або розрахованої кількості ідентичних копій оригінального (батьківського) антитіла. Існують різні види клонування:

- статичне клонування – кількість отриманих клонів є вхідним параметром алгоритму і однакова для всіх клонованих антитіл;

- обернено пропорційне клонування – кількість клонів обернено пропорційна афінності, тобто є унікальною для кожного антитіла. У цьому випадку кількість клонів для кожного антитіла визначається за наступною формулою:

$$Nc = \lfloor 1 - aff(ag_i; ab_j) \rfloor, \quad (2.2)$$

тут N_c – кількість клонів, вироблених антитілом ab_j, $aff(ag_i;ab_j)$- залежить від афінності між i-м антигеном і j-м антитілом, а n – загальна кількості антитіл;

- пропорційне клонування – кількість клонів залежить від афінності клонованого антитіла до антигену і визначається за формулою: [20]

$$N_c = n \cdot aff(ag_i; ab_j). \quad (2.3)$$

2.5 Оператор мутації

Оператори мутації – один з фундаментальних механізмів штучного інтелекту (ШІ), який використовується в еволюційних обчисленнях та алгоритмах машинного навчання. Їх основна роль полягає у внесенні мінливості в популяцію розв'язків для полегшення пошуку нових розв'язків і запобігання передчасній збіжності до локального оптимуму.

Оператори мутації – це механізми, які вносять випадкові зміни в процесі оптимізації структур даних і моделей. Вносячи невеликі зміни, вони дозволяють системі вийти за межі поточного рішення і дослідити альтернативні варіанти. [21]

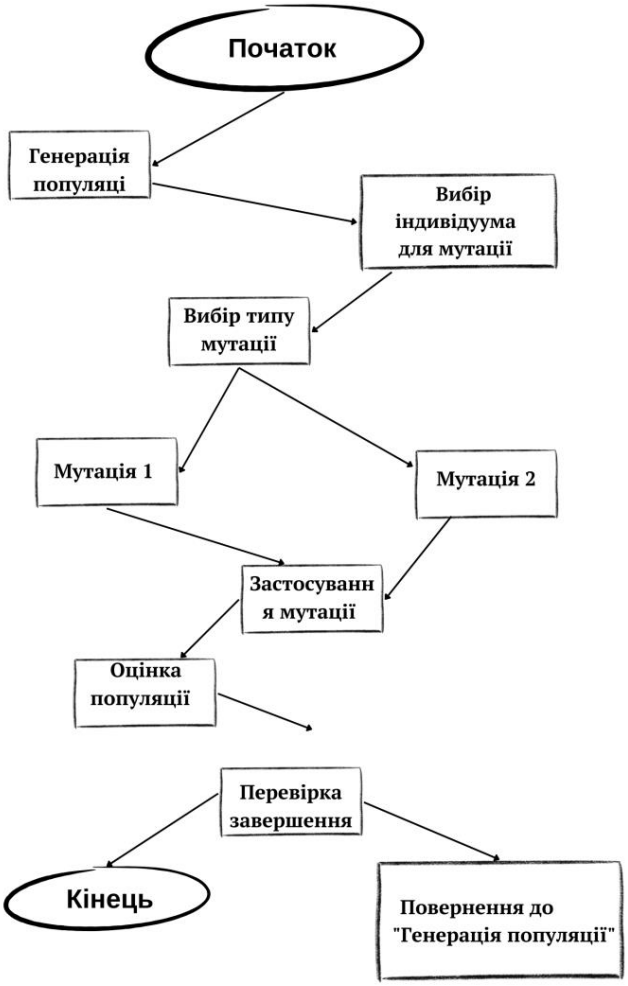

Рисунок 2.4 – Блок-схема оператора мутації

Залежно від характеру дії мутації можуть бути спрямованими або випадковими:

- випадкові мутації – зміни, що контролюються мутаційним коефіцієнтом, вносяться випадковим чином шляхом додавання або віднімання в клональну ознаку;

- спрямовані мутації – зміни вносяться в клональну ознаку і в найближчу за спорідненістю ознаку антигену Зміна вноситься після порівняння Якщо значення клональної ознаки перевищує значення антигенної ознаки, коефіцієнт віднімається від клональної ознаки; якщо значення антигенної ознаки перевищує значення клональної ознаки, то додається.

Найпоширенішими методами, що використовуються для визначення коефіцієнтів мутації, є наступні:

- пропорційна мутація – швидкість (рівень) мутації клону залежить від афінності його батьківського антитіла до даного антигену і визначається за формулою:

$$\mu = rand\left[0; aff\left(ag_i; ab_j\right)\right], \tag{2.4}$$

де N_c – кількість клонів, які виділяються антитілу ab_j для клонування, а $aff(ag;ab_j)$– афінність між i та j антитілом;

- обернено-пропорційний коефіцієнт мутація-мутація обернено пропорційний афінності антитіла до антигену і визначається за формулою:

$$\mu = rand\left[0; 1 - aff\left(ag; ab_j\right)\right]. \tag{2.5}$$

Після визначення швидкості мутації кожна характеристика мутантного клону змінюється наступним чином: [21]

$$cl.prop = cl.prop \pm \mu \cdot \left(\frac{1}{aff\left(ag_i; ab_j\right)} - 1\right). \tag{2.6}$$

Отже, оператори мутацій є фундаментальним елементом еволюційних алгоритмів у штучному інтелекті, що дозволяє шукати нові рішення, підтримувати різноманітність популяції та уникати алгоритмічної стагнації. Їх використання в нейронних мережах, алгоритмах імунізації та машинному навчанні демонструє ефективність підходу в різних галузях. Подальші дослідження мають бути спрямовані на розробку більш ефективних та адаптивних стратегій мутації.

2.6 Оператор супресії мережі

Зростаюча складність штучних нейронних мереж призвела до проблем перенавчання та надмірного використання обчислювальних ресурсів. Ефективним вирішенням цієї проблеми є оператор супресії мережі, який дозволяє контролювати активність нейронних елементів.

Оператори супресії в штучному інтелекті – це механізми, які пригнічують активність певних елементів мережі для покращення продуктивності.

Цього можна досягти шляхом тимчасового вимкнення нейронів, зменшення ваг або видалення непотрібних параметрів. У глибоких нейронних мережах супресія використовується для покращення продуктивності моделі шляхом мінімізації впливу неважливих або надмірно домінуючих ознак. Основні сфери застосування:

- уникнення перенавчання: у великих нейронних мережах з великими обсягами даних існує ризик надмірного пристосування, коли модель занадто сильно адаптується до навчального набору і втрачає здатність узагальнювати результати для нових даних. Придушення може допомогти запобігти цьому, наприклад, за допомогою регуляризації (норми L1/L2, відсіву). Регуляризація може змусити модель ігнорувати незначні властивості або випадковим чином відключати нейрони під час навчання; [22]
- покращена узагальнюваність: узагальнюваність – це здатність моделі правильно передбачати результати невидимих даних. Придушення гарантує, що модель не зосереджується на зашумлених або випадкових аспектах даних, а натомість фокусується на важливих закономірностях;
- важливість маскування: методи супресії також використовуються в архітектурі трансформаторів через механізми уваги, які дозволяють «супресувати» нерелевантну інформацію, визначаючи важливість різних частин вхідних даних.

Супресія може бути виражена математично через маску активації:

$$y_i = f(x_i) \cdot s_i, \quad s_i \in \{0,1\}, \tag{2.7}$$

де y_i – вага коефіцієнта, x_i – вхідний сигнал, s_i – коефіцієнт супресії (1 – активний нейрон, 0 – вимкнений).

Загальна модель вагового оновлення для супресії може бути виражена як:

$$\omega_i^{(t-1)} = \omega_i^{(t)} - \eta \left(\frac{\delta L}{\delta \omega_i} + \lambda s_i \right), \tag{2.8}$$

де $\omega_i^{(t)}$ – вага нейрона на ітерації , η – коефіцієнт навчання, L – функція втрат, λ – коефіцієнт регуляризації, s_i – коефіцієнт супресії (0 або 1).

У випадку регуляризації L1 (сприяє розрідженню вагових коефіцієнтів), використовують наступне рівняння:

$$L_{L1} = \lambda \sum |\omega_I|. \tag{2.9}$$

При регуляризації L2(запобігає занадто великим вагам), використовується вираз: [23]

$$L_{L2} = \lambda \sum \omega_i^2. \tag{2.10}$$

2.7 Критерії зупинки роботи моделі aiNET

Модель aiNET базується на принципах роботи імунної системи людини. Її основна мета – пошук різних рішень для задач кластеризації, оптимізації та пошуку закономірностей. Для забезпечення ефективності моделі важливо вирішити, коли зупинити прогін, щоб уникнути зайвого використання ресурсів і забезпечити високу якість результатів.

