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)

Штучна імунна система на основі моделі клонального відбору для керування поведінкою ботів у RPG-іграх

(тема)

Виконав:

студент II курсу, групи КСМм-23-1

Худяков А.А.
(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 Худякову Артему Андрійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Штучна імунна система на основі моделі клонального відбору для керування поведінкою ботів у PRG-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

3.1 Документація мову програмування C#

3.2 Література про штучні імунні системи

3.3 Документація середовища розробки Visual Studio 2019 Community Edition

3.4 Документація середовища розробки Unity

4. Перелік питань, що потрібно опрацювати у роботі

4.1 Аналіз предметної області

4.2 Аналіз використаних технологій

4.3 Програмна реалізація

4.4 Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Слайд-презентація – 25 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	Огляд існуючих методів вирішення задачі	26.11.24-10.12.24	
2	Вибір та обґрунтування методики дослідження	11.12.24-18.12.24	
3	Вибір інструментальних засобів	19.12.24-26.12.24	
4	Проведення експериментів	27.12.24-06.01.25	
5	Оформлення матеріалів кваліфікаційної роботи	07.01.25-12.01.25	
6	Подання кваліфікаційної роботи керівникові та	14.01.25-15.01.25	
	її попередній захист		
7	Подання кваліфікаційної роботи на	16.01.25-17.05.25	
	рецензування		

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: 91 с., 12 рис., 3 табл., 1 дод., 35 джерел.

ШТУЧНА ІМУННА СИСТЕМА, АФІННІСТЬ, АНТИТІЛО, АНТИГЕН, КЛОНУВАННЯ, МУТАЦІЯ, ВІДБОР КЛОНІВ, ІГРОВИЙ БОТ, КЛАСИФІКАЦІЯ , ІГРОВИЙ ЖАНР, C#, UNITY.

Метою кваліфікаційної роботи є розробка, реалізація та практичні дослідження штучної імунної системи для керування поведінкою ігрових ботів у RPG-грі, побудованої на основі моделі клонального відбору. Також в процесі роботи передбачається аналіз існуючих імунних методів, які функціоують на основі моделі клонального відбору, аналізу їх операторів, та можливостей налашування цих операторів, а також адаптація обраного метоу для вирішення задачі керування ігровими ботами.

В ході виконання кваліфікаційної роботи був проведений аналіз поширених моделей та методів, які функціонують на основі принципів роботи біологічних імунної системи людини. Також було проаналізовано вплив імунних операторів, з яких формується будь-яка модель штучної імунної системи, на результати її роботи. Окрім того було досліджено можливості щодо налаштування роботи імунних операторів та їх особливості в залежності від тієї чи іншої імунної моделі або методу. Також було реалізовано ігровий додаток у жанрі RPG, в якому передбачається керування гравцем повною кількістю обраних ігрових персонажів, які в процесі гри змагаються із іншою командою ігрових ботів, якими керує штучна імунна система, що функціонує на основі моделі клонального відбору. Розроблений програмний засіб виконує покладене на нього завдання у повному обсязі.

ABSTRACT

Master’s thesis: 91 pages, 12 figures, 3 tables, 1 appendices, 35 sources.

ARTIFICIAL IMMUNE SYSTEM, AFFINITY, ANTIBODY, ANTIGEN, CLONING, MUTATION, CLONAL SELECTION, GAME BOT, CLASSIFICATION, GAME GENRE, C#, UNITY.

The purpose of the qualification work is the development, implementation and practical research of an artificial immune system for controlling the behavior of game bots in an RPG game, built on the basis of the clonal selection model. Also, the work involves the analysis of existing immune methods that operate on the basis of the clonal selection model, the analysis of their operators, and the possibilities of configuring these operators, as well as the adaptation of the selected method to solve the problem of controlling game bots.

During the qualification work, an analysis of common models and methods that operate on the basis of the principles of the biological human immune system was conducted. The influence of immune operators, from which any model of an artificial immune system is formed, on the results of its work was also analyzed. In addition, the possibilities of configuring the work of immune operators and their features depending on a particular immune model or method were investigated. A game application in the RPG genre was also implemented, in which the player controls a full number of selected game characters, who compete with another team of game bots during the game, which are controlled by an artificial immune system that operates on the basis of the clonal selection model. The developed software tool fulfills the task assigned to it in full.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ІА – імунний алгоритм

ІМ – імунна модель

ПЗ – програмне забезпечення

СКВ – системи контролю версій

АВ – компонент штучної імунної системи, внутрішня клітина, що відповідає можливому рішенню задачі (англ., Antibody)

AG – зовнішній вплив (вірус, бактерія)(англ., Antigen)

AIS – штучна імунна система (англ., Artificial immune system)

BCA – алгоритм B-клітин (англ., B-Cell Algorithm)

CL – клон (англ., Clone)

CLONALG – алгоритм клонального відбору (англ., Clonal-Selection Algorithm)

ВСТУП

Стрімкий розвиток та поширення ігрових додатків оказав значний вплив на поведінку, навички та звички суспільства. Це обумовило поширення ігрових механік та особливостей в організації роботи багатьох мобільних та web додатків, що отримало назву гейміфікації. Елементи гейміфікації зараз можна спостерігати у багатьох сферах життєдіяльності сучасної людини – від служб замовлення та доставки їжі та систем дистанційного навчання до додатків мобільного банкінгу. Безперечно, на сьогоднішній день безпосередньо ігрова індустрія суттєво впливає на розвиток суспільства та перестає відігравати роль виключно засобів відпочинку, але й створює робочі місця та цілі галузі виробництва. Нещодавня пандемія COVID-19 також сприяла поширенню інтересу до ігрових додатків у світі. Через це ігрова індустрія характеризується стрімким розвитком, особливо ігри на ігрових консолях та мобільних пристроях.

Останнє п’ятиріччя також характеризується популяризацією систем штучного інтелекту та поширенням безкоштовного доступу до таких систем з боку суспільства для вирішення різноманітних побутових та творчих задач. За останні роки сфера штучного інтелекту перестала бути виключно дослідницькою та науковою, а стала популярною і доступною кожному, аналогічно тому, як це відбулося на ринку комп’ютерів в середині 90-х років минулого сторіччя. Більшість існуючих систем організації штучного інтелекту спираються на біологічні принципи роботи нервової або імунної систем людини, а також принципах еволюційних досліджень геному та поведінки тварин та мурах. Штучний інтелект також оказав суттєвий вплив і на розвиток сучасної ігрової індустрії.

У цій роботі розглядаються проблеми створення системи штучного інтелекту ігрових ботів, які імітують поведінку гравців у RPG-грі під час протидії діям команди персонажів користувача.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Огляд ігрових жанрів

Історія розвитку ігрових проектів бере свій початок з середини сімдесятих років минулого століття. Тоді велику популярність набули ігрові консолі, які дозволяли користувачам грати в одну або декілька ігор і виглядали як ігровий великий ігровий автомат. Потім їх змінили ігрові приставки, які можна було приєднувати до телевізорів. Їх основа перевага була у тому, що вони дозволяли користуватися спеціальними картриджами, що надавало користувача можливості використання однієї приставки за запуску безлічі ігрових додатків, що значно збільшило популярність ігрових додатків на ринку. Широке розповсюдження персональних комп'ютерів в кінці вісімдесятих та середині дев'яностих років минулого століття призвело до виникнення цілої індустрії комп'ютерних ігор, які розроблялися окремими корпораціями, що спеціалізувалися виключно на ігрових проектах. Швидке розповсюдження глобальних комп'ютерних мереж на початку двотисячних років призвело до виникнення ігрових web-проектів, які набули великої популярності, особливо серед користувачів соціальних мереж. В свою чергу революція в засобах зв'язку, яка відбулася наприкінці двотисячних і призвела до виникнення смартфонів та планшетів також призвела до виникнення цілої індустрії ігрових додатків для мобільних пристроїв. Слід зазначити, що зараз популярність ігрових мобільних додатків серед користувачів навіть перевищила популярність комп'ютерних ігор та ігрових консолей.

В процесі розвитку ігрової індустрії виникало багато специфічних видів ігрових додатків, які, в свою чергу, оказали значний вплив на формування ігрових жанрів та відокремлення їх один від іншого. Слід зазначити, що не взиваючи на довгу історію розвитку ігрової індустрії та

велике розмаїття існуючих ігрових жанрів, на сьогоднішній день не існує однієї загальної класифікації видів ігрових додатків, особливо враховуючи те, що багато сучасних та популярних серед користувачів ігрових проектів мають елементи того чи іншого ігрового жанру. Однак, за загальними рисами та спільними особливостями, можна виділити перелік головних ігрових жанрів, які є популярними на сьогодні. На рисунку 1.1 наведено загальну класифікацію актуальних на сьогодні жанрів ігрових додатків.

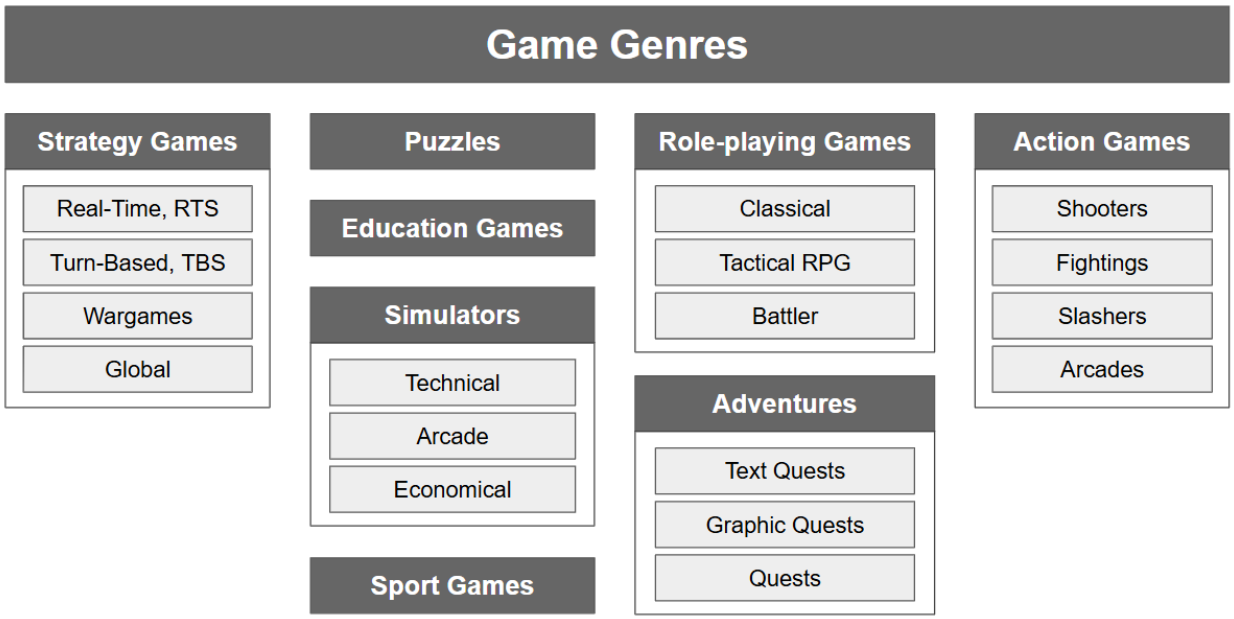

Рисунок 1.1 – Класифікація жанрів ігрових додатків

Жанр стратегічних ігор є одним з найбільш поширених видів ігрових додатків, особливо тих, що передбачають використання комп'ютера та різноманітних мобільних пристроїв. Головними особливостями цього жанру є те, що користувач виступає у ролі керівника окремої держави, військової бази або міста на карті глобального ігрового світу. При цьому основною задачею, яка ставиться перед користувачем є організації економічного, військового та технологічного розвитку розвиток своєї бази або держави, що передбачає не тільки дій на своїй локальній мапі, але й активні взаємодії із іншими користувачами, які є справжніми людьми, або ігровими ботами,

поведінкою яких керує система штучного інтелекту, вбудована у проект. Відповідно до цього, взаємодія між гравцями може приймати як торгівельний, так і міліарний характер, що передбачає як торгівлю та обмін технологіями, так і військові дії. При цьому умовою перемогу гравця зазвичай є захоплення всієї карти глобального світу та знищення всіх своїх супротивників. Історично клалося так, що першим видом стратегій є покрокові стратегії, або Turn-Based Strategies / TBS, головною особливістю яких є те, що гравці роблять свої ходи по черзі, яка змінюється лише у випадку зменшення кількості гравців,наприклад, через поразку того чи іншого користувача. Іншим поширеним видом жанру стратегічних ігор є стратегії у реальному часі, або Real Time Strategies / RTS, які передбачають відсутність черг та можливість одночасної гри з боку кількох гравців або одного гравця та визначеної кількості ігрових ботів. Найбільш популярними іграми жанру TBS є “Civilization” та “Heroes of Might and Magic”, які історично відіграли велику роль у відокремленні цього виду ігрового жанру. В свою чергу, формуючими іграми для жанру RTS є, безперечно, гра “Dune II” 1992 року випуску, яка відіграла велику роль у поширенні та набутті популярності всього ігрового жанру стратегій. Слід за нею важливими прикладами для жанру RTS є такі проекти як “Warcraft”, “Starcraft”, “Command and Conquer”, “Age of Empires” та “Cossacks”, які були створені у проміжку з 1995 по 2000 та мали довгу історію перевидання та популярності і по теперішній час.