Основні критерії зупинки це:

1. Критерії сходження, базуються на аналізі динаміки роботи моделі. Мета цих критеріїв – визначити, коли модель досягає стабільного стану. Це можна зробити за допомогою: коефіцієнту силуету (Silhouette Score) – ця метрика визначає, наскільки добре об'єкти відокремлені між кластерами. Якщо коефіцієнт силуету для кількох останніх ітерацій залишається майже незмінним (ΔS≈0), модель можна зупинити; індексу Данна – Високі значення індексу Данна вказують на добре розділені та компактні кластери, наприклад, якщо значення D не змінюється понад 5% протягом кількох ітерацій, можна припустити, що кластеризація стабільна; зменшення динаміки змін антитіл – антитіла в aiNET представляють рішення, якщо кількість нових антитіл перестає змінюватися або темп їхньої появи значно сповільнюється, це сигналізує про завершення адаптивного процесу. Формально: [24]

$$\triangle_t = \frac{|A_t - A_{t-1}|}{|A_{t-1}|}, \tag{2.11}$$

де $\triangle_t$ < є (порогове значення, наприклад, є = 0.01) свідчить про стабілізацію.

Мінімальні зміни у функції придатності – функція придатності F є основою для оцінки рішень, якщо приріст якості (або зниження похибки) протягом калькох утерацій стає незначним:

$$|F_t - F_{t-1}| < \delta, \tag{2.12}$$

де δ – прийнята межа (наприклад δ = 10^{-3}), модель можна завершити.

2. Критерії оцінки якості рішення – ці критерії оцінюють, чи відповідає отримане рішення поставленим цілям. Модель завершує роботу, коли досягнуто глобального мінімуму чи максимуму функції: [25]

$$F(x^*) = \min_{x \epsilon \cap} F(x). \tag{2.13}$$

Де ∩ – область допустимих рішень. У багатовимірних задачах оптимізації важливо оцінити функцію за допомогою чисельних методів оптимізації (наприклад, градієнтного спуску).

3. Обмеження ресурсів – час виконання, якщо час виконання T перевищує встановлене значення Tmax, модель завершує роботу; кількість ітерацій, фіксована кількість ітерацій Imax є простим і надійним критерієм. Наприклад, модель завершується після 1000 ітерацій; обмеження пам'яті, якщо обсяг пам’яті M перевищує доступний ресурс Mmax, модель зупиняється.

Дослідження (De Castro & Von Zuben, 2002) показало, що адаптивне налаштування критерію зупинки може підвищити ефективність моделей aiNET на 30% порівняно з використанням жорстких фіксованих правил. Аналіз ітераційних моделей кластеризації підтверджує, що поєднання критеріїв збіжності та ресурсних обмежень дозволяє знаходити точніші рішення за менший час. Критерій зупинки є невід'ємною частиною моделі aiNET і забезпечує компроміс між якістю рішення та обчислювальними витратами. Адаптивний підхід може значно підвищити ефективність моделі. Подальші дослідження мають бути зосереджені на автоматизації вибору критеріїв для складних задач оптимізації та кластеризації.

2.8 Визначення поведінки ботів, як задача, окремий випадок задачі класифікації даних

Боти (автоматизовані програми, які виконують різні завдання в цифровому середовищі) мають значний вплив на функціонування сучасних онлайн-платформ. Залежно від ситуації, боти можуть бути корисними (наприклад, чат-боти, засоби автоматизації) або шкідливими (спам-боти, скрепери, шкідливе програмне забезпечення). Виявлення поведінки ботів є

важливим питанням для забезпечення цифрової безпеки, боротьби з дезінформацією та покращення користувацького досвіду.

Проблему виявлення ботів можна сформулювати як проблему класифікації даних. У цьому завданні дані з цифрового середовища (наприклад, з веб-сайтів і соціальних мереж) аналізуються, щоб визначити, чи належать вони користувачам або ботам. У цьому контексті завдання класифікації набуває певних особливостей, пов'язаних з динамікою поведінки ботів і великою кількістю різнорідних даних.

Завдання ідентифікації поведінки ботів у цифровому середовищі полягає в тому, щоб класифікувати ціль (користувачів або їхню поведінку) на дві основні категорії: «людина» та “бот”. Вхідні дані включають: метадані про поведінку (часові мітки, IP-адреси, тривалість сеансів тощо). Користувацький контент (текст, зображення, відео). Поведінкові патерни (частота кліків, маршрути навігації). Формально задача класифікації визначається як пошук функції класифікації. Тут простір вхідних даних відповідає людям і ботам. Для цього використовуються методи машинного навчання, які навчаються на маркованих прикладах. Платформи соціальних мереж, такі як Twitter, можуть використовувати такі дані, як частота публікацій, схожість контенту та час активності, для ідентифікації ботів. Наприклад, бот може публікувати 100 твітів на годину, тоді як середньостатистичний користувач публікує один-два твіти. [26]

Після того, як дані зібрані, їх потрібно попередньо обробити. Основні етапи Очищення даних: видалення дублікатів, пропущених та викидних значень. Перетворення даних: нормалізація числових параметрів, класифікація текстових даних. Створення ознак: генерування нових параметрів, які покращують якість класифікації.

Завдання виявлення ботів зводиться до класифікації, тобто визначення приналежності об'єкта (користувача або сесії) до «людини» або «бота». Найпоширенішим алгоритмом є логістична регресія: він використовується для вирішення проблеми двокласової класифікації. Його перевагами є

швидкість навчання та простота інтерпретації. Дерева рішень і випадкові ліси: вони об'єднують кілька дерев в ансамбль для забезпечення високої точності. Методи на основі нейронних мереж: глибокі нейронні мережі ефективно розпізнають складні патерни з великих обсягів даних. Алгоритми градієнтного підсилення: XGBoost і LightGBM підходять для табличних даних.

Глибокі нейронні мережі (DNN) підходять для завдань, де доступні великі обсяги даних. Наприклад, рекурентні нейронні мережі (RNN) можна використовувати для аналізу поведінки користувачів, тоді як згорткові нейронні мережі (CNN) підходять для аналізу візуальних даних (наприклад, аватарів і контенту).

Боти часто демонструють поведінку, яка суттєво відрізняється від поведінки звичайних користувачів. Методи виявлення аномалій (такі як Isolation Forest та One-Class SVM) можуть виявляти такі випадки без необхідності явної класифікації.

В онлайн-іграх ботів використовують для автоматизації гри. Наприклад, в іграх MMORPG боти збирають ресурси або виконують завдання, які створюють несправедливу перевагу. Для виявлення ботів аналізують такі ознаки поведінки гравців: час реакції між подіями невідповідність рівня навичок неприродні траєкторії руху персонажів. [27]

Дослідження в цій галузі показали, що гібридні методи є дуже ефективними для виявлення ботів. Наприклад, поєднання глибоких нейронних мереж з алгоритмами кластеризації може підвищити точність виявлення до 95% у багатьох сценаріях.

3 МОДИФІКАЦІЯ МОДЕЛІ AINET ДЛЯ КЕРУВАННЯ ПОВЕДІНКОЮ ІГРОВИХ БОТІВ В RTS-ІГРАХ

3.1 Оператор клонування

Оператори клонування є одним з ключових механізмів моделі aiNET і відповідають за копіювання, розвиток і подальшу адаптацію найбільш вдалих рішень (антитіл). У контексті управління поведінкою ігрових ботів у RTS-іграх оператори клонування функціонують для оптимізації стратегії та дозволяють ботам швидко адаптуватися до змін у навколишньому середовищі та виробляти ефективні моделі поведінки.

У стратегічних іграх у реальному часі (RTS) оператори клонування можуть створювати ботів з різними стратегіями: адаптуватися до стилю гри гравця: наприклад, якщо гравець віддає перевагу оборонній стратегії, клони-антитіла можуть спеціалізуватися на оборонних проривах. Оптимізація управління ресурсами: стратегії клонів можуть включати варіації розподілу ресурсів, таких як розвиток баз, будівництво армії та дослідження технологій. Покращене прийняття тактичних рішень: клони і мутації дозволяють ботам експериментувати з різними тактичними підходами, такими як маневри ухилення або несподівані атаки.

Уявімо, що боти StarCraft грають роль адміністраторів бази. Його основна стратегія полягає в одночасному розвитку економіки та оборони. Якщо в процесі гри виявляється, що гравець-суперник має агресивну стратегію початку атак на ранніх стадіях, оператор клонування модифікує початкову стратегію: збільшує частку ресурсів, що витрачаються на будівництво оборонних об'єктів. Приділяйте більше уваги ранньому виявленню противника. Тестувати різні схеми оборонних споруд. Таке адаптивне клонування дозволяє ботам реагувати на зміну умов гри в режимі реального часу.

Поведінка оператора клонування вдосконаленої моделі aiNET виконується в кілька етапів:

- вибір оптимальних антитіл Для кожного антитіла (що відповідає заданій стратегії бота) обчислюється функція придатності F, яка враховує успішність бота в тій чи іншій ігровій ситуації. Наприклад, в якості показників можуть бути використані кількість знищених ворогів, ефективність управління ресурсами, утримання стратегічних очок тощо.

$$F = \omega_1 \cdot E_k + \omega_2 \cdot R_k + \omega_3 \cdot D_k, \quad (3.1)$$

де E_k – ефективність атаки (кількість перемог над ворогами); R_k – продуктивність у зборі ресурсів; D_k – стійкість до атак; $\omega_1\omega_2\omega_3$ – агові коефіцієнти, що відображають важливість кожного аспекту.

Проаналізувавши всю інформацію про оператор клонування, в проекті створено функцію CloneGeneralPart (лістинг 3.1). Метод CloneGeneralPart слугує для того, щоб клонути або копіювати основні елементи поведінки (як-от чорну дошку, умови тестів, параметри станів, типи поведінки та вартість) з одного об'єкта на інший.

Лістинг 3.1 – Приклад CloneGeneralPart

```
public void CloneGeneralPart(IA_Behavior cloneItem)
        {
            cloneItem.blackboard = new BehaviorBlackboard();
            cloneItem.conditonsTests = conditonsTests;
            cloneItem.stateParameters = stateParameters;
            cloneItem.goalTypeBehavior = goalTypeBehavior;
            cloneItem.cost = cost;
        }
```

Новий об'єкт BehaviorBlackboard ініціалізується для cloneItem. Це надає новий простір для зберігання даних або змінних. Після цього всі параметри, пов'язані з умовами тестів, параметрами станів, типами поведінки і

витратами, копіюються в новий об'єкт. Це дозволяє клону зберігати основні налаштування поведінки.

3.2 Оператор мутації

Мутації дозволяють адаптивним системам уникати локальних оптимумів, забезпечуючи різноманітність у популяції антитіл (шаблонів, що представляють стани гри та поведінку). Завдяки мутаціям боти можуть динамічно адаптуватися до мінливих умов навколишнього середовища і знаходити нові стратегії поведінки.

Основними завданнями оператора мутацій є: внесення мутацій в популяцію антитіл, уможливити пошук нових можливих рішень для покращення стратегії бота, покращити здатність системи реагувати на невідомі ситуації.

У процесі мутації параметри антитіла випадковим чином змінюються з певною ймовірністю, яка визначається швидкістю мутації; в моделі aiNET це реалізовано в наступні етапи: вибір антитіла для мутації: вибирається антитіло, яке буде змінено. Вибір може базуватися на його ефективності або рандомізований. Застосування мутації: кожен компонент антитіла мутується з ймовірністю, що задається параметром mutation_rate. Наприклад, в RTS-грі це може бути зміна напрямку руху юніта, вибір нових пріоритетів для збору ресурсів або атак, адаптація до нового стилю гри суперника.

Роль мутації в RTS-іграх полягає в тому, щоб реагувати на динамічні ситуації. У RTS-іграх ситуації швидко змінюються, і мутація може генерувати нові стратегії, які є більш ефективними в певних ситуаціях. Мутація створює непередбачувану поведінку ботів, що робить їх більш конкурентоспроможними проти гравців та інших ботів. Дозволяє ботам поступово покращувати свою поведінку, вчитися на своїх помилках і адаптуватися до нового стилю гри своїх опонентів.

Мутація – це важливий крок у генетичних алгоритмах, де особини в популяції еволюціонують шляхом відбору, кросинговеру та мутації. Мутація слугує для введення нових генетичних варіантів у популяцію і допомагає знаходити кращі рішення. У генетичних алгоритмах «гени» можуть бути представлені по-різному. Наприклад, це можуть бути числа, двійкові рядки або більш складні структури даних. Для кожного гена в індивідуальному геномі генерується випадкове число. Якщо це число менше заданої ймовірності мутації (mutation_rate), ген модифікується. Це дозволяє підтримувати генетичне різноманіття в популяції і запобігає швидкому зациклюванню алгоритму на одному рішенні.

Фрагмент коду з лістингу 3.2 є реалізацією функції мутації генетичного алгоритму або еволюційного процесу. Функція mutate приймає два аргументи: individual – індивідуум в еволюційному процесі з геномними характеристиками. mutation_rate – ймовірність того, що даний ген мутує. Це значення має бути між 0 та 1, де 0 означає відсутність мутацій, а 1 означає, що мутація кожного гена гарантована.

Лістинг 3.2 – Приклад mutate

```
def mutate(individual, mutation_rate):
    for gene in individual.genome:
        if random.random() < mutation_rate:
            gene.mutate()
```

for gene in individual.genome – цей цикл проходить через усі елементи геному індивіда. Зазвичай, individual.genome – це список або інший повторюваний об'єкт, що містить «гени», які визначають характеристики індивіда. if random.random() < mutation_rate – функція random.random() генерує випадкове число від 0 до 1. Це випадкове число порівнюється з mutation_rate: якщо випадкове число менше за mutation_rate, для поточного гена відбувається мутація. Якщо це випадкове число більше за mutation_rate, мутація для цього гена не відбудеться.

3.3 Оператор супресії мережі

Оператор супресії може використовуватися для "вимикання" частини мережі під час навчання чи виконання для різних цілей, таких як зниження перенавчання або для емуляції певних фізичних або біологічних процесів.

Вимикаючи деякі нейрони, мережу можна зробити більш стійкою до перенавчання, оскільки вона більше не буде сильно залежати від окремих нейронів. Подібно до методів відсіювання в нейронних мережах, нейрони випадковим чином вимикаються під час навчання, щоб підвищити загальну узагальнюваність мережі. У деяких випадках це може імітувати тимчасову «втому» або відключення нейронів у природних системах. Функція ітераційно перебирає всі нейрони в мережі і для кожного нейрона порівнює випадкове число з коефіцієнтом придушення, щоб випадковим чином вирішити, чи потрібно деактивувати цей нейрон.

Якщо випадкове число менше за suppression_rate, нейрон деактивується. Ймовірність деактивації нейрона визначається значенням suppression_rate: якщо suppression_rate = 0.2, то кожен нейрон має 20% шанс бути вимкненим. Це корисно для підвищення стабільності мережі під час навчання та імітації випадкових збоїв у біологічних системах.

В лістингу 3.3 створила функцію suppress_network_activity, яка приймає два аргументи: network: об'єкт, що представляє нейронну мережу, як клас, що містить список нейронів, кожен з яких має певні властивості та методи; suppression_rate: число від 0 до 1, що визначає ймовірність того, що кожен нейрон буде деактивовано (вимкнено) на цьому проході.

for neuron in network.neurons – цей цикл проходить через усі нейрони мережі, припускаючи, що network.neurons – це список об'єктів-нейронів, які складають мережу.

if random.random() < suppression_rate: – це перевірка на генерування числа від 0 до 1, за допомогою функції random.random(). Якщо це випадкове число менше, ніж значення suppression_rate, то нейрон деактивується.

Наприклад, якщо suppression_rate дорівнює 0.3, 30% нейронів буде деактивовано.

Лістинг 3.3 – Приклад suppress_network_activity

```
def suppress_network_activity(network, suppression_rate):
    for neuron in network.neurons:
        if random.random() < suppression_rate:
            neuron.deactivate()
```

neuron.deactivate() – якщо умова виконується (випадкове число менше, ніж suppression_rate), викликається метод deactivate для конкретного нейрона. Метод передбачає, що зупинка роботи нейрона означає його вимкнення (наприклад, нейрон не бере участі в обчисленнях мережі або його активація дорівнює нулю).

3.4 Оператор вибору поведінки ігрового бота

Оператор вибору дій – важливий компонент моделі aiNET для управління ботами в RTS-іграх. Його завдання полягає у визначенні найбільш доцільної дії для бота на основі аналізу поточного стану гри. Цей оператор виступає сполучною ланкою між розпізнаванням стану навколишнього середовища та виконанням стратегії.

Основні завдання оператора забезпечення відповідності обраної дії поточній ігровій ситуації. Максимізувати ефективність дій бота в контексті поставлених цілей (наприклад, оборона бази, атака противника, збір ресурсів). Збалансувати стратегії нападу, оборони та розвідки відповідно до ситуації.

Оператор вибору дій базується на наступних аспектах:

- аналіз поточного стану гри: aiNET оцінює стан навколишнього середовища, включаючи: розташування та кількість ворожих юнітів. Рівень ресурсів та стан інфраструктури. Поточна поведінка дружніх юнітів;

- розпізнавання шаблонів: кластеризація aiNET використовується для визначення того, до якого шаблону належить поточна ситуація (наприклад, «ворожий напад», «можливість нападу», «необхідна розвідка»);

- визначення пріоритетності стратегій: дії обираються з набору стратегій на основі розпізнаних патернів. До них відносяться: оборона (розміщення підрозділів ближче до бази). Наступ (відправлення підрозділу на базу противника). Розвідка (відправка юніта для збору розвідданих);

- виконати обрану дію: обрана дія виконується через ігровий інтерфейс, а результати зворотного зв'язку використовуються для оновлення пам'яті aiNET.