З часом серед стратегій відокремився жанр “Wargame”, як різновид специфічних військово-економічних ігор, де користувач на початку гри має певну кількість ігрових ресурсів, яку він поступово втрачає під час проведення гри та зіткнення зі своїми супротивниками. При цьому головною метою ігор цього жанру є повне знищення ресурсів противника та збереження або відновлення якомога більшої кількості власних ресурсів. Найбільш відомими іграми цього виду є “Blitzkrieg” та “Fallout Tactics”.

Також з часом з жанру стратегічних ігор відокремився жанр глобальних стратегій, де гравець здебільшого виступає в ролі керівника окремої держави або лідера цілої нації, яка протистоїш супротивникам в той чи інший час історичного процесу. Такі ігри відрізняються своєю автентичністю та привертанні уваги до певних історичних деталей тієї чи іншої епохи, яка відтворюється у грі. Найбільш відомими іграми цього виду є “Hearts of Iron” та всі проекти серії “Total War”.

Окремим жанром ігрових додатків є логічні ігри, або Puzzles, які здебільшого орієнтуються на увагу та логіку користувача з наймолодшої за віком аудиторії. Також до цього жанру відноситься більша кількість так званих “офісних ігор”, які передбачають невелику тривалість ігрової сесії та розраховані на можливість гри під час перерви на каву чи обід. Найбільш відомими головоломками є “Сапер” та “Sokoban”.

Серед різних ігрових жанрів великою популярністю користуються освітні ігри – Education Games, які за рахунок гейміфікації дозволяють користувачу отримати або підсилити свої навички у різних науках або іноземних мовах. Також до цієї групи ігрових додатків також можна віднести клавіатурні тренажери, які набули великої популярності у середині-кінці дев’яностих років минулого століття під час стрімкого зросту попиту на персональні комп’ютери. Одним з найбільш відомих проектів цього жанру є “Lingua Leo” та інші.

Історично велику популярність серед користувачів мав жанр ігрових симуляторів – Simulators, які фокусувалися на якомога детальному відтворенні та імітації особливостей керування того чи іншого транспортного засобу, наприклад, автомобіля, літака, танка, гелікоптера, тощо. Інколи такі ігрові додатки можна одночасно віднести і до Education Games, оскільки вони можуть використовуватися для навчання пілотів літаків та курсантів військових училищ. Окрім того, у різний час велику популярність мали економічні симулятори, наприклад “SimCity”, “Cesar”, “Zeus”, “Pharaoh”, які відтворювали економічну модель та взаємодії у суспільстві окремого міста.

На сьогоднішній день найбільш популярним серед користувачів є жанр рольових ігор – Role-playing Games / RPG. Його головними особливостями є те, що користувач обирає, налаштовує та розвиває в продовж своєї гри одного чи декілька ігрових персонажів, які можуть мати свої унікальні особливості на навички, що можуть підсилюватися користувачем в процесі виконання ігрових завдань. Визначальним для таких персонажів є рівень, завдяки якому інші гравці швидко можуть зрозуміти силу та можливості того чи іншого ігрового персонажу. Слід зазначити, що у сучасних RPG можуть активно використовуватися і елементи інших ігрових жанрів, таких як Action, Adventure, Sport, та навіть Strategy. Дуже часто в RPG-іграх використовується великий відкритий ігровий світ, де гравцю надаються повні можливості до будь-яких дій. Окрім того, в RPG-проектах зазвичай передбачається використання великої кількості специфічних ігрових предметів, як то: зброя, ліки, їжа, одяг, транспорт і т.д., що додає цікавості ігровому процесу. Найбільш відомими іграми цього жанру є “Diablo”, “Fallout”, “Mount and Blade”, “Grand Thief Auto”, “Red Dead Redemption”.

Одним з найперших жанрів ігор для персональних комп’ютерів є Adventures та Quests. Головною особливістю ігор цього жанру є чітка сюжетна лінія, велика кількість логічних задач та загадок, які має розв'язати користувач задля того, що перейти на наступній рівень або отримати ту чи іншу ігрову винагороду. Найбільш відомими іграми цього жанру є “Space Quest”, “Myst” та “Syberia”, “Legend of Zelda” та “Resident Evil”.

Ігровий жанр Action історично є одним з найперших видів, та зародився це на початку вісімдесятих років минулого століття. Ігри такого жанру надають користувачу можливості, здебільшого, стрільби на ураження по різноманітним супротивникам з різних видів зброї, під час проходження різних рівнів. Найбільш відомими іграми цього жанру є “Half-life”, “Doom”, “Return to Castle Wolfenstein”. Окремо серед ігор цього жанру можна виділити файтінги, які передбачають поєдинок користувача проти бота або іншого користувача за правилами різних бойових мистецтв. Історично сере

файтінгів найбільшу популярність мала лінійка ігор “Mortal Combat”, “Tekken”, “King of Fighters” та “ Killer Instinct”.

Також значний вплив на ігровий процес та особливості жанру відіграє класифікація ігрових додатків за кількістю гравців. Так розрізняють поодинокі ігри – Single-Player Games, де користувач проходить гру без впливу з боку інших користувачів і взаємодіє лише з ігровими ботами, які імітують дії інших гравців, та ігри, розраховані на велику кількість користувачів – Multi-Player Games.

1.2 Огляд поширених систем розробки ігрових проектів

Стрімкий ріст популярності ігрових проектів в сучасному світі обумовив створення великої кількості програмних середовищ, які надають можливості розробки, налаштування та оптимізації різноманітних ігрових проектів. Інколи такі середовища називають ігровими движками, які можуть бути універсальними або специфічними. Універсальні ігрові движки можуть використовуватися для створення ігор і для ігрових консолей, і для комп’ютерів, і для мобільних пристроїв, в той час, як специфічні ігрові движки фокусуються на особливостях однієї платформи чи переліку консольних пристроїв.

Одним з перших універсальних середовищ для ігрової розробки є технологія CryEngine. Ця технологія дозволяє створювати ігрові додатки для комп’ютерів, ігрових консолей та різноманітних мобільних пристроїв. Однією з великих переваг цієї технології є можливість її безкоштовного використання для некомерційних проектів та проектів, які мають невелику кількість користувачів. Програмне середовище CryEngine надає можливості роботи із 2D- та 3D- графікою, звуком, текстурами, мешами та іншими видами ресурсів, які використовуються розробниками під час створення ігрового проекту. На час своєї появи на ринку ця технологія буда чи не єдиною універсальною технологією розробки ігор, що обумовило великий

інтерес з боку великих ігрових компаній, а також певною мірою почала формувати ринок програмного середовища, специфічного для розробки ігрових додатків. Зовнішній вигляд інтерфейсу CryEngine представлено далі.

Рисунок 1.2 – Інтерфейс CryEngine

Головними можливостями та особливостями CryEngine є зручний візуальний редактор ігрових сцен, який дозволяє легко робити версії сцен під час реалізації проекту, технологія зручного формування статичного інтерфейсу користувача додатку, широкі можливості щодо роботи із текстурами та іншою графікою для проекту. До того ж це програмне середовище забезпечує широкі можливості щодо експорту/імпорту графічних елементів сцени, створених в популярних 3D редакторах типу Blender, 3Ds Max та Maya. Для розробки частини коду ігрового проекту використовується Lua, який залучався для розробки популярних ігор типу “STALKER” та “World of Warcraft”. Також це середовище розробки ігрових проектів надає багато розробникам багато можливостей щодо використання та налаштування системи штучного інтелекту ігрових ботів, та навіть

можливості 3D моделювання ігрових об’єктів – SandBox, а також інтегрована система фізики та можливості підтримки та налаштування об’ємного звуку. Головними недоліками CryEngine я високий поріг входження користувачів та недостатній рівень документального опису можливостей ігрового движка. Слід додати, що на даний час це середовище не оновлюється, тому зараз серед розробників виникають істотні сумніви щодо подальшої підтримки цього проекту з боку його розробників.

Одним з найкращих та найпотужніших середовищ розробки ігрових проектів на теперішній час є Unreal Engine. Зараз це середовище ігрової розробки надає максимальну кількість функціональності важливої для реалізації різноманітних ефектів на можливостей. Ця платформа також дозволяє формувати збірки ігрових проектів під різні платформи: консолі, комп’ютери, мобільні пристрої та браузери. Достатньо цікавою була історія розробки цього середовища – його створила компанія Epic Games як ядро своєї популярної гри Unreal. Воно виявилося таки успішним, що після її створення в компанії вирішили додати у це середовище кілька корисних функціональних можливостей та випустити на ринок у якості окремого самостійного проекту. Головною перевагою цього програмного засобу для ігрової розробки перед аналогами є найкращі можливості щодо роботи із графікою, шейдерами та мешами поверхонь.

Головною областю застосування середовищ Unreal Engine є розробка комп’ютерних ігор, а також ігор, створених для ігрових консолей типу Sony Play Station 3 та 4 покоління. Таким чином головні програмні можливості цього середовища сфокусовані на сам перед на реалізацію 3D- ігрових проектів, які можуть користуватися великими апаратними можливостями, як то великою кількістю оперативної пам’яті, великим об’ємом жорсткого диску, процесорами із великою потужністю. Звичайно, Unreal Engine дозволяє розробникам створювати і 2D- ігри, а також ігри для мобільних пристроїв та web- ігри, але ці напрямки не набули такої популярності серед великих ігрових компаній. Іншими областями практичного використання

середовища Unreal Engine є використання для створення спец-ефектів у кінематографі, рекламних проектах, науковому моделюванні, а також моделюванні поведінки 3D-об’єктів у просторі. Окрім того, це середовище користується великим попитом серед розробників проектів із віртуальною реальністю або доповненою реальністю. Інтерфейс програмного середовища Unreal Engine представлено на рисунку 1.3.

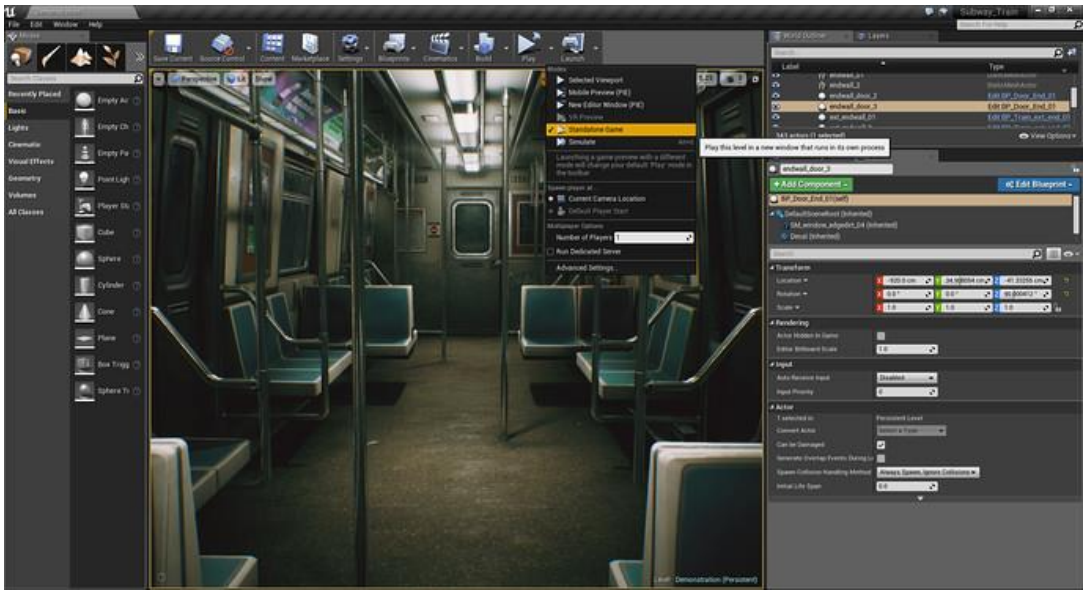

Рисунок 1.3 – Інтерфейс середовища Unreal Engine

Головними перевагами середовища Unreal Engine над іншими подібними середовищами ігрової розробки є високоякісна графіка та якість візуалізації ігрових сцен – середовище дозволяє створювати фото реалістичні візуальні ефекти, також воно має специфічну технологію Nanite, яка дозволяє розробникам користуватися можливостями візуалізованної геометрії без необхідності безпосереднього запуску проекту, окрім того, Unreal Engine має технологію Lumen, яка створена для організації динамічного глобального освітлення сцен проектів та роботи із тінями та дзеркальними поверхнями. Все це дає розробникам можливість створювати неймовірно деталізовані сцени та максимальну реалістичність освітлення у реальному часі.