У грі бот може мати різні стратегії або поведінки, наприклад, атака, захист, вивчення оточення або пошук ресурсів. Клонування дозволяє копіювати ці поведінки і застосовувати їх без зміни оригінальних налаштувань. Тому, створивши функцію IA_Attack (лістинг 3.2), яка створює новий об'єкт армії (або бота) через IA_Attack createArmy = new IA_Attack();.це дозволяє ініціалізувати нового бота для виконання дій в грі.

Рядок коду createArmy.behaviors = behaviors.ConvertAll<IA_Behavior>(c => { return c.Clone(); }); – дозволяє клонувати список поведінок бота. У грі бот може мати різні стратегії або поведінки, наприклад, атака, захист, вивчення оточення або пошук ресурсів. Клонування дозволяє копіювати ці поведінки і застосовувати їх без зміни оригінальних налаштувань. Після того, як бот створений і його поведінки клоновані, викликається метод createArmy.Init();, який ініціалізує бота. Це дозволяє налаштувати бота з початковими параметрами, що можуть залежати від обраної поведінки (наприклад, стратегія атаки чи оборони).

Лістинг 3.5 – Приклад IA_Attack

```
public class IA_Attack : IA_Direction
    {
        public override IA_Direction Clone()
        {
            IA_Attack createArmy = new IA_Attack();
            createArmy.parameterObject = parameterObject;
            createArmy.behaviors =
behaviors.ConvertAll<IA_Behavior>(c => { return c.Clone(); });
            createArmy.Init();
            return base.Clone();
        }
    }
```

Залежно від отриманих параметрів, система може вибрати одну з попередньо скопійованих і встановлених моделей поведінки. Це дозволяє боту діяти відповідно до ситуації. Після того, як стратегія або поведінка обрана, бот може бути налаштований на виконання цієї поведінки. Завдяки клонуванню кожен бот може мати унікальну конфігурацію, зберігаючи при цьому гнучкість і динамічність при виборі стратегії.

3.5 Загальна схема модифікованої моделі aiNET для керування ботами у RTS-грі

На рисунку 3.1 показано модифіковану модель aiNET для керування ботами в RTS-іграх. Схема включає такі ключові компоненти, як вхідний рівень (отримання даних гри), блоки обробки (оператори поведінки, мутації, придушення), вихідні команди (рух, атака, управління ресурсами) і зворотний зв'язок для навчання.

Вхідний шар відповідає за отримання фактичних ігрових даних, на яких ґрунтуються рішення бота. Позиції ворожих і дружніх підрозділів – відстежуйте позиції ворожих сил, союзників і ключових стратегічних об'єктів. Також включає інформацію про про наявність ресурсів на карті та їх розподіл, про наявність ресурсів на карті та їх розподіл, рівень здоров'я,

боєприпаси, енергія та інші характеристики, загальний аналіз розвитку подій у грі (атаки, оборона, контроль територій тощо). Ці дані передаються до наступного етапу – блоку прийняття рішень.

Блок обробки та прийняття рішень (Processing Units). На основі отриманих вхідних даних оператор вибору поведінки обирає одну з можливих стратегій для бота таких, як атака ворога, якщо сили бота переважають; якщо ворон сильніший, то відбувається оборонна тактика, тобто утримання позицій. Розвідка слугує для переміщення юнітів для збору інформації та виконується оптимізація ресурсів, пошук і збір ресурсів, створення нових юнітів.

У модель вбудовано механізми мутації, що дають змогу ботам адаптуватися до нових ситуацій. Наприклад, змінюються певні параметри управління юнітами, швидкість реакції – коригування затримки між діями. Регулювання агресивності – коригування ймовірності атаки або відступу. Переналаштування стратегії розвідки – боти можуть змінити алгоритм виявлення ворожих підрозділів.

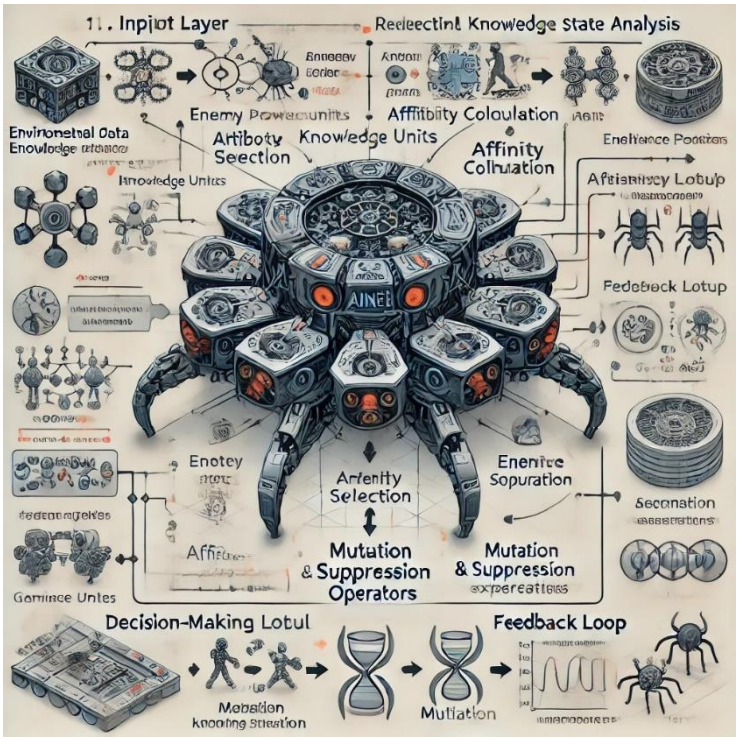

Рисунок 3.1 – схема модифікованої моделі aiNET

Оператор супресії мережі (Network Suppression Operator) - цей механізм керує активацією і деактивацією певних нейронів у моделі бота. Він відповідає за оптимізацію і зниження непотрібної обчислювальної складності.

Вихідний шар (Action Output) – після обробки всієї інформації та прийняття рішення бот виконує конкретну дію в грі. Ботам передаються такі команди, як: переміщення юнітів – пересування на визначені точки, уникнення загроз; атака ворогів – запуск бойових дій відповідно до обраної стратегії; контроль ресурсів – накази щодо збору ресурсів та їх розподілу; формування бойових груп – координація декількох юнітів для проведення тактичних операцій.

І останній етап роботи системи – це самонавчання. На основі отриманих результатів (виграш чи програш, ефективність окремих дій) бот коригує свою поведінку у майбутніх іграх. Ключове навчання з підкріпленням – алгоритми, які навчаються на основі винагород і покарань. Аналіз помилок – виявлення та усунення неефективних стратегій. Оптимізація параметрів – коригування коефіцієнтів для поліпшення продуктивності. Завдяки цьому блоку боти поступово вдосконалюються, стаючи ефективнішими та складнішими супротивниками для гравців.

Модифікована модель aiNET дозволяє адаптивно та стратегічно керувати ботами в RTS-іграх. Вона дозволяє ботам аналізувати навколишнє середовище, приймати тактичні рішення, застосовувати мутації для навчання та контролювати продуктивність за допомогою операторів придушення. Модель може бути використана не тільки в RTS-іграх, але й у військових симуляторах, автономних системах, розумних бізнес-стратегіях та будь-яких завданнях, що потребують адаптивного управління агентами.

4 АРХІТЕКТУРА ІГРОВОГО ДОДАТКУ

4.1 Реалізація рівня Model

Архітектурний патерн Model-View-Controller (MVC) – один із найпопулярніших у сучасній розробці програмного забезпечення. Він розділяє код додатка на три основні частини: модель, подання і контролер. Model відповідає за логіку застосунку та обробку даних. [28]

В архітектурах, заснованих на патерні Model-View-Controller (MVC), рівень Model забезпечує додаткову абстракцію для логічного представлення та обробки даних. На цьому рівні відбувається комунікація з базами даних, логікою та іншими сервісами. Модель має надавати додатку доступ до даних у зручній для обробки формі, незалежно від того, де вони фізично зберігаються. Наприклад, замість того щоб працювати з SQL-запитами, розробники взаємодіють із класами та методами, що відображають структури даних.

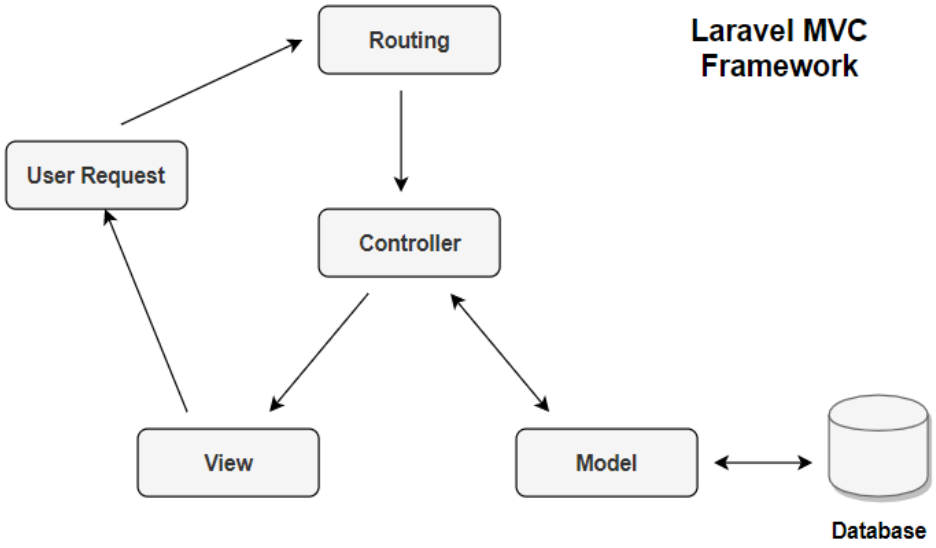

Рисунок 4.1 – Схема архітектури MVC

Model не повинна залежати від конкретної реалізації подання або контролера. Це досягається завдяки використанню чітко визначених API для взаємодії з іншими компонентами програми. Завдяки добре продуманій архітектурі шар моделей можна повторно використовувати в різних проєктах. Наприклад, модель для роботи з даними у веб-додатку може бути використана в API без будь-яких змін. База даних є основним джерелом інформації для рівня Model. Взаємодія між моделлю та базою даних зазвичай здійснюється через ORM (Object-Relational Mapping). ORM забезпечує автоматизацію перетворення даних між реляційними таблицями та об’єктами програмного забезпечення.

База даних є основним джерелом інформації для шару моделі. Взаємодія між моделлю та базою даних зазвичай відбувається через Object-Relational Mapping (ORM), що забезпечує автоматизацію перетворення даних між реляційними таблицями та програмними об'єктами.

Лістинг 4.1 – Реалізація рівня Model

```
class GameStateModel:
def __init__(self):
self.units = []
self.resources = []
self.map = None

def update_state(self, game_data):
self.units = game_data['units']
self.resources = game_data['resources']
self.map = game_data['map']
```

Оголошення класу GameStateModel в лістингу 4.1, використовується для роботи з даними стану гри. Назва класу говорить про його призначення: модель стану гри.

__init__ – це спеціальний метод, який викликається автоматично під час створення об'єкта класу. self.units – список, що містить дані про одиниці (наприклад, війська, персонажі тощо). Спочатку це порожній список.

self.resources: список, що зберігає інформацію про ресурси (в даному проекті золото). self.map – змінна для зберігання інформації про карту, спочатку вона має значення None, що означає, що карта не визначена.

Метод update_state оновлює стан гри, використовуючи дані, передані через аргумент game_data. game_data: словник, що містить дані про одиниці, ресурси та карту, ключі цього словника – це 'units', 'resources', і 'map'.

4.2 Реалізація рівня View

У сучасних комп'ютерних іграх шар представлення – це важливий елемент програмної архітектури, що відповідає за візуалізацію даних, взаємодію з користувачем і передачу інформації в зручному форматі.

Рівень View є важливим компонентом ігрового програмного забезпечення, який реалізує візуалізацію даних у реальному часі. Його роль полягає не лише у відображенні графічної інформації, але й у забезпеченні захопливого ігрового досвіду для користувача. Рівень представлення взаємодіє з рівнями моделі та контролера для забезпечення архітектурної цілісності.

Рівень View відповідає тільки за відображення даних, отриманих від Model, і взаємодію з користувачем. Логіка обробки даних залишається на рівні Моделі та Контролера. Відображення ігрового процесу в реальному часі, включно з анімацією, фізикою та оновленнями користувацького інтерфейсу (UI). Використання багаторазово використовуваних компонентів для відображення різних частин гри (HUD, меню, спливаючі вікна тощо). Оптимізація, що гарантує високу частоту кадрів (FPS) навіть на слабкому обладнанні. Адаптація до різних роздільних здатностей екрана і форм-факторів пристроїв. [29]

Перевантаження графіки і великі обсяги даних знижують FPS у бою; цього можна уникнути, використовуючи LOD (Level of Detail) для зниження рівня деталізації віддалених об'єктів; підтримка різних платформ, включно з

ПК, консолями та мобільними пристроями. Важливим аспектом є розробка користувацького інтерфейсу для різних роздільних здатностей екрана. Робота з рівнем моделі для забезпечення своєчасного відображення змін. Використання асинхронної передачі даних для мінімізації затримок – важливий момент. Реалізація механізмів взаємодії користувача з ігровими елементами, як-от натискання кнопок, використання жестів або голосове керування. Відображення фізики, освітлення та інших ефектів, що забезпечують занурення в ігровий процес. Використання технології трасування променів для досягнення фотореалістичного освітлення.

Реалізація та опис цього рівня представлена в лістингу 4.2:

Лістинг 4.1 – Реалізація рівня View

```
using UnityEngine;

namespace RTS
{
    public class GameView : MonoBehaviour
    {
        public GameObject unitPrefab;
        public GameObject buildingPrefab;

        public void CreateUnit(Vector3 position)
        {
            Instantiate(unitPrefab, position,
Quaternion.identity);
        }

        public void CreateBuilding(Vector3 position)
        {
            Instantiate(buildingPrefab, position,
Quaternion.identity);
        }
    }
}
```

Цей фрагмент коду визначає клас GameView, який відображає ігрові об'єкти, такі як юніти та будівлі у стратегічній грі в реальному часі (RTS). Клас GameView знаходиться в просторі імен RTS, що вказує на його належність до компонентів, пов'язаних зі стратегією в реальному часі. Клас

GameView успадковує від MonoBehaviour, що дозволяє йому бути прикріпленим до об'єктів у Unity та використовувати його життєвий цикл.

public GameObject unitPrefab – поле типу GameObject, яке містить посилання на префаб юніта. Префаб – це попередньо сконфігурований шаблон об'єкта, який можна повторно використовувати для створення об'єктів у грі.

public GameObject buildingPrefab – зберігає посилання на префаб будівлі.

public void CreateUnit(Vector3 position) – метод відповідає за створення нового юніта в грі. Він приймає параметр position типу Vector3, який визначає позицію в просторі, де буде створено юніт. Використовується метод Instantiate для створення екземпляра unitPrefab на заданій позиції з початковою ротацією, визначеною Quaternion.identity (без обертання).

public void CreateBuilding(Vector3 position) – метод відповідає за створення нової будівлі в грі. Він також приймає параметр position типу Vector3, який визначає місце розташування будівлі.

Клас GameView надає можливість створювати юніти та будівлі в грі за допомогою префабів. Це дає змогу відокремити логіку відображення об'єктів від бізнес-логіки та відповідає принципам архітектури Model-View-Controller (MVC).

4.3 Реалізація рівня Controller

Рівень Controller є центральною ланкою між моделлю даних і представленням. Він обробляє та інтерпретує дії користувача і передає відповідні команди іншим рівням. У класичній моделі MVC (Model-View-Controller) цей рівень виступає в ролі посередника, який забезпечує гнучкість і розширюваність системи. Рівень контролера відповідає за взаємодію між гравцем і грою, а тому має вирішальне значення для створення інтерактивних

ігор: незалежно від типу гри – 2D-платформер, 3D-шутер, стратегія, тощо – він впливає на загальний користувацький досвід.

Рівень контролера приймає сигнали від пристроїв введення, таких як клавіатури, миші, геймпади та сенсорні екрани. Наприклад, у платформері натискання кнопки зі стрілкою вправо інтерпретується як команда на переміщення персонажа в цьому напрямку. Контролер аналізує дії користувача і змінює стан гри, викликаючи методи на рівні моделі. Наприклад, у грі-шутері контролер змінює положення персонажа та його взаємодію з ігровими об'єктами. Після зміни стану моделі контролер оновлює подання і відображає зміни на екрані. Наприклад, зміни положення персонажа в 3D-просторі повинні негайно відображатися на екрані гравця. Контролер відповідає за дотримання основних правил гри. Наприклад, якщо персонаж пошкоджений пострілом противника, контролер відреагує на це зменшенням індикатора здоров'я. У сучасних іграх контролер також відповідає за керування NPC за допомогою алгоритмів штучного інтелекту. [30]

Підходи до реалізації рівня Controller:

Класичний MVC-підхід: є окремим компонентом, який посередньо зв’язує Model і View. З мінусів це те, що складні ігри можуть потребувати додаткових механізмів синхронізації. З плюсів – чіткий розподіл обов'язків полегшує розробку та тестування.

Event-Driven: контролер реагує на події, що надходять від пристроїв введення та інших частин гри. Наприклад, у багатокористувацьких іграх система подій може обробляти дії кількох гравців одночасно.

State Machines: він використовується для обробки складної ігрової логіки, яка залежить від поточного стану гри. Наприклад, коли персонаж перебуває у стані «біжить», «стрибає» або «атакує», контролер змінює стан залежно від дій гравця.

Реактивне програмування – підхід використовує потоки даних і дає змогу контролеру динамічно реагувати на зміни в моделі та дії користувача.