Використання технології Blueprints для візуального програмування дозволяє створювати ігрову логіку для об’єктів сцени без необхідності безпосереднього написання програмного коду. Ця технологія дозволяє ігровим дизайнерам технічним художникам впливати на ігрову механіку без необхідності вивчення мови програмування C++, яка використовується у Unreal Engine. Слід зазначити, що середовище передбачає можливості комбінування Blueprints та традиційного програмування для підвищення якості проекту та швидкості ігрової розробки.

Серед переваг Unreal Engine також виділяють реалістичну фізику та анімацію. Складовою системою цього середовища є оологія Control Rig, яка фокусується на створенні реалістичної анімації ігрових персонажів та інших об’єктів сцени. В свою чергу технологія Chaos Physics забезпечує реалістичну взаємодію між об’єктами та високу якість симуляції руйнувань ігрових об’єктів та інших об’єктів сцени.

Окрім того, слід зазначити, що Unreal Engine має відкритий вхідний код, що надає великі можливості розробникам щодо модифікації та персональних налаштуваннях цього середовища під специфіку проекту. До того ж Unreal Engine вже має велику та розвинену спільноту лояльних розробників, які допомагають один одному у вирішенні специфічних задач у численних професійних форумах, що робить використання цього середовища достатньо привабливим.

Серед головних переваг Unreal Engine також слід виділити технологію LOD – Level of Detail, яка дозволяє розробникам ефективно працювати із складними сценами, які мають специфічну структуру та меші. Ця технологія надає розробникам широкі можливості тонкої оптимізації та налаштування якості компонентів проекту для різних платформ та пристроїв.

Головними недоліками Unreal Engine є занадто високі системні вимоги до апаратної частини потужностей розробника. Великі графічні можливості середовища вимагають великої потужності від центрального та графічного процесорів, велику ємність жорсткого диску та оперативної пам’яті, що

значно скорочує можливості використання Unreal Engine, особливо при розробці невеликих та некомерційних проектів. Окрім того, одним з головних недоліків середовища ігрової розробки є значно вищій поріг входження, тобто воно значно складніше у використанні ніж інші поширені аналоги. Це призводить до збільшення часу на пошук фахівців при розробці ігрових проектів, а також підвищує вартість розробки проектів.

Також велика кількість можливостей цього середовища не може бути застосована під час розробки невеликих ігрових проектів, тому у галузі мобільної ігрової розробки це середовище значно поступається за популярністю таким середовищам, як Unity та Godot.

Одним з найпопулярніших середовищ ігрової розробки, особливо орієнтованої на мобільні пристрої, на теперішній час є Unity. Компанія Unity була заснована у 2005 році як середовище ігрової розробки для платформи MacOs, але швидко розповсюдилося і на інші платформи та операційні системи. У 2010-х роках Unity стало найбільш популярним середовищем ігрової розробки для мобільних платформ.

Окрім того, ця технологія активно розвивалася у напрямках альтернативної та віртуальної реальності, кіновиробництва, архітектурного моделювання. Завдяки своїй доступності та простоті Unity залишається ключовим та найбільш затребуваним інструментом в області ігрової розробки, особливо для мобільних пристроїв, від маленьких стартапів, до великих корпорацій. Інтерфейс програмного середовища Unity представлено на рисунку 1.4.

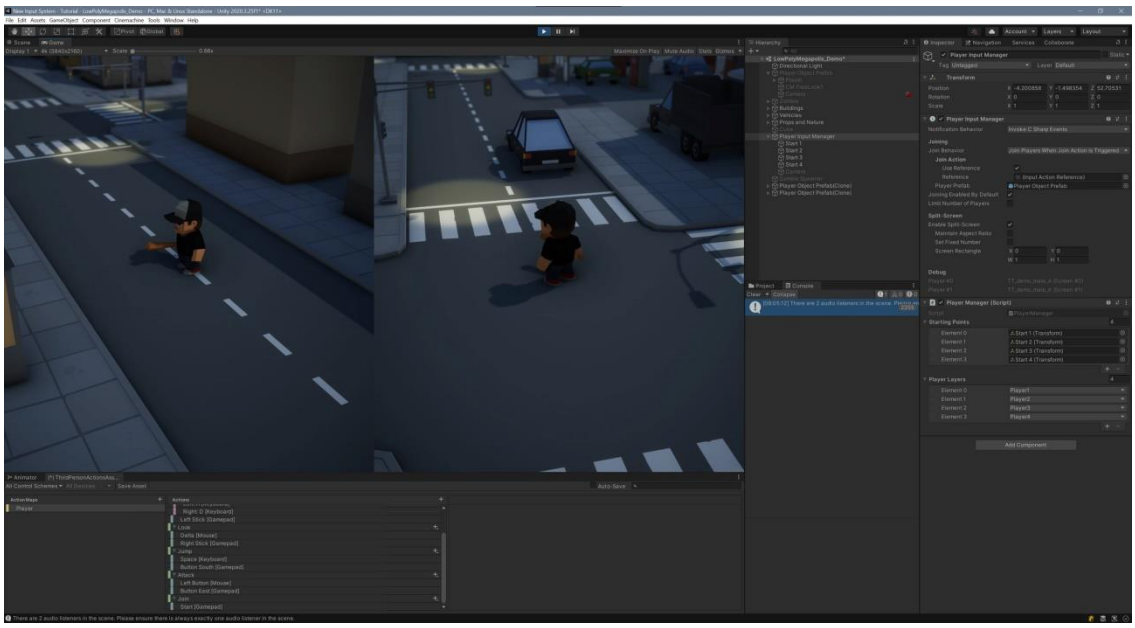

Рисунок 1.4 – Інтерфейс середовища Unity

Однією з най головних переваг Unity перед іншими середовищами ігрової розробки є універсальність та кросс-платформовість. Станом на тепер Unity дозволяє працювати із 25 різними платформами, включаючи персональні комп’ютери, мобільні пристрої (iOS, Android), ігрові консолі (PS5, Xbox) та інші. Особливою перевагою при цьому є забезпечення легкого процесу збірки гри для різних платформ з мінімальними змінами у початковому коді проекту.

Головною перевагою Unity перед аналогами є простота його освоєння та низький поріг входження. Це стало наслідком використання над простого інтерфейсу користувача у програмному середовищі Unity, використанню високорівневої мови програмування C#, яка забезпечує достатньо швидкий тем розробки завдяки тому, що є повністю об’єктно-орієнтованою та велика кількість навчальних ресурсів, які створюються як самою компанією, так і доволі активною розгалуженою спільнотою розробників. Завдяки цьому Unity забезпечило собі на сьогоднішній день найбільший рівень довіри та популярності серед невеликих ігрових стартапів. До того ж Unity представляє розробникам можливість використання специфічного Asset

Store, в якому можна придбати або завантажити безкоштовно велику кількість реалізованих патернів, плагінів та бібліотек, які дозволяють значно прискорити розробку.

Завдяки вбудованим інструментам монетизації Unity Ads та аналітики – Unity Analytics значно спрощується процес розгортання, підтримки та монетизації ігрових проектів. А завдяки підтримки технології URP (Universal Render Pipeline) та HDRP (High Definition Render Pipeline) забезпечується висока якість процесу рендерінгу та візуалізації складних сцен із великою кількістю ігрових об’єктів. Окрім того, Unity надає широкі можливості щодо налаштування рейдерів та візуальних ефектів. Це все дозволяє Unity залишатися найбільш затребуваною технологією мобільної ігрової розробки.

Серед головних недоліків Unity визначають менші можливості, порівняно із Unreal Engine, при розробці великих ігрових проектів, орієнтованих на ринок ігрових консолей та персональних комп’ютерів. Окрім того, серед недоліків Unity часто визначають низьку швидкість роботи із великими ігровими сценами, порівняно із Unreal Engine. Також часто відзначають складнощі у налаштуванні анімації ігрових персонажів у будованому інструменті MECANIM з боку розробників-початківців.

Альтернативою для Unreal Engine та Unity на сьогодні є середовище ігрової розробки Godot Engine. Це середовище було створено у 2007 році як внутрішній інструмент ігрової розробки, але потім було перероблено на більш універсальний варіант програмного засобу. Завдяки можливості безкоштовного використання, підтримці 2D- та 3D- графіки, інтеграції VR / AR та можливості створювати проекти під різні платформи, Godot швидко набув великої популярності у ігровому стартап середовищі. Завдяки інтеграції технології PBR (фізично коректного рендерігну) та підтримці технології Vulkan середовище Godot суттєво покращило свою продуктивність та зміцнило свої позиції на ринку ігрової розробки. Інтерфейс програмного середовища Godot Engine представлено на рисунку 1.5.

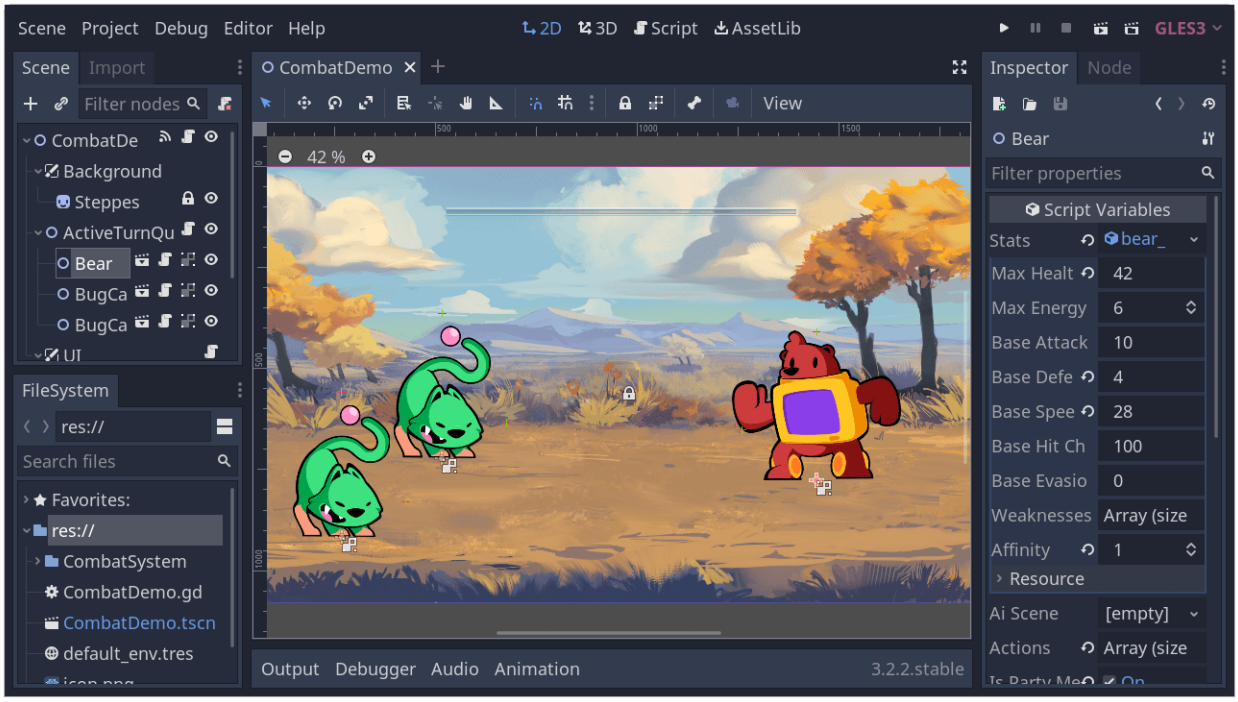

Рисунок 1.5 – Інтерфейс середовища Godot Engine

Серед переваг Godot Engine виділяють можливість безкоштовного використання та повністю відкритий початковій код, який можна достатньо легко модифікувати під специфічні потреби проекту. Також особливістю проекту є використання окремих технологій для рендерігну 2D та 3D графіки, що дозволяє досягати високої продуктивності в роботі різних ігрових проектів.

Godot Engine як і більшість наведених раніше середовищ ігрової розробки дозволяє експортувати ігри під різні платформи та операційні системи: Windows, macOS, Linux, Android, iOS, Web та різні ігрові консолі. При цьому Godot не належить жодній великій корпорації, тому розвиток цього інструменту керується виключно інтересами спільноти лояльних ігрових розробників, а не прибутками корпорації. Завдяки цьому середовище Godot має велику кількість прихильників, особливо серед невеликих start-up компаній, які створюють велику кількість ігрових ресурсів, навчальних посібників та готових до використання патернів та рішень.

Серед переваг Godot також виділяють низький поріг входження, особливо порівняно із Unreal Engine, тому вважають, що Godot є найкращим вибором для початківців ігрової розробки. Також це програмне середовище має відкритий початковий код, що дозволяє розробникам легко модифікувати редактор та інтегрувати у проекти зовнішні бібліотеки з ресурсами, створеними у сторонніх програмних засобах. Окрім того однією з переваг Godot є низькі системні вимоги до обчислювальної техніки, порівняно іншими побідними середовищами ігрової розробки.

Специфікою Godot є спосіб формування проектної архітектури, а саме Сцено-орієнтована архітектура додатків, яка використовує систему сцен та вузлів (Nodes), що дозволяє легко формувати, організовувати та модифікувати набори взаємопов’язаних ігрових об’єктів. Це забезпечує високу гнучкість і спрощує роботу над складними проектами. Окрім того, це середовище забезпечує підтримку декількох мов програмування, а саме C++, C# та власної мови програмування GDScript, схожої на популярний Python.

Серед недоліків Godot Engine виділяють недостатні графічні можливості для створення AAA-ігрових проектів, а також відсутність вбудованих інструментів для створення масштабних відкритих світів, як це відбувається у Unreal Engine. Окрім того, це середовище характеризується обмеженою підтримкою для консолей, оскільки формування консольних збірок проекту вимагає додаткових ліцензій та залучення у проект сторонніх бібліотек. Також Godot, порівняно із Unreal Engine та Unity має критично мали кількість вбудованих технологій, плагінв та інструментів, а також не має власного магазину готових рішень та високоякісних ресурсів для проектів різних типів типу Unity Asset Store або Unreal Marketplace.

1.3 Вибір засобів розробки та мови програмування

Враховуючи особливості наведених систем для розробки ігрових додатків, які користуються популярністю на ринку, а також особливості

розробки проекту в рамках даної роботи, можна зробити висновок про те, що для виконання задачі дослідницької роботи більше підійдуть два наступні середовищ: Unity або Godot Engine. Через те, що проект Cry Engine на сьогоднішній день вважається застарілим та майже не має оновлень, спрямованих на підвищення якості графіки або можливостей збирання проекту під різні платформи та операційні системи, це середовище не може бути використаним для розробки дослідницького проекту в рамках даної роботи. Звичайно середовище Unreal Engine має значно більшу кількість можливостей, ніж усі наведені аналоги, але його використання вимагає від команди розробки значного досвіду у ігровій розробці та глибокого розуміння процесів, пов’язаних із візуалізацією та рендерінгом ігрової графіки та ігрової анімації. Тому використання цього надпотужного середовища ігрової розробки для даного проекту є недоцільним.

Для остаточного визначення системи розробки можна проаналізувати мови програмування, які підтримуються в Unity та в Godot Engine. Як було зазначено вище, система Godot може підтримувати програмний код, створений за допомогою мов GDScript, C++ та C#, в той час, як середовище Unity орієнтується виключно на мову програмування C#.

Мова GDScript є спеціалізованою мовою програмування, створеною виключно для Godot Engine, яка поєднує зручність Python із потужними інструментами ігрової розробки. Серед переваг GDScript можна виділити простоту та низький поріг входження. Синтаксис цієї мови дуже подібний синтаксису Python, що робить її зрозумілою та доступною для розробників без великого досвіду роботи, а також призводить до формування чистого та лаконічного коду проекту, що спрощує та пришвидшує процес ігрової розробки. Окрім того, ця мова програмування дуже ефективно взаємодії із Godot Nodes, з яких формуються ігрові сцени у Godot та може легко корегуватися безпосередньо у редакторі Godot і не потребує залучення додаткових інструментів середовищ для написання та редагування коду. Легкість цієї мови програмування забезпечує високий темп розробки та

прототипування, особливо, порівняно із іншими мовами програмування, такими як C# та C++. Серед переваг GDScript також визначають сумісність із концепціями ООП та використання динамічної типізації даних, так само як у популярних мовах Python та JavaScript.

Головними недоліками GDScript є менша продуктивність, порівняно із мовою C#, та особливо з мовою C++. Для ресурсномістких завдань, наприклад ігрова фізика або штучний інтелект, продуктивність є вузьким місцем цієї мови. Слід зазначити, що мова GDScript не використовується ніде окрім проектів, створених на Godot, що значно зменшує можливості щодо пошуку технічно документації та готових програмних рішень. Окрім того, ця мова програмування має значно менше інструментів та можливостей для налагодження, порівняно із мовами C++ та C#, що негативно впливає на процес дебагінгу та виправлення помилок. Також серед недоліків GDScript виділяють залежність до специфічних версій Godot Engine, тобто навіть під час переходу з Godot 3x на Godot 4x можуть виникати значні несумісності різних версій GDScript в межах одного проекту, що значно ускладнює та уповільнює процес розробки проекту. Відповідно до цього, використання мова програмування GDScript недоцільно для виконання даної роботи.

Мова програмування C++ є одною з найпотужніших та найпопулярніших мов, вона використовується для розробки проектів різної складності та спрямованості. Головною перевагою цієї мови програмування є висока продуктивність, швидкість та ефективність. Мова C++ забезпечує максимальну продуктивність та контроль над ресурсами проекту завдяки широким можливостям роботи із пам’яттю. До того ж ця мова має широкий спектр можливостей використання різних стилів та технік програмування, від процедурного та структурного, до об’єктно-орієнтованого та узагальненого програмування. Через свою велику популярність та поширеність, мова C++ може використовуватися у проектах, які можна легко зібрати для різних видів популярних операційних систем та платформ, таких як Windows, macOS, Linux, iOS, Android та інші. Окрім того, ця мова програмування має

велику кількість корисних бібліотек та фреймворків, які можуть бути легко інтегровані в будь-який проект, а також велику кількість інструментів та середовищ для швидкого написання та аналізу коду проекту. Завдяки цьому проекти, створені за допомогою мови C++ зазвичай характеризуються високим рівнем стабільності та швидкості виконання при невеликій кількості системних вимог до обчислювальних ресурсів комп'ютера розробника.

Серед недоліків мови програмування C++ визначають високу складність для початківців через специфіку синтаксису та використання вказівників для керування розподілом оперативної пам'яті у проекті а також реалізацією різних патернів. Окрім того складнощі використання C++ пов'язані із необхідністю ручного керування оперативною пам'яттю та своєчасному видаленні непотрібних об'єктів під час виконання коду програми, що часто призводить до виникнення специфічних помилок, таких як Memory Leaks та Segmentation Faults. Також проекти, створені за допомогою C++ часто мають тривалий час компіляції, що негативно впливає на швидкість розробки та виправлення помилок. При цьому сам процес налагодження проекту – Debugging у проектах, створених на C++, зазвичай буде значно складнішим, ніж у проектах створених на C#, Java чи JavaScript. До того ж використання різних версій стандартів C++, наприклад, C++ 98, C++ 11, C++17 та інші можуть призводити до виникнення проблем сумісності, особливо через те, що деякі бібліотеки не оновлюються до нових стандартів. Всі ці фактори призводять до зростання технічної складності процесу розробки проектів та негативно впливають на швидкість розробки. Тому використання C++ при розробці даного дослідницького проекту може призвести до виникнення певних проблем.

Мова програмування C# створена на основі C++ та є більш сучасним її аналогом, що враховує новітні тенденції розвитку об'єктно-орієнтованого програмування та автоматизованого керування пам'яттю. Вона так само як і C++ має широкий спектр використання, в тому числі і використання при розробці ігрових проектів. Основними перевагами мови програмування C# є

достатньо простий синтаксис, схожий на популярні мови програмування C++, Java та JavaScript, можливість автоматичного керування розподілом пам’яті за допомогою інтегрованого інструменту Garbage Collector, що унеможливлює проблеми Memory Leaks, притаманні мові C++. Мова програмування також май достатньо високий рівень кросплатформовості, та дозволяє формувати збірки проектів для Windows, macOS, Linux, Android, iOS та інші операційні системи та платформи. Також серед переваг цієї мови можна виділити велику кількість інструментів та бібліотек з корисними функціями, які можуть бути розроблені як корпорацією Microsoft, так і іншими відомими на IT-ринку компаніями. Окрім того, середовища розробки C#-проектів мають інтегрований інструмент NuGet, який спрощує менеджмент бібліотек та пакетів із корисною функціональністю, які можуть бути додані до проектів, а також спрощувати процес налаштування та оптимізацію проекту та процес погуку помилок. Слід зазначити, що мову програмування C# зазвичай рекомендують для ігрової розробки, особливо разом із програмним середовищем Unity Engine.

Серед недоліків C# відзначають менші можливості, порівняно із C++, особливо це стосується питань низькорівневого програмування та керування менеджментом пам’яті. Також C# програє C++ у швидкодії. Окрім того, C# залежний від платформи .NET та більш складний для розробників без великого практичного досвіду роботи, порівняно із Python та GDScript. Проекти, створені за допомогою мови C# зазвичай мають більш високі вимоги до апаратного забезпечення, ніж проекти, створені на C++. Однак, незважаючи на ці недоліки, C# вважають одним з найкращих варіантів мови програмування для ігрових додатках, особливо враховуючи можливості та переваги середовища ігрової розробки Unity. Тому у якості технології розробки було ігри було обрано саме технологію Unity та мову програмування C#.

1.4 Складання вимог та постановка задачі проектування

Метою роботи є розробка імунної моделі клонального відбору для керування ботами у RPG-грі. Відповідно до зазначеної мети виникає необхідність вирішення наступних завдань:

- аналіз завдання, пошук необхідної інформації;
- проектування та реалізація ігрового додатку на основі одного з багатошарових архітектурних патернів;
- проектування та реалізація моделі клонального відбору для прийняття рішення щодо поведінки ігрових ботів у RPG-грі;
- налаштування імунних операторів для оптимізації процесу керування ігровими ботами;
- адаптація реалізованої моделі клонального відбору у гру для керування ботами, враховуючи особливості ігрового дизайну.

2 ОСОБЛИВОСТІ ВИКОРИСТАНИХ ТЕХНОЛОГІЙ ТА ІГРОВИЙ ДИЗАЙН ПРОЕКТУ

2.1 Особливості компонентно-орієнтованої архітектури Unity-проектів

Однією з особливостей середовища Unity є використання компонентно-орієнтованої архітектури проектів. Така архітектури побудови ігор представляє собою специфічний підхід до побудови, налаштування та використання ігрових об’єктів – GameObjects, яка передбачає, що кожен такий об’єкт ігрової сцени додатку, незалежно від того, чи він активний та буде мати анімацію та можливості керування користувачем, чи він використовуються у якості статичного об’єкту сцени, є контейнером для компонентів різних типів. При цьому сам ігровий об’єкт не несе ніякої функціональності, окрім як можливість формування ієрархії та носіїв інстальованих компонентів. Такий підхід дозволяє створювати складні ігрові об’єкти шляхом додавання та налаштування окремих специфічних компонентів, які зазвичай є незалежними один від одного та виконують свої специфічні завдання на сцені ігрового проекту.

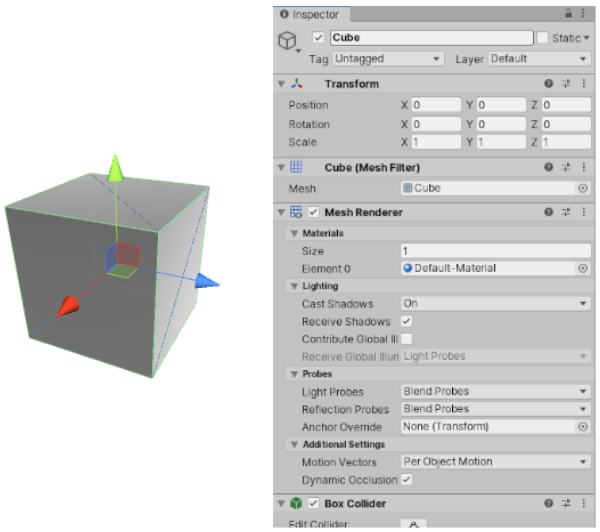

Рисунок 2.1 – Ігровий об’єкт Unity та його компоненти

Відповідно до цього основними концепціями використання компонентно-орієнтованої архітектури в проектах Unity: поняття ігрового об’єкт, компоненти, ієрархія, розширюваність, повторне використання та зберігання прототипів.

Ігровий об’єкт – GameObject в Unity – це основний контейнер, який представляє будь-який об’єкт у грі, від персонажа, яким керує користувач, до джерела світла, камери, чи порожній об’єкт, який взагалі не буде відображатися користувачу під час гри. При цьому GameObject не має власної функціональності а виконує лише роль носія компонентів, які наділяють його специфічним відображенням, поведінкою та властивостями.

Компоненти в Unity використовуються як основні функціональні елементи, які додаються до того чи іншого ігрового об’єкту на сцені та визначають особливості його поведінки. При цьому кожен компонент створений для виконання специфічної функції або окремої специфічної задачі, наприклад, компонент Transform використовується для визначення розмірів та позиції ігрового об’єкту сцени у тривимірному просторі; компонент Renderer – для відображення того чи іншого мешу (2В-моделі) на сцені; компонент Collider – для визначення факту зіткнення із іншими ігровими об’єктами на сцені та взаємодії із їх компонентами; компонент RigidBody – для додавання ігровому об’єкту сцени фізичних властивостей та передавання тих чи інших фізичних сил; компонент Script – для додавання власної поведінки, реалізованої за допомогою мови програмування C#. Зазвичай компонент є незалежними один від одного, але Unity має також перелік взаємозалежних компонентів, як додаються до ігрового об’єкту разом, та реалізовують складний функціонал із багатьма налаштуваннями.