Це покращує масштабованість і дає змогу ефективно обробляти асинхронні події. В лістингу 4.3 показано фрагмент реалізації Controller. Клас GameController, який відповідає за обробку введення користувача та взаємодію з рівнем View (представлення) у грі жанру RTS (Real-Time Strategy). GameController успадковує від MonoBehaviour, що дозволяє йому бути прикріпленим до об'єктів у Unity та використовувати його життєвий цикл.

Лістинг 4.2 – Реалізація рівня Controller

```
using UnityEngine;

namespace RTS
{
    public class GameController : MonoBehaviour
    {
        public GameObject unitPrefab;
        public GameObject buildingPrefab;

        private GameView gameView;

        void Start()
        {
            gameView = FindObjectOfType<GameView>();
        }

        void Update()
        {
            if (Input.GetMouseButtonDown(0))
            {
                Vector3 position =
Camera.main.ScreenToWorldPoint(Input.mousePosition);
                position.z = 0;

                if (Input.GetKey(KeyCode.U))
                {
                    gameView.CreateUnit(position);
                }
                else if (Input.GetKey(KeyCode.B))
                {
                    gameView.CreateBuilding(position);
                }
            }
        }
    }
}
```

FindObjectOfType<GameView>() – знаходить компонент GameView у сцені.

Перевіряє, чи була натиснута ліва кнопка миші (Input.GetMouseButtonDown(0)). Отримує позицію миші у світі та встановлює її координату z в 0, щоб об'єкти створювались на площині. Перевіряє, чи натиснута клавіша U або B для визначення, чи створювати юніта або будівлю відповідно. Викликає відповідні методи CreateUnit або CreateBuilding з компонента GameView для створення об'єктів у грі.

Клас GameController обробляє користувацьке введення і взаємодіє з шаром View. Він отримує введення від користувача (клацання миші або клавіатури) і передає відповідні команди компоненту GameView для створення юнітів і будівель у грі. Це дає змогу відокремити логіку опрацювання введення від логіки відображення і відповідає принципам архітектури Model-View-Controller (MVC).

5 ІНСТРУКЦІЯ КОРИСТУВАЧА

Відкривши гру з’являється ігрове поле з початковою базою. В лівому верхньому куті на рисунку 5.1 відображається баланс монет, монети з кожною секундою ростуть на 1. Під монетами знаходиться рахунок захоплених веж. В правому куті знаходитсья міні-карта ігрового поля на якому відображається поточний стан гри нашого персонажа та захоплені вежі. Карту можна пересувати за допомогою стрілок вверх/вниз та масшатбувати за допомогою колеса мишки.

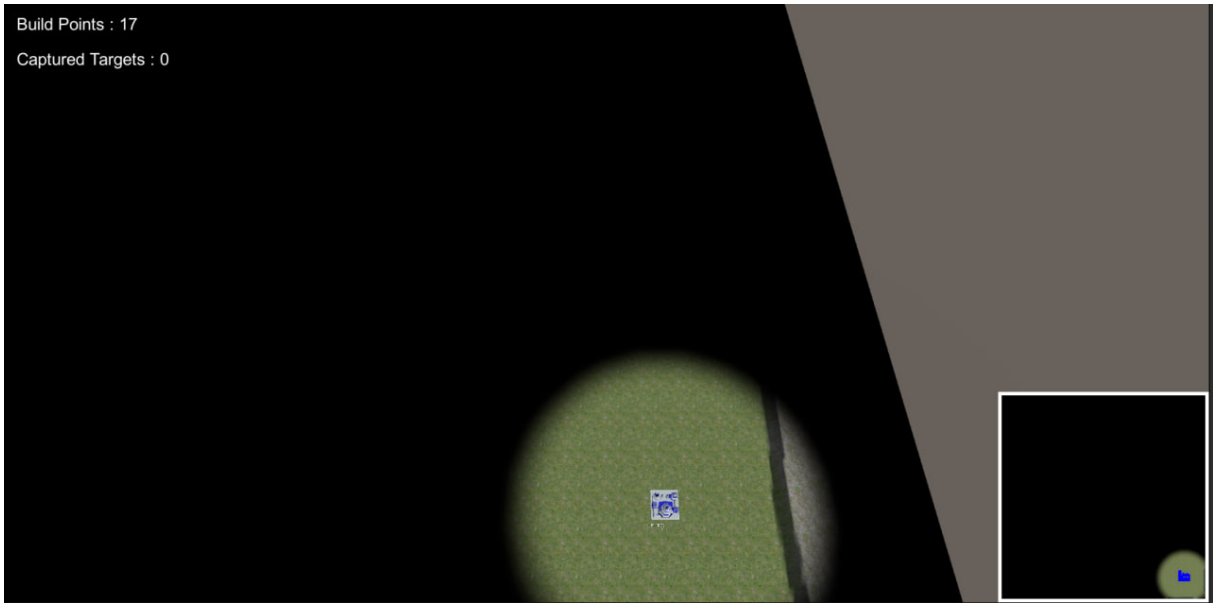

Рисунок 5.1 – Початок гри

На рисунку 5.2 зображено повноцінну ігрову карту з перешкодами та вежами, які нам потрібно захопити. Також тут показано нашу базу з правої нижньої сторони та базу противника зліва зверху. Також показані пагорби, як перешкода, через які наші гравці не можуть пройти.

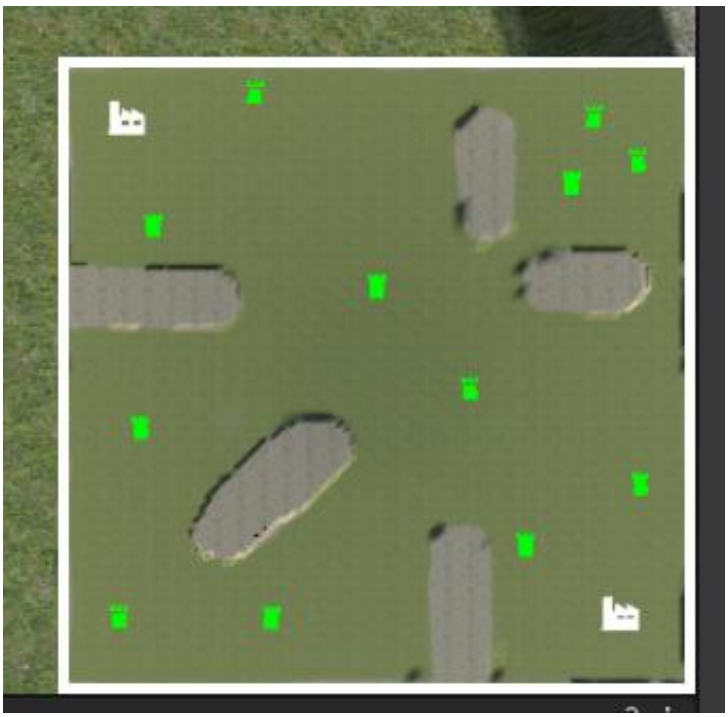

Рисунок 5.2 – Карта поля бою

Зліва внизу зображено меню магазину. В цьому магазині можна придбати 3 види бійців та 2 різних виробництва. Також на малюнку 5.3 зображено міцність будівлі.

Рисунок 5.3 – Основні компоненти

Щоб купити бійців потрібно натиснути лівою кнопкою миші на виробництво та вибрати відповідний клас бійця. На рисунку 5.4 зображено 3 види бійців від найслабшого (50 HP) до найсильнішого (100 HP).