Під концепцією ієрархії в Unity розуміють специфіку організації ігрових об’єктів та їх функціональних компонентів на сценах проекту. Відповідно до цього, всі ігрові об’єкти однієї сцени створюють одну ієрархію, яка передбачає виділення батьківських – parent, та вкладених – child об’єктах так, щоб дочірні вкладені ігрові об’єкти успадковували будь-які

трансформації та зміни батьківського об’єкту. Таким чином, будь які зміни батьківського об’єкту призводять до неминучих змін у всіх його вкладених дочірніх об’єктах. Слід зазначити, що при цьому вкладі об’єкти одночасно можуть мати свою складну ієрархію та бути батьківськими по відношенню до своїх вкладених дочірніх об’єктів.

Концепція розширюваності компонентно-орієнтованої архітектури в Unity передбачає мінімальну кількість обмежень щодо додавання або видалення компонентів без зміни самого ігрового об’єкту. Це призводить до значного підвищення гнучкості та масштабованості проекту. Окрім того, розробник може самостійно створювати будь-які компоненти і додавати їх у ігрові об’єкти за потреби. Слід зазначити, що і саме середовище Unity також є розширюваним та дозволяє розробникам легко створювати свої власні плагіни для специфічного налаштування ігрових об’єктів та їх компонентів.

Під концепцією повторного використання та зберігання прототипів розуміють те що будь-який компонент може бути пере використаним на різних ігрових об’єктах будь-якої сцени проекту. Таким чином створення та використання компонентів може розповсюджуватися не на один конкретний ігровий об’єкт сцени, а на безліч різних об’єктів проекту. Окрім того, ця концепція передбачає можливість створення так званих префабів – Prefabs – збережених елементів ієрархії ігрових об’єктів, які самі по собі несуть велику завершену частину функціональності ігрового додатку та можуть бути збереженими на пере використаними як в рамках однієї, так і в рамках декількох ігрових сцен проекту. При цьому керування та об’єктами, створеними за допомогою префабу можна робити безпосередньо через цей префаб, додаючи, видаляючи, чи змінюючи налаштування його компонентів.

Завдяки цьому компонентно-орієнтована архітектура в Unity має декілька переваг. Серед цих переваг виділяють гнучкість, як можливість легкого комбінування та налаштування компонентів на різних ігрових об’єктах сцени. Також серед переваг виділяють масштабованість, яка полягає у легкому додаванні нових функцій будь-якому ігровому об’єкту сцени без

необхідності переписувати код проекту або створювати нові специфічні класи чи інші типи даних. Повторне використання підготовлених та налагоджених ігрових об’єктів, збережених у вигляді префабів, підвищує швидкість та ефективність розробки проекту. Окрім того, важливою перевагою цієї архітектури є простота підтримки проектів завдяки тому, що кожен компонент відповідає за свою невеличку окрему функцію і його налагодження або навіть видалення не відобразиться на налаштуваннях інших компонентів на ігрових об’єктах сцени.

Головними недоліками цієї архітектури називають ускладнення структури проекту впродовж довгої його розробки. Таке можна спостерігати лише на великих проектах, де велика кількість компонентів може призвести до формування складних та заплутаних ієрархій та виникнення залежностей між ними. Також недоліком цього підходу вважають можливе виникнення затримок у виконанні багатьох компонентів, яке може спостерігатися в моменти, коли багато компонентів на різних ігрових об’єктах працюють одночасно. Це може призвести до менш ефективної роботи на складних сценах або під час використання складних префабів. Однак, незважаючи на це, компонентно-орієнтований підхід має значно більше переваг порівняно із декількома виявленими недоліками. Тому він якнайкраще відходить для розробки невеликих ігрових проектів.

2.1 Тенденції сучасного ООП та їх вплив на розробку проектів

Поява та розповсюдження концепцій об’єктно-орієнтованого програмування (ООП) зробило революцію у процесі розробки програмного забезпечення, завдяки впровадженню нового рівня структуризації коду проекту, підвищення швидкості розробки, можливостям пере використання коду, можливостям швидкої зміни початкового коду та підвищенню якості читання. Розвиток ООП сприяв створенню гнучких та багаторазово використовуваних компонентів – класів та об’єктів, що значно знизило

втрати на розробку, розгортання та підтримку проектів. Саме завдяки ООП з’явилися перші патерни програмування та стандарти організації коду проектів, які підвищили якість та надійність програмних засобів. Перші концепції ООП з’явилися у 1960-х роках завдяки мові програмування Simula, яка дозволяла симулювати поведінку реальних об’єктів та процесів у коді проекту. Саме тоді на світ з’явилося поняття класів, об’єктів та ідея можливості успадкування різних типів даних один від одного. Також тоді була закладена ідея можливості інкапсуляції та керування доступом до коду того чи іншого типу даних у проекті. Завдяки створенню мови програмування Smalltalk у 1972 році, з’явилася ідея використання повністю об’єктно-орієнтованого коду проекту, де будь-який елемент програми буде представлений специфічним об’єктом. Окрім цього, великий вплив Smalltalk на розвиток ООП відзначився введенню ідеє поліморфізму як можливості зміни поведінки у класів, пов’язаних однією ієрархією успадкування. Однак максимальний внесок в популяризацію концепцій ООП зробила мова програмування C++, створена у 1983 році, яка поєднала потужність та популярність мови C із об’єктно-орієнтованими можливостями у вигляді інкапсуляції, успадкування, поліморфізму, абстрагування та можливістю використання шаблонів та патернів програмування задля уніфікації коду проекту та пришвидшення темпів його розробки. Створення мов програмування Java на початку 1990-х, а згодом і C# забезпечило можливість використання повністю об’єктно-орієнтованого коду проекту, а також забезпечило платформо-незалежне програмування. Окрім того, використання цих мов передбачало можливості автоматичного прибирання сміття з оперативної пам’яті, що суттєво знизило ризики виникнення витоків пам’яті під час роботи проектів та підвищило темп розробки. Слід також зазначити, що мова C#, створена у 2002 році, надавала розробникам можливості інтеграції запитів до баз даних безпосередньо в мову програмування, завдяки технології LINQ, а згодом, це було реалізовано у повністю об’єктно-орієнтований спосіб через використання бібліотеки розширювальних

методів. Сучасні високо-рівневі мови програмування, такі як Python та JavaScript, також оказали значний вплив на розвиток ООП через запровадження мульти-парадигмальності як можливості поєднання об’єктно-орієнтованих та функціональних парадигм розробки програмного забезпечення. Це дозволило спростити архітектурні рішення та суттєво прискорити темп розробки завдяки гнучкості проектної архітектури. Тому на теперішній час серед сучасних тенденцій ООП домінують ідеї поєднання функціонального програмування із базовими концепціями ООП, використання мікросервісної архітектури, а також концепцій візуального програмування та елементи штучного інтелекту (наприклад, Copilot). Однак найбільший вплив на сучасний стан ООП зробили принципи SOLID та принципи CUPID, які вважаються альтернативними між собою.

Принципи SOLID – це розширений набір концепцій ООП, сформований Робертом Мартіном у середині 2000-х років, який доповнює звичну інкапсуляцію, успадкування, поліморфізм та абстрагування новими ідеями. Ці принципи призвані допомогти розробникам створювати гнучке до змін та модифікацій, зрозуміле для великої команди розробки та легке у підтримці та масштабуванні програмне забезпечення. Абревіатура SOLID має пояснення кожної літери: S – single responsibility (принцип єдиної відповідальності), O – open-closed (принцип відкритості/закритості коду), L – Liskov substitution (принцип підстановки типів від Барбари Лісков), I – interface segregation (принцип розділення великих інтерфейсів) та D – dependency inversion (принцип інверсії залежностей між типами даних). Використання цих принципів дає розробникам можливість будувати легку, зрозумілу та гнучку до змін архітектуру програмних засобів при мінімізації виникнення помилок. Суть принципу єдиної відповідальності (літера S) полягає у тому, що кожен тип даних повинен мати лише одну причину свого існування, тобто він повинен реалізовувати тільки одну велику функцію, один тип – одна відповідальність за одну важливу функціональність проекту. Це значно спрощує процес тестування проекту та сприяє полегшенню

розширюваності коду проекту. Принцип відкритості / закритості типів (літера O) накладає певні обмеження можливостей на типи даних. Відповідно до цього тип має бути відкритим для розширення новим важливим функціоналом, але закритий для модифікації існуючого. Тобто нова функціональність має додаватися в типи через їх розширення, а не через зміну існуючого коду. Такий підхід значно пришвидшує процес тестування проекту, зменшує імовірність внесення помилок у існуючий код та спрощує додавання нових можливостей через механізм успадкування типів та використання методів розширення. Принцип підстановки Барбари Лісков (літера L) передбачає те, що зміна об’єктів дочірнього типу об’єктами базового типу не має призводити до виникнення помилок у роботі проекту. Відповідно до цього принципу, дочірні класи не повинні порушувати поведінку свого базового класу. Використання цього принципу під час розробки призводить до того, що код проекту стає більш передбачуваним та легким для розширення та масштабування. Принцип розділення великих інтерфейсів (літера I) затверджує, що для проекту корисніше мати багато невеликих специфічних за функціональністю інтерфейсів, які можна буде легко комбінувати у типах даних, аніж використовувати один великий інтерфейс, який буде важче реалізувати. Відповідно до цього, класи не повинні залежати від методів, які вони не використовують. Впровадження цього принципу призводить до полегшення підтримки та модифікації коду проекту та полегшенню його тестування. Принцип інверсії залежностей (літера D) декларує, що залежності між типами повинні будуватися на основі абстракцій (абстрактних класів та інтерфейсів), а не конкретних реалізацій. Цей принцип також має значний вплив на побудову архітектури проекту, оскільки відповідно до нього високо рівневі модулі не повинні залежати від низькорівневих модулів та конкретних типів даних. Використання цього принципу дозволяє розробникам створювати гнучку до змін архітектуру проекту завдяки зменшенню зв’язності компонентів у програмних засобах, а також підвищує швидкість їх тестування.

Альтернативою SOLID є принципи CUPID від Майкла Фізерса. Як і SOLID, CUPID представляє собою абревіатуру з нових концепцій ООП, які фокусуються на створенні зрозумілого, чистого та дружнього до розробників різного технічного рівня коду проекту. Абревіатура CUPID також надає пояснення кожної літери: C – composable (принцип скомпонованості), U – unix philosophy (принцип відповідності філософії Unix), P – predictable (принцип передбачуваної поведінки типів даних), I – idiomatic (принцип відповідності ідіомам ООП), та D – domain-based (принцип організації коду на ідеях доменної логіки). Суть принципу скомпонованості (літера C) полягає у тому, що код повинен бути легким для поєднання та повторного використання, його компоненти мають бути максимально незалежними один від одного та повинні мати простий інтерфейс для забезпечення взаємодії між собою. В рамках цього принципу визначають, що велика кількість малих, чітко визначених частин системи значно краще, ніж використання великих супер-типів з великою кількістю функціональності. Відповідно до цього, важливою визначається можливість легкого комбінування функцій або класів без їх значної модифікації для реалізації критичної функціональності проекту. Принцип використання філософії Unix (літера U) полягає у мінімізації відповідальностей в типах та функціях, тобто програмні елементи повинні роботи одну річ і робити її швидко і без помилок. Цей принцип фокусується на мінімізації залежностей між типами та функціями та ізоляції їх відповідальностей. Відповідно до цього, кожен модуль або функція у проекті повинні мати своє чітке практичне призначення. Принцип передбачуваності поведінки коду (літера P) передбачає, що код проекту завжди має бути легким у читанні та працювати виключно так, як це очікується. У практичному сенсі це накладає певні обмеження на пере випуск віртуальних та абстрактних методів у класах-спадкоємцях. Окрім того, впровадження цього принципу передбачає використання чітких угод про іменування методів та прогнозованих результатів їх роботи. Принцип ідіоматичності коду (літера I) передбачає, що

код проекту завжди повинен відповідати популярним та поширеним практикам та стандартам обраної мови програмування. Відповідно до цього на розробника накладається вимога щодо використання універсальних патернів та технік, які є простими у читанні та розумінні, а також є добре відомими у спільноті розробників. Також в рамках цього принципу декларують, що легкість читання коду важливіша за його оптимальність та швидкодію у більшості випадків. Принцип використання доменної логіки побудови проекту (літера D) передбачає фокусування уваги на бізнес-логіці та доменних моделях. Відповідно до цього код проекту повинен відображати реальні бізнес правила та моделювати процеси реального світу та ринку. При цьому велику увагу приділяють відокремленню технічних деталей від бізнес-логік програмного засобу.

Слід зазначити, що і використання SOLID, і використання CUPID має як свої переваги, так і свої недоліки. Якщо порівняти ці концепції між собою, то можна визначити, що принципи SOLID зосереджують увагу розробника на дизайні класів, в той час, як принципи CUPID – на поведінці системи. До того ж SOLID є більш об’єктно-орієнтовним, в той час, як ідеї CUPID передбачають поєднання класичного ООП із можливостями функціонального програмування. Окрім того, SOLID-проекти зазвичай характеризуються використанням великої кількості високорівневих абстракцій, в той час як CUPID-проекти фокусуються на практичності та реальних сценаріях використання різних типів даних під час вирішення тієї чи іншої практичної задачі. При цьому зазвичай відмічають, що використання SOLID може призвести до зростання складності проекту, особливо для людей без великого технічного досвіду роботи, в той час як CUPID вважають більш інтуїтивним та простим для розуміння всіх розробників, незалежно від їх кваліфікації.

Незважаючи на це, на сьогодні немає чітких рекомендацій щодо використання SOLID або CUPID для розробки специфічних типів проектів, тому під час розробки слід самостійно обирати ті принципи, які можуть принести максимальної користі у кожному конкретному випадку.

2.3 Особливості багатошарових архітектур програмних засобів

Термін архітектура програмного засобу визначають як багаторівневий дизайн проекту, що визначає структуру його компонентів. Цей дизайн визначає перелік компонентів та їхньої взаємодії між собою, а також описує особливості використання того чи іншого архітектурного патерну, який врегульовує особливості взаємодії компонентів проекту між собою. Архітектура визначає спосіб реалізації вимог до функціональності проекту, його продуктивності та надійності. Грамотно спроектована архітектура полегшує розробку та тестування проекту, його масштабування та модифікацію і підтримку.

Відсутність архітектури у програмному засобі може мати значні наслідки, які впливають на швидкість розробки, тестування, якість та кінцеву вартість проекту. Одним з головних недоліків відсутності архітектури є погана підтримка та масштабованість через те, що зміна будь-яка частини коду проекту може спричинити непередбачувані наслідки в його інших місцях. Це призводить до зростання складності додавання або модифікації функціональних можливостей проекту. Відсутність архітектури суттєво підвищує складність тестування через неможливість ізольованого тестування окремих компонентів проекту. Окрім того, відсутність архітектури часто призводить до підвищення рівня зв’язаності компонентів проекту, що ставить їх у залежність один від одного та ускладнює їх повторне використання та модифікацію. Через це спроба заміни або видалення одного модуля в проекті призводить до необхідності додаткових змін у великій кількості інших частин проекту. Відсутність чіткої архітектури у програмному засобі призводить до суттєвого зросту складності коду та погіршує його якість через труднощі у розумінні з боку інших розробників, залучених у проект, що в підсумку призводить до значного збільшення часу розробки та вартості програмного засобу. Слід зазначити, що проект, створений без використання архітектури дуже важко підтримувати, що накопичує технічний борг, а також

перешкоджає повторному використанню окремих частин та компонентів проекту та часто унеможливлює інтеграцію із сторонніми бібліотеками, системами та сервісами. В результаті цього код проекту стає закритим і не гнучким до модифікації та оптимізації. В свою чергу це негативно впливає на якість та конкурентоздатність проекту.

Використання програмної архітектури позбавляє розробку програмного засобу цих недоліків та зазвичай призводить до чіткого розділення відповідальностей між модулями та компонентами, зменшення зв’язаності компонентів, полегшення підтримки та масштабованості проекту. Також це пришвидшує процес тестування та підвищує якість проекту. Слід зазначити, що само по собі використання тієї чи іншої архітектури не гарантує успіх програмного засобу на ринку та може мати негативні впливи на процес розробки. Некоректне використання архітектурного патерну або формування занадто складної архітектури призводить до значного підвищення складності розробки та підвищення вимог до технічного рівня розробників. В свою чергу це призводить до збільшення витрат на розробку. Окрім того, некоректне використання того чи іншого архітектурного патерну може призвести до формування непотрібних абстракцій, які суттєво ускладнюють розуміння проекту та призводять до зростання кількості інфраструктурного коду, може негативно впливати на темп розробки та якість тестування. Однак, слід зазначити, що присутність архітектури у програмному засобі зазвичай приносить більше переваг, ніж недоліків, особливо для великих за кількістю функціональності та довгострокових проектів. При цьому вибір архітектури, яка буде відповідати розміру проекту, його складності та цілям реалізації є ключовим для забезпечення швидкості, вартості та якості його розробки, розгортання та підтримки.

Багатошарові архітектури програмних засобів почали формуватися на початку та в середині 1990-х років у відповідь на потребу в побудові більш структурованих, масштабованих та гнучких програмних засобів. На їх розвиток мали суттєвий вплив еволюція обчислювальної техніки, мережевих

технологій та домінуючих концепцій щодо написання програмного коду. Ключовими етапами у розвитку багатошарової архітектури стала поява концепції клієнт-серверної архітектури, формування тришарових архітектур, їхня подальша еволюція завдяки широкому розповсюдженню Інтернету та формування концепції мікросервісної архітектури. Формування ідеї клієнт-серверної архітектури відбувалося на початку 1990-х років, вона передбачала розділення проекту на клієнтську та серверну частини, де клієнт виконував роль інтерфейсу користувача, а сервер обробляв клієнтські запити. Концепція тришарової архітектури передбачає розподіл функціональності проекту на три основних рівня: рівень інтерфейсу користувача, рівень бізнес-логіки, та рівень-посередник між інтерфейсом користувача та бізнес-логікою. В залежності від виду патерну ці рівні можуть бути ізольованими один від одного як в патерні MVVM, або відкриті до взаємодії між собою, як у патерні MVC. Слід зазначити, що початок 2000-х років характеризувався бурхливим розвитком саме трирівневих архітектурних патернів, саме тоді було запропоновано і патерн MVP і поширений на сьогодні патерн MVVM. Наступне покоління архітектурних патернів зосередилися на ідеях «чистого коду» та «чистої архітектури», а також поширенні мікросерсіної архітектури, яка продовжила ідею багатошаровості, але з акцентом на модульність та гнучкість окремих систем-сервісів. Відповідно до цього велике поширення набули технології контейнеризації (Docker та Kubernetics), а також хмарних та розподілених обчислень (AWS, Azure), що зробило багатошарові патерни ще більш динамічними. Однак слід зазначити, що складність сучасних патернів суттєво зросла, що робить недоцільними їх використання під час розробки невеликих дослідницьких проектів.

При розгляді дворівневих архітектур зазвичай говорять о найбільш поширеній клієнт-серверній архітектурі яка дуже швидко стала домінуючою на ринку розробки програмних засобів. Але у випадку, якщо у проекті не передбачається використання мережі та виділення окремої серверної частини, дворівневі патерни також можуть використовуватися і підчас

розробки клієнтських проектів або single-page проектів. В такому випадку код проекту розділяється між рівнем, View який обслуговує потреби інтерфейсу користувача та рівнем Model, який представляє всю бізнес-логіку проекту та реалізує всю функціональність, не пов’язану безпосередньо із інтерфейсом користувача. Відповідно до цього ревінь View відправляє рівню Model запити виконання тієї чи іншої операції, яка призводить до зміни даних, після чого рівень Model валідує або не валдує цей запит, виконує його, змінює свої дані та надсилає свої змінені дані на рівень View для відображення змін користувачеві. В залежності від вибору мови програмування, ця взаємодія може бути або автоматизованою за рахунок використання патерну Observer або івентів, чи виконуватися старим процедурним способом. Головними перевагами використання таких дворівневих патернів є їх технічна простота, яка забезпечує високий темп розробки при мінімальних фінансових затратах завдяки можливості залученню до проекту розробників, які не мають великого досвіду та високої технічної кваліфікації. Недоліками такого підходу зазвичай є складнощі під час масштабування проекту, велика зв’язаність модулів в рамках того чи іншого рівня архітектури та вразливість проекту під час його модифікації.

Поява трирівневих архітектурних патернів стала логічним результатом еволюційного розвитку дворівневих патернів. Незалежно від вибору конкретного типу патерну зазвичай три рівневі патерни мають розподіл на рівень Model або Data Layer, який представляють дані проекту та методи щодо їх редагування, файлового обміну та оновлення, презентаційний рівень View, який забезпечує можливості взаємодії із користувачем, та рівень бізнес логіки, який в залежності від виду конкретного патерну має назву Controller, Presenter або ViewModel. Перевагами такого архітектурного підходу є чіткий розподіл функціональної відповідальності між рівнями проекту, легка масштабованість та забезпечення високого темпу розробки і тестування проекту. Серед недоліків відзначають підвищення технічної складності в реалізації, внаслідок чого виникає необхідність залучення до розробки

проекту спеціалістів з велики технічним досвідом та високим професійним рівнем, що підвищує його вартість.

Архітектурний патерн Model-View-Controller – MVC на сьогодні є одним з найбільш поширених видів патернів, незважаючи на те, що він був створений ще наприкінці 1980-х років і став основою для створення всіх трирівневих патернів. Рівень Model відповідає за управління даними та бізнес-правилами проекту. Він інкапсулює дані та методи їх обробки та оновлення, оновлюється на основі дій, ініційованих контролером та повідомляє усі інші рівні про свої зміни в різний спосіб, в залежності від обраної модифікації.

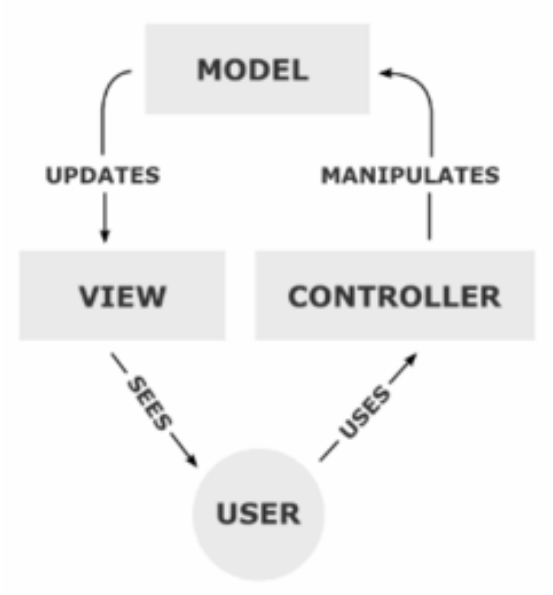

Рисунок 2.2 – Загальна схема роботи патерну MVC

Ревінь View відповідає за візуалізацію даних та взаємодію із користувачем. Він отримує дані безпосередньо від моделі, реагує на її зміни та оновлює відповідно до них інтерфейс користувача. Важливо особливістю рівня View є те, що він не може містити бізнес-логіки проекту, а зберігає лише логіку відображення даних користувачам. Рівень Controller відповідає за валідацію вводу користувача та взаємодію між представленням та рівнем моделі. Під валідацією дій користувача розуміють перевірку введених даних,

перевірку можливості виконання тієї чи іншої команди моделі в залежності від поточного стану програми та наявності необхідних дозволів та коректних даних у користувача. Він приймає запити на від рівня View, перевіряє можливість виклику методів моделі із отриманими даними користувача, запитує виконання цих методів та оновлює рівень View результатами їх виконання. Таким чином рівень View не може безпосередньо запитувати на виконання ті чи інші команди з рівня Model. Перевагами використання цієї архітектури визначають чіткий розподіл рівнів відповідальності між різними рівнями архітектури проекту, що полегшує процес розробки та модифікації проекту, можливість повторного використання компонентів на різних рівнях архітектури, підвищення зручності масштабування та тривалої підтримки проекту. Головними недоліками цього архітектурного патерну визначають збільшення кількості інфраструктурного коду за рахунок виділення додаткових абстракцій, висока імовірність перевантаження контролеру надмірною кількістю функціональності у випадку якщо логіка не розподілена правильно, висока імовірність виникнення залежностей між компонентами в рамках кожного з рівнів архітектури та велика кількість файлів проекту, яка ускладнює проект розширення команди розробки. Але незважаючи на це MVC залишається одним з найпоширеніших патернів завдяки своїй простоті та зручності у використанні.

Архітектурний патерн Model-View-Presenter – MVP з’явився в результаті еволюції патерну MVC. Його головна особливість полягає у більшому відокремленні логіки представлення від бізнес-логіки проекту, що полегшує тестування та підтримку. Слід зазначити, що цей патерн активно використовується у розробці мобільних додатків різної складності. Відповідальності рівня Model цього патерну аналогічні відповідальностям Model патерні MVC. Але важливою відмінністю цього рівня в MVP є те, що він повністю ізольований від рівня View, тобто будь-яка взаємодія між цими рівнями архітектури проекту виключена як така. Рівень View відповідає за візуалізацію даних користувача та забезпечує із ним взаємодію. Окрім того,

він не містить власної бізнес-логіки, а лише викликає методи презентера. Через таку особливість цей рівень в MVP часто називають пасивним, оскільки він лише відображає дані користувачу та транслює дії користувача на рівень Presenter.