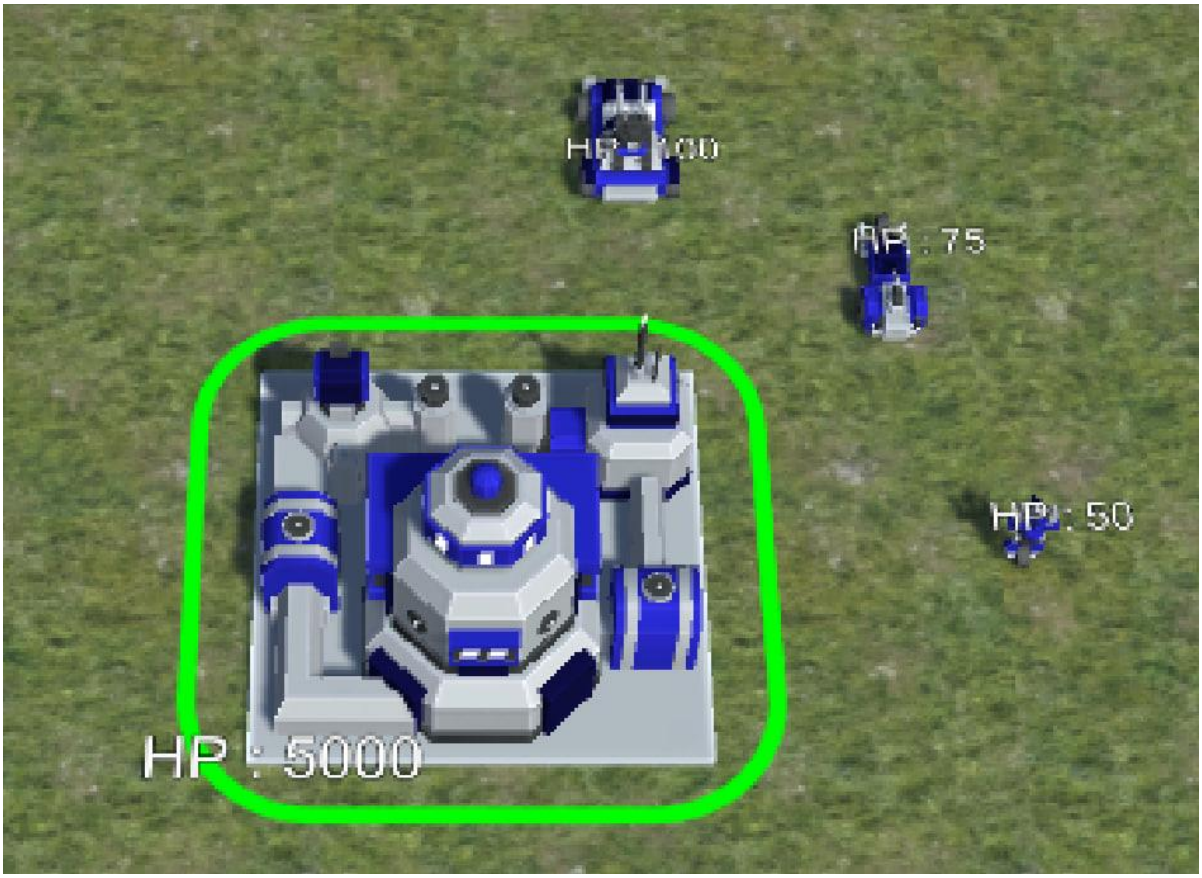

Рисунок 5.4 – Зображення кількості життя

На рисунку 5.5 зображено вибір місця для будівництва. Щоб побудувати будь-яку будівлю потрібно в меню магазина вибрати 1 з категорій Light Factory/Heavy Factory. Вибравши одну з категорій натискаємо лівою кнопкою миші на вибрану категорію в меню яке з'явиться на екрані (рисунок 5.3). Щоб розпочати будівництво потрібно перетягнути напис з меню (рисунок 5.3) на вибране місце на карті.

Рисунок 5.5 – Будівництво бази

На рисунку 5.6 зображена Heavy Factory та вежа яку потрібно окупувати. Чим більше окупованих веж має одна зі сторін тим більші шанси ця команда має на перемогу.

Рисунок 5.6 – Heavy Factory

Щоб запустити гравців потрібно виділити всіх персонажів, вибрати місце атаки та натиснути правою кнопкою миші. На рисунку 5.7 зображена окупація однієї з веж. Після окупації вежа приймає колір команди яка захопила її. Незахоплені вежі відображаються зеленим кольором.

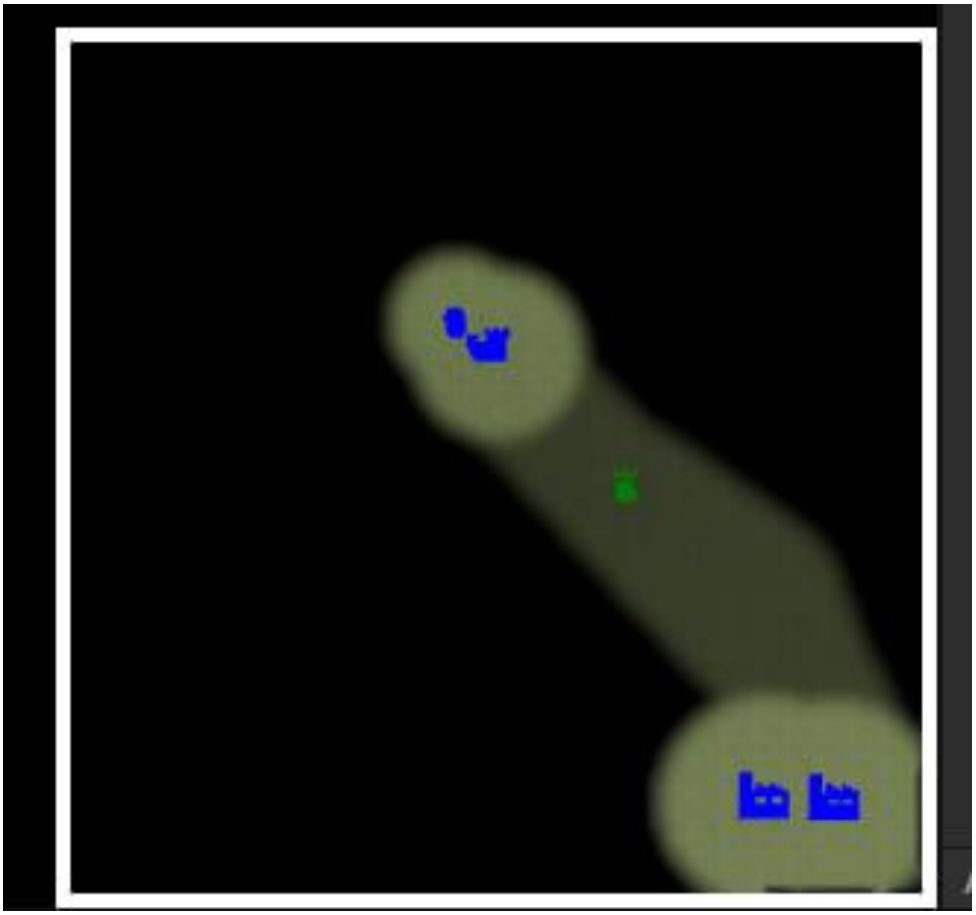

Рисунок 5.7 – Процес захоплення

На рисунку 5.8 зображено групу ботів яка є противником нашої команди. Вона зображена червоним кольором. Ці боти вміють приймати бій, атакувати та окуповувати вежі. Щоб перемогти нам потрібно вбити всіх ботів та, по можливості, захопити як найбільше веж.

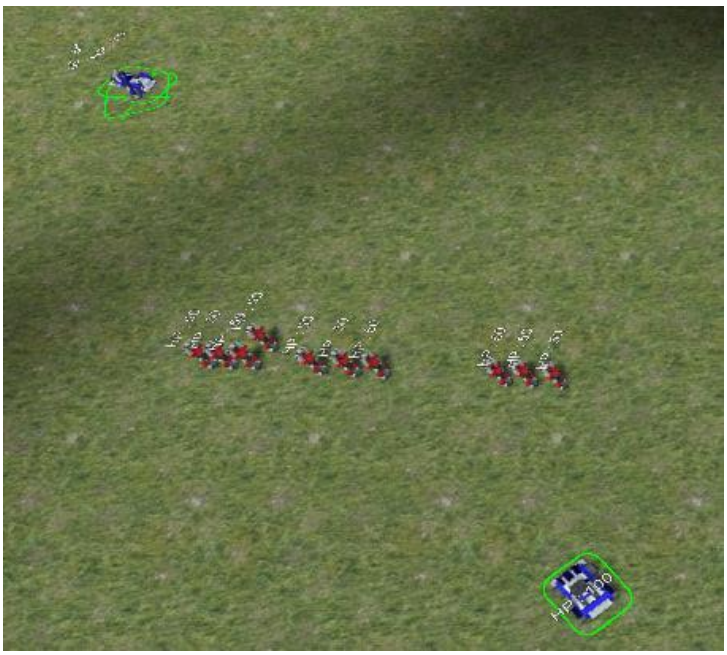

Рисунок 5.8 – Зображення ботів

На рисунку 5.9 зображено бій нашої команди з противником. Союзні війська відображаються синім кольором, в той час як ворожі війська відображаються червоним кольором. Постріли атак відображаються синім кольором, а війська обох команд, які загинули, відображаються помаранчевим кольором.

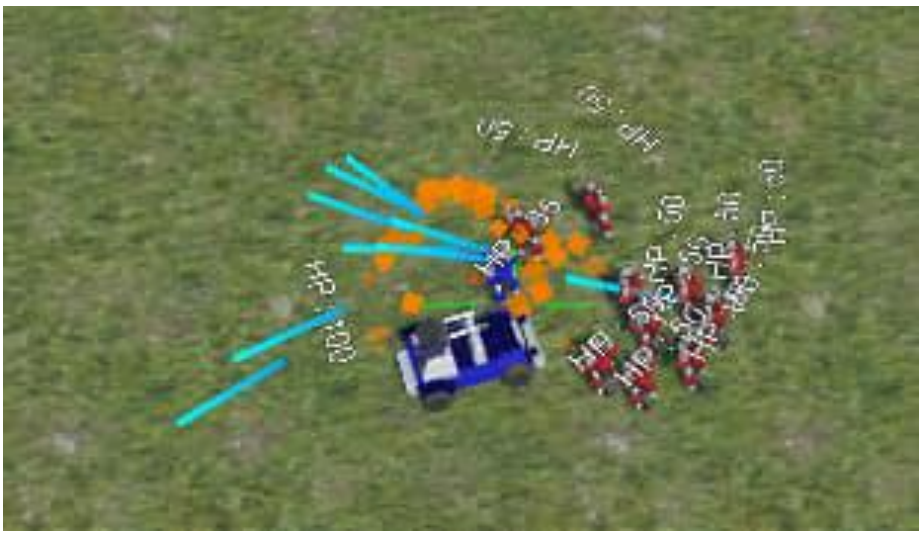

Рисунок 5.9 – Зображення бою

На рисунку 5.10 зображено сповіщення про перемогу нашої команди "Winner is Blue team".