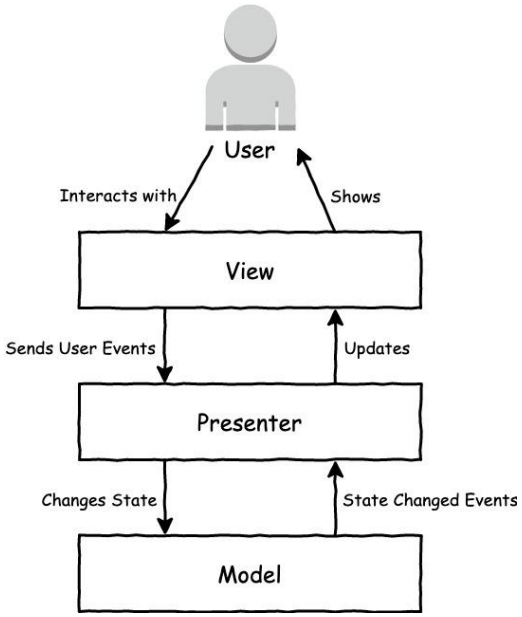

Рисунок 2.3 – Загальна схема роботи патерну MVP

Рівень презентації Presenter є посередником між моделлю та рівнем View, він отримує дані від моделі або результати виконання її методів та передає їх рівню View для відображення користувачу. Окрім того, він обробляє події користувача, які надходять з рівня View. Особливістю рівня Presenter є те, що він не залежить від платформи відображення (web-, мобільний пристрій чи desktop) та концентрує в собі всю бізнес-логіку проекту, залишаючи моделі лише методи обробки даних. Серед переваг MVP визначають спрощення рівня представлення, що позитивно впливає на темп розробки, спрощення можливостей тестування та повторного використання та підвищення гнучкості у розробці та можливостях модифікації проекту. Серед недоліків MVP відзначають підвищення складності в реалізації, порівняно із патерном MVC та збільшення кількості інфраструктурного коду.

Архітектурний патерн Model-View-ViewModel – MVVM був розроблений для спрощення та розробки рівня інтерфейсів користувача. В свою чергу він є еволюцією патерну MVP і концентрується на автоматизації процесів обміну даними між всіма рівнями архітектури завдяки використанню технології двонаправленого зв’язування даних Data Binding.

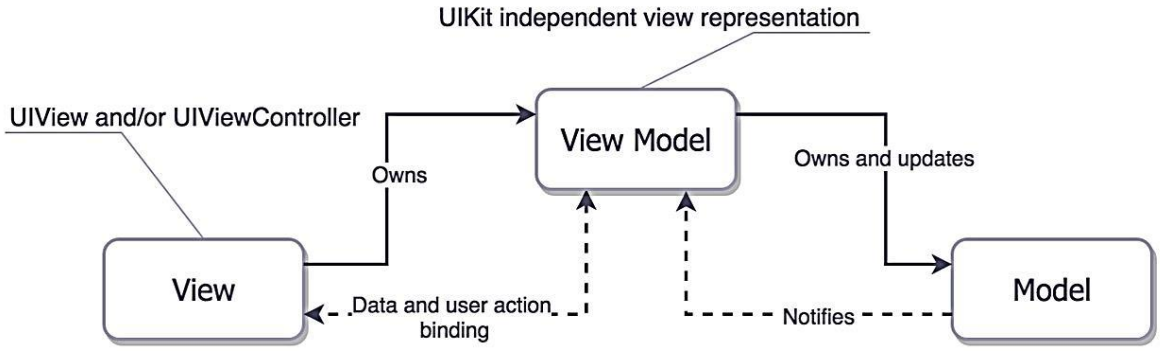

Рисунок 2.4 – Загальна схема роботи патерну MVVM

Рівень Model цього патерну інкапсулює дані та бізнес-логіку та відповідає за управління станом даних і виконання ключових операцій щодо їх модифікації так само, як і в патерні MVP. Важливою відмінністю MVVM від MVP та MVC на рівні Model є те, що цей рівень існує сам по собі і нічого не знає про рівні View та ViewModel, а представляє лише інтерфейс до даних на читання та методів щодо їх обробки. Рівень представлення View у MVVM відповідає за відображення даних та організацію взаємодії із користувачем так само, як і в патерні MVP, але є повністю пасивним компонентом проекту, оскільки транслює запити на рівень ViewModel та отримує від нього дані на відображення. Головною особливістю цього рівня, яка відрізняє його від View у MVP є використання техніки прив’язування даних Data Bindings, яка забезпечує автоматичне оновлення інтерфейсу при зміні даних у ViewModel. Рівень ViewModel виконує роль посередника між іншими рівнями MVVM та відповідає за підготовку даних із Model для їх подальшого відображення на рівні View. Головною особливістю ViewModel, яка відрізняє його від рівня

Presenter у MVP, є використання специфічної техніки властивостей зі спостереженням – Observable Properties, яка автоматизує відстеження змін в даних та інформування рівня View про необхідність оновлення. Головними перевагами MVVM перед іншими поширеними трирівневими архітектурними патернами є автоматизація обігу та валідації даних завдяки технікам Data Bindings та Observable Properties на різних рівнях архітектури, а також ізольованість всіх рівнів між собою. Це призводить до підвищення темпу розробки проекту та спрощення процесу масштабування та портування проекту на різні платформи завдяки тому, що навіть повна зміна рівня View не впливає на інші рівні проекту. Головним недоліком цього MVVM є підвищення технічної складності його реалізації та збільшення кількості інфраструктурного коду проекту, порівняно із патернами MVC та MVP. Але незважаючи на це, MVVM на сьогодні є одним з найпопулярніших патернів як для desktop-, так і для мобільних проектів.

2.4 Теорія штучних імунних систем та імунні моделі

Штучні імунні системи – Artificial Immune Systems – специфічний різновид обчислювальних моделей штучного інтелекту, які використовують у своїй роботі механізми та принципи роботи біологічної імунної системи живих організмів, в першу чергу, людини. На сьогоднішній день ці системи поки не користуються такою увагою, як генеративні штучні нейронні мережі, але мають значний науковий потенціал. Вони зазвичай використовуються для вирішення різних завдань у галузі інформатики, інженерії, аналізу даних, кібербезпеки, розпізнавання образів, тощо.

Головними властивостями штучних імунних систем, які можуть бути використані під час вирішення різноманітних технічних задач є можливість розпізнавання елементів системи та чужорідних зовнішніх об’єктів та можливість вивчення їх структури, можливість організації асоціативної імунної пам’яті для зберігання інформації про досліджені об’єкти, здатність

використовувати механізми саморегуляції задля підтримання балансу активних елементів системи та об’єктів, які створюються імунну пам’ять та можливість адаптації елементів системи під зміни зовнішнього простору або властивостей об’єкту спостереження. Відповідно до цього на сьогодні існує декілька базових моделей та алгоритмів, які дозволяють моделювати, налаштовувати та використовувати для вирішення різних практичних завдань на основі основних властивостей поведінки штучних імунних систем. Такими імунними алгоритмами та моделями є:

- модель негативного відбору – Neagive Selction Algorithm, яка концентрує увагу дослідників на відтворенні механізму відбору імунних об’єктів, з яких формується імунна система та використовується для виявлення різноманітних аномалій, які можуть виникати у об’єкті спостереження впродовж тривалого часу;

- модель клонального відбору – Clonal Selection Algorithm, яка фокусує увагу дослідників на процесі генерації імунних об’єктів системою, їхньої мутації та саморегуляції системи, що може використовуватися у задачах оптимізації пошуку рішення серед множини можливих варіантів;

- модель штучної імунної мережі – Artificial Immune Algorithm, яка ґрунтується на особливостях поведінки імунних об’єктів та контролює їх поведінку під час вирішення складних задач класифікації, кластеризації, розпізнавання образів, задач DataMining та задач прогнозування;

- модель визначення алергенів – Danger Theory, яка використовує можливості системи щодо реакції на виявлення чужорідних об’єктів та застосовується на практиці для визначення ризиків та пошуку аномалій в об’єктах дослідження.

Слід зазначити, що найбільшим дослідницьким інтересом користуються саме модель клонального відбору та модель штучної імунної мережі, оскільки вони мають широкий спектр застосування та можливості щодо модифікації та гібридизації, тобто додавання в поведінку штучної імунної системи елементів, притаманних іншим принципам організації

систем штучного інтелекту, таким як штучні нейронні мережі, алгоритми рою та мурашнику та алгоритми нечіткої логіки.

Подібно до більшості систем інтелектуальної обробки інформації та прийняття рішень, теорія штучних імунних систем передбачає використання специфічного термінологічного апарату, який включає поняття лімфоцитів, антигенів, фагоцитів, афінностей, авідностей, поняття клонування, мутації, презентації антигенів та антитіл, супресія та апоптоз. Серед цих термінів можна виділити ключові поняття антитіл, антигенів та афінностей між ними. Під антитілом в ШІС розуміють активний об'єкт, який є частиною системи та використовується нею для вирішення різних задач, антигеном називають будь-який чужорідний до ШІС об'єкт, який необхідно дослідити, а афінность використовується як показник подібності між антитілами та антигенами. Також, в залежності від вибору моделі ШІС афінность може використовуватися як міра подібності між антитілами самої імунної системи. Слід зазначити, що значення цього показника знаходиться в межах (0.0; 1.0] та обчислюється наступним чином:

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

де d_{ij} – евклідова відстань між ознаками і-го та j-го імунних об'єктів у багатовимірному просторі ознак.

Модель клонального відбору є однією за найперших моделей ШІС, яка була описана ще на початку 1970-х років М. Барнетом та Н. Джерне, які запропонували телію клональної селекції як принцип саморегуляції штучної імунної мережі. Цей принцип використовує особливості роботи лімфоцитів – антитіл, які активуються після факту виявлення імунної системою присутності чужорідних об'єктів – антитіл. Він передбачає внесення специфічних змін в набор (популяцію) антитіл шляхом їх клонування, зміни параметрів клонів за рахунок мутації, та саморегуляції популяції антитіл за рахунок відбору клонів, які характеризуються найкращими показниками

афінностей до антигенів, а також механізмом старіння антитіл попередньої популяції антитіл. Слід зазначити, що об’єкти ШІС, незалежно від того, яка саме модель використовується, не можуть вносити зміни в набор виявлених чужорідних об’єктів – антигенів, а можуть тільки вивчати їх ознаки. За рахунок цього механізму поступових змін в популяції антитіл досягається їх специфічність до антигенів. При цьому під специфічністю розуміють такий стан ШІС, коли вона має хоча б одне антитіло із максимальною афінністю до визначеного антигена. Принцип роботи моделі клонального відбору можна розділити на декілька ключових етапів, описаних нижче.

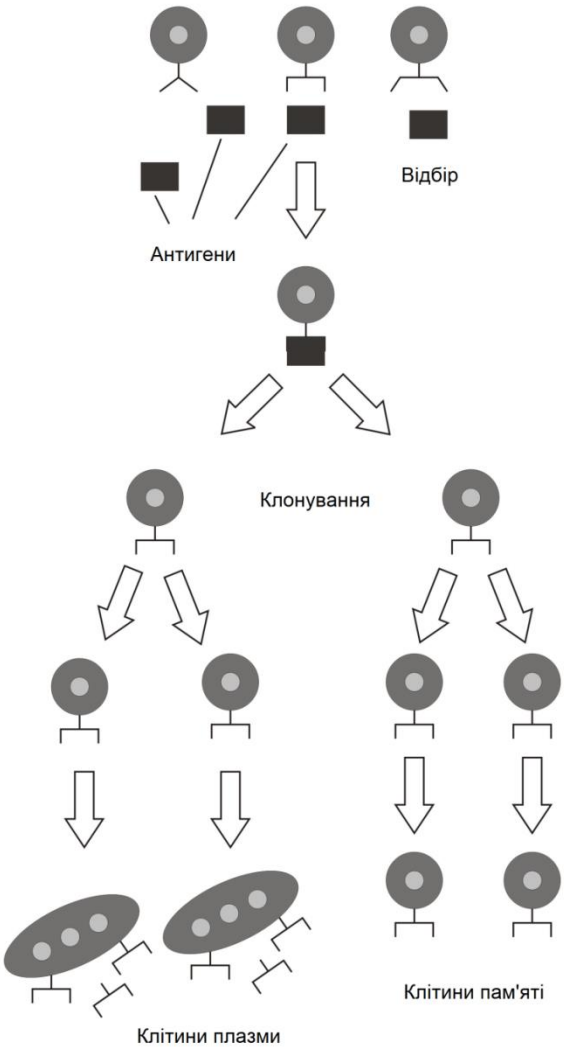

Рисунок 2.5 – Принцип роботи моделі клонального відбору

Першим етапом роботи моделі клонального відбору, реалізованого в алгоритмі ClonAlg, є презентація набору антигенів до популяції антитіл. Під час цієї презентації відбувається визначення афінностей між кожним антитілом системи та кожним зовнішнім антигеном. Після цього починається процес клонування антитіл, в результаті чого для кожного антитіла формується певна кількість клонів, яка визначається заздалегідь або завдяки використанню деякого більш специфічного підходу. Слід зазначити, що клони, які створюються під час мутації повністю повторюють знаки антитіл, від яких вони створюються. Окрім того, в залежності від модифікації алгоритму ClonAlg етапу клонування передує етап первинного відбору антитіл під час якого з популяції антитіл відбираються об’єкти, які характеризуються максимальним показником афінності до популяції антигенів або до деякого окремого антигену х цієї популяції. Після клонування антитіл розпочинається мутація сформованої популяції клонів. процес мутації полягає у змінах деяких ознак клонів відповідно до обраного способу мутації. Зміненим в процесі мутації клонам презентуються антигени, під час чого для кожного клону відбувається визначення афінностей до кожного з антигенів. Після цього розпочинається етап саморегуляції імунної системи завдяки редагуванню популяції клонів та антитіл та формуванню об’єктів імунної пам’яті. Під час саморегуляції відбувається співставлення клонованих антитіл із клонами та відбір тих об’єктів, які характеризуються найкращими показниками афінностей до популяції антигенів, або до якогось окремого обраного антигену. Окрім того, деяка частина імунних об’єктів, яка набула стану специфічності до деякого антигену, поповнює набір антитіл, з яких формується пам’ять ШІС. Після саморегуляції популяції антитіл та формування імунної пам’яті відбувається перевірка можливості зупинки алгоритму ClonAlg завдяки використанню деякого визначеного заздалегідь специфічного критерію зупинки. У випадку, якщо мережа антитіл не задовольняє цьому критерію зупинки, система переходить до початку роботи

та розпочинає клонування відредагованої популяції імунних об’єктів та цикл її адаптації повторюється новою ітерацією знову.