Рисунок 5.10 – Вираз про виграш

На рисунку 5.11 зображено сповіщення про поразку виразом "GAME OVER".

Рисунок 5.11 – Вираз про поразку

За допомогою інструкції користувача в дану гру можуть грати як дорослі так і малі, гра максимально зрозуміла та проста в використані.

ВИСНОВКИ

Під час роботи було проведено комплексний аналіз предметної області, що охоплює огляд жанрів відеоігор, сучасних ігрових рушіїв і мов програмування, які використовуються для розробки. У результаті було обрано ігровий рушій Unity і мову C# як оптимальні інструменти для реалізації поставленого завдання.

Штучні імунні системи (ШІС), особливо моделі на основі aiNET, є потужним інструментом для вирішення складних обчислювальних задач, таких як класифікація, оптимізація та виявлення аномалій. Використовуючи такі оператори, як клонування, мутація і придушення, вони є високоадаптивними і ефективними, швидко реагуючи на зміни і покращуючи свої класифікаційні можливості. Хоча ШІ має обмежену адаптивність порівняно з природними імунними системами, його потенціал у практичних застосуваннях, таких як безпека і оптимізація процесів, є значним і, як очікується, поширяться в різних галузях промисловості.

Що стосується розробки ігор, то поєднання мови програмування C# та ігрового рушія Unity є ефективним і корисним інструментом для створення високоякісних ігор: C# пропонує гнучкість, простоту та високу продуктивність, а Unity спрощує роботу з ресурсами, анімацією та фізикою Unity надає інтегроване середовище, яке спрощує роботу з ресурсами, анімацією, фізикою тощо. Таке поєднання дозволяє розробникам створювати ігри для різних платформ, відкриваючи широкий спектр можливостей для професіоналів різного рівня кваліфікації та роблячи його популярним вибором у сучасній ігровій індустрії.

Таким чином, як штучні імунні системи, так і технології розробки ігор продовжують розвиватися і надають нові можливості для вирішення практичних завдань у різних галузях. Подальші дослідження та вдосконалення цих технологій сприятимуть їх ширшому застосуванню та розвитку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Video game genre [Електронний ресурс]. – Режим доступу : https://en.wikipedia.org/wiki/Video_game_genre - 26.11.2024р.

2. Contributors to Wikimedia projects. Third-person shooter - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Third-person_shooter - 26.11.2024р.

3. 10 КРАЩИХ РПГ ІГОР ДЛЯ ПРОКАЧУВАННЯ СВОГО УМІННЯ. URL: https://www.moyo.ua/ua/news/10_luchshikh_rpg_igr_kotorye_pogloshchayut_s_golovoy.html?srsltid=AfmBOopMslATUHf3CuyV-VtnHDuqrGO0LzJrX_Qdjedl--QoecTnQ4mi - 26.11.2024р.

4. RTS game [Електронний ресурс]. – Режим доступу : https://en.wikipedia.org/wiki/Real-time_strategy - 27.11.2024р.

5. Казуальні ігри: що це, якими вони бувають і як розвивається жанр • VOKI Games. Voki Games. URL: https://vokigames.com/ua/kazualni-igri-shho-cze-yakimi-voni-buvayut-i-yak-rozvivayetsya-zhanr/#:~:text=Казуальні%20ігри%20(Casual%20and%20Puzzle,іграми%20були%20Тетріс%20і%20Змійка. - 27.11.2024р.

6. Platform game [Електронний ресурс]. – Режим доступу : https://en.wikipedia.org/wiki/Platform_game - 27.11.2024р.

7. Редакція Т. Що таке стратегії в реальному часі (RTS)? Класичні представники жанру. ТСН.ua. URL: https://tsn.ua/cybersport/scho-take-strategiyi-v-realnomu-chasi-rts-klasichni-predstavniki-zhanru-1712143.html - 27.11.2024р.

8. 10 найкращих ігрових рушіїв - Ulab - SumDU. Ulab - SumDU. URL: https://ulab.sumdu.edu.ua/uk/10-najkrashhih-igrovih-rushiivC# Programming [Електронний ресурс]. – Режим доступу: https://www.programiz.com/csharp-programming/guide - 27.11.2024р.

9. Unreal Engine [Електронний ресурс]. – Режим доступу : https://www.unrealengine.com/en-US/unreal-engine-5 - 01.12.2024р.

10. Patryk. Lumberyard to darmowy silnik graficzny opracowany przez Amazon ze wsparciem dla PlayStation 4. PPE.pl - portal graczy. Serwis o grach na PS5, PS4, Xbox, Switch, PC. URL: https://www.ppe.pl/news/107573/lumberyard-to-darmowy-silnik-graficzny-opracowany-przez-amazon-ze-wsparciem-dla-playstation-4.html - 02.12.2024р.

11. Unity [Електронний ресурс]. – Режим доступу : https://unity.com/ - 02.12.2024р.

12. C# Programming [Електронний ресурс]. – Режим доступу: https://www.programiz.com/csharp-programming/guide - 02.12.2024р.

13. Radware Captcha Page. IOPscience. URL: https://iopscience.iop.org/article/10.1088/1742-6596/1203/1/012036/pdf - 02.12.2024р.

14. Żurek T. Metody sztucznej inteligencji. Lublin : Uniwersytet Marii Curie Skłodowskiej, 2011. 150 c. - 02.11.2024р.

15. arXiv.org e-Print archive. URL: https://arxiv.org/pdf/0803.3912 - 07.12.2024р.

16. Tianxing Cai. Application of Artificial Intelligence Techniques to Handle the Uncertainty in the Chemical Process for Environmental Protection, 2015. 32 c. - 03.12.2024р.

17. Artificial Immune Systems / ed. by E. Hart et al. Berlin, Heidelberg : Springer Berlin Heidelberg, 2010. - 03.12.2024р.

18. Castro L. N. d., Timmis J. Artificial Immune Systems: A New Computational Intelligence Approach. Springer, 2002. 364 c. - 03.12.2024р.

19. Analysis of the Use of Artificial Immune Systems. URL: https://www.researchgate.net/publication/350299496_Analysis_of_the_Use_of_Artificial_Immune_Systems - 04.12.2024р.

20. An Introduction to Artificial Immune Systems. URL: https://www.researchgate.net/publication/278701678_An_Introduction_to_Artificial_Immune_Systems - 04.12.2024p.

21. Artificial Immune Systems / ed. by E. Hart et al. Berlin, Heidelberg : Springer Berlin Heidelberg, 2010. - 05.12.2024p.

22. An upper bound of the mutation probability in the genetic algorithm for general 0-1 knapsack problem. arXiv.org. URL: https://arxiv.org/abs/2403.11307

23. Fast Mutation in Crossover-based Algorithms. arXiv.org. URL: https://arxiv.org/abs/2004.06538. - 05.12.2024p.

24. Contributors to Wikimedia projects. Silence suppression - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Silence_suppression. - 07.12.2024p.

25. Data Suppression Algorithms for Surveillance Applications of Wireless Sensor and Actor Networks. arXiv.org. URL: https://arxiv.org/abs/1606.09452. - 08.12.2024p.

26. Graph-Hist: Graph Classification from Latent Feature Histograms With Application to Bot Detection. arXiv.org. URL: https://arxiv.org/abs/1910.01180?utm_source=chatgpt.com. - 10.12.2024p.

27. Contributors to Wikimedia projects. Author profiling - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Author_profiling?utm_source=chatgpt.com. -10.12.2024p.

28. MVC Architecture | Ramotion Agency. Web Design, UI/UX, Branding, and App Development Blog. URL: https://www.ramotion.com/blog/mvc-architecture-in-web-application/ - 15.12.2024p.

29. Contributors to Wikimedia projects. JSP model 2 architecture - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/JSP_model_2_architecture. - 12.12.2024p.

30. Contributors to Wikimedia projects. Model–view–controller - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Model–view–controller. - 16.12.2024p.

31. Жукова, А.С. Модель штучної імунної мережі для керування поведінкою ботів у RTS-іграх / А.С. Жукова, А.А. Худяков, О.О. Фомічов // Тези доповідей І міжнародної науково-прктичної дистанційної конференції Global Trends in Science and Education. Київ. – 10-12 лютого 2025р (подано до друку).