Задача керування поведінкою ігрових ботів в RPG-грі може розглядатися як специфічний випадок вирішення задачі класифікації, коли з переліку варіантів поведінки бота формується набор класів, що представляється популяцією антигенів. При цьому ознаками класу є декілька параметрів, які мають специфічні незмінні значення. Ці параметри беруться з обмеженого переліку характеристик, які визначають поведінку боту та враховують його основні ознаки та властивості у RPG-грі. Так шляхом клонування, мутації, взаємодії з іншими персонажами та супресії мережі для кожного окремого ігрового боту ШІС обирає специфічний та оптимальний для поточного стану виконання програми тип поведінки.

2.5 Ігровий дизайн проекту

Ігровий додаток моделює футуристичний світ у стилі відкритий космос, де гравець керує групою космічних апаратів, кількістю від 1 до 6 одиниць, яка змагається із іншою такою групою подібних космічних апаратів аналогічної кількості. Принцип ігрового бою запозичений з популярної у середині двотисячних років RPG-гри Disciples II, але із суттєво спрощеним ігровим дизайном. У гри передбачається декілька ігрових фракцій, які можуть випускати свої специфічні типи космічних апаратів із особливими можливостями та властивостями. При цьому кожен космічний корабель у команді користувача або команді його супротивника має перелік специфічних характеристик, яка визначають його можливості:

- вартість – характеристика, яка визначає кількість ігрових грошей, які користувач має витратити на придбання цього корабля;
- рівень корабля – характеристика, яка є одним з головних показників корабля, який збільшує інші параметри корабля;
- бойовий досвід – характеристика, яка визначає одиниці досвіду,

отримані у боях із кораблями супротивника, які визначають рівень корабля;

- міцність – характеристика, яка визначає життєздатність корабля у бою із кораблями супротивника;
- енергія – характеристика, яка визначає коефіцієнт підвищення атаки або захисту, в залежності від вибору гравця;
- атака – характеристика, яка визначає рівень ушкодження, що завдає корабель користувача обраному кораблю супротивника в бою, яке впливає безпосередньо на його захист та міцність;
- захист – характеристика, яка визначає броню, або додатковий рівень захисту, який зменшує кількість одиниць атаки, яку корабель супротивника наносить по цьому космічному кораблю;
- швидкість – характеристика, яка визначає пріоритетність при формуванні черги ходів космічних кораблів під час бою;
- відновлення – характеристика, яка визначає кількість одиниць міцності корабля, які корабель може відновлювати замість виконання атаки по кораблю супротивника.

Слід зазначити, що у грі передбачається декілька фракцій, які створюють свої типи космічних кораблів та наділяють їх специфічними властивостями. Для простоти реалізації, фракції умовно позначаються кольорами, а саме: жовті, сіни, червоні. Головною особливістю кораблів жовтої фракції є збільшення показника міцності на 10%, порівняно із кораблями інших фракцій, особливістю кораблів синьої фракції є збільшення показника захисту на 10%, порівняно із кораблями інших фракцій, а головною особливістю коралів червоної фракції є збільшення показника відновлення на 10%, порівняно із кораблями інших фракцій. Завдяки цьому кожна може унікальним способом підвищувати значимі характеристики свої кораблів порівняно із кораблями інших фракцій.

Окрім того, кожен корабель, як з боку команди користувача, так і з боку команди його супротивника, може належати до одного з рангів. Ці

ранги оказують суттєвий вплив на вартість та параметри цього корабля. Перелік основних рангів визначений далі:

- support – корабель підтримки, займає 1 місце в групі кораблів, має 1 спосіб атаки ближнього бою та 3 способи відновлення;
- scout – корабель розвідки, займає 1 місце в групи кораблів, має 1 спосіб атаки ближньої та 1 спосіб атаки дальньої атаки, а також має 1 спосіб відновлення, відрізняється серед інших кораблів своєю швидкістю;
- cruiser – базовий бойовий корабель, займає 1 місце в групі кораблів, має 2 способи ближньої та 1 спосіб дальньої атаки, також має 1 спосіб відновлення;
- frigate – підсилений базовий бойовий корабель, займає 1 місце в групі кораблів, має 2 способи ближньої та 2 способи дальньої атаки, має 1 спосіб відновлення, відрізняється серед інших кораблів підсиленим захистом;
- destroyer – важкий лінійний бойовий корабель, займає 2 місця в групі кораблів, має 2 способи ближньої та 2 способи дальньої атаки, має 2 способи відновлення, відрізняється серед інших кораблів підсиленим показником міцності;
- dreadnought – надмірно важкий бойовий корабель, займає 2 місця в групі кораблів, має 3 способи ближньої та 3 способи дальньої атаки, має 3 способи відновлення, відрізняється серед інших кораблів підсиленими показниками міцності та захисту, а також зменшеним показником швидкості.

Для опису можливостей кораблів використовувалися різі варіанти атаки або відновлення, слід зосередити увагу на особливостях їх дії з точки зору ігрового дизайну. По-перше, в описі можливостей кораблів було використано два види атаки: ближня та дальня атаки. При цьому ближня атака передбачає можливість корабля атакувати тільки ті кораблі супротивника, які стоять на першої лінії ігрового поля битви до нього, тобто на 2, 4 та 6 позиціях карти ігрового бою. Нанесення дальньої атаки передбачає можливість атакувати кораблі супротивника, які знаходяться на другій лінії у команді супротивника, тобто на 1, 3 та 5 позиціях карти

ігрового бою. Слід зазначити, що у описі можливостей зазначених кораблів було використано три способи атаки, при цьому перший спосіб передбачає концентровану атаку одного обраного корабля супротивника, другий спосіб передбачає розподілену атаку всіх кораблів супротивника, які знаходяться на карті ігрового бою, третій спосіб атаки передбачає зменшення показника енергії обраного корабля супротивника на 30% від поточного значення. Окрім того, при описі можливостей ігрових кораблів було визначено два способи відновлення, а саме – перший спосіб відновлення полягає у тому, що корабель замість атаки починає ремонт себе, в процесі чого відновлює певну визначену кількість одиниць міцності, яку він втратив після атаки кораблів команди супротивника. Другий спосіб відновлення полягає у тому, що корабель замість атаки проводить ремонт одного будь-якого обраного існуючого корабля своєї групи, передаючи йому певну відзначену кількість одиниць міцності. Третій спосіб відновлення полягає у тому, що корабель замість атаки починає ремонт всіх існуючих кораблів своєї групи, розподіляючи порівну між ними певну відзначену кількість одиниць міцності. Слід зазначити, що у випадку повного вичерпання одиниць міцності в ході ігрового бою, корабель перестає існувати та його не можливо відновити в бою шляхом виконання відновлення другого чи третього способів. Відновлення такого корабля відбувається автоматично по закінченню бою.

Значний вплив на ігровий процес має визначення способу нанесення атаки між кораблями. Відповідно до цього, а також враховуючи визначені типи ближньої та бальної атаки, слід описати вигляд ігрового поля для проведення битви між командами кораблів. Все ігрове поле умовно ділиться на 3 частини, де першу частину займає команда кораблів першого гравця, друга частина розділяє команди між собою, а третю частину займає команда другого гравця. Слід зазначити, що при цьому в кожній команді кораблі вишикувані у 2 ряди – ближній до ворога та дальній від ворога, і тільки кораблі, які займають 2 місця на карті ігрового бою займають обидва ряди,

тобто можуть бити атаковані і ближньою і дальньою атаками. Окрім того, у випадку знищення всіх кораблів першої лінії, всі кораблі, які знаходяться на другій лінії можуть бути атаковані в будь-який спосіб атаки.

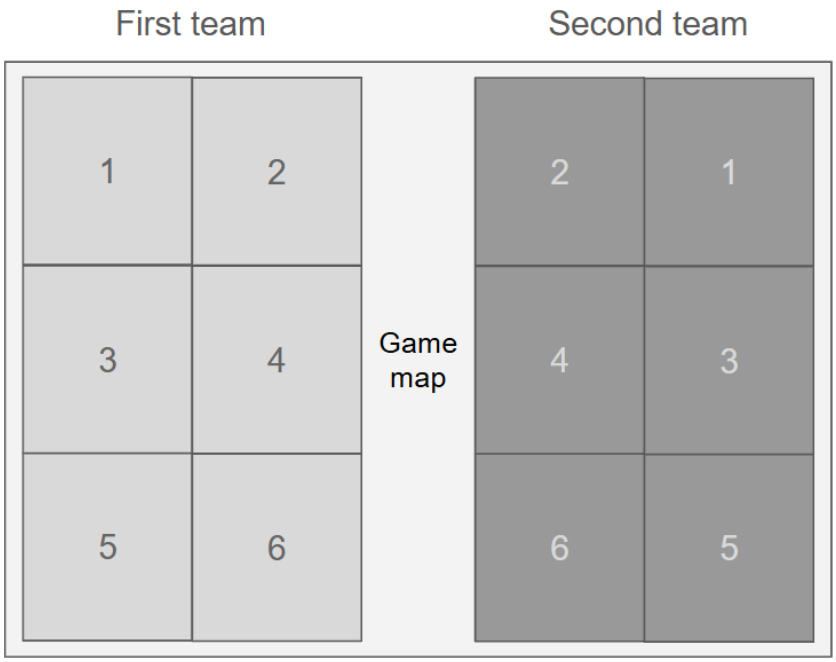

Рисунок 2.6 – Загальний вигляд карти ігрового бою

В рамках ігри великий вплив по можливості корабля має його рівень, а саме, він збільшує показники міцності, захисту та атаки на 1% від початкових значень корабля визначеного типу 1 рівня, враховуючи особливості фракції. Підвищення рівня відбувається за рахунок перемоги над командою супротивника, де за кожні 100 одиниць міцності кожного знищеного корабля команди супротивника нараховується 10 одиниць досвіду, який рівномірно розподіляється між всіма кораблями, які брали участь у битві. Окрім того, важливий вплив на ігровий процес має спосіб нанесення урону від атаки між кораблями, який визначається у формулі нижче:

$$damage = \frac{e_1}{e_2} \times (A_1 - D_2), \tag{2.2}$$

де $damage$ – значення скорегованого урону, який наносить корабель іншому в процесі атаки; A_1 – значення атаки корабля 1; e_1 – значення енергії корабля 1, D_2 – значення захисту корабля 2; e_2 – значення енергії корабля 2.

Відповідно до цього, показник міцності корабля, який атакують зменшується на значення $damage$ у випадку, якщо воно має позитивне значення. У випадку, якщо воно має негативне значення – показник міцності корабля, який знаходить під атакою не змінюється, тобто корабель взагалі не отримує пошкоджень.

Окрім того, важливий вплив на ігровий процес має спосіб відновлення ігрового корабля, який можна виконати під час ходу замість виконання атаки. Цей показник визначається наступним чином:

$$rp = R \times ke, \tag{2.3}$$

де rp – значення віднов ленної міцності корабля; R – характеристика відновлення корабля; e – значення енергії корабля, а k – коефіцієнт відновлення, якій відбирається випадковим чином з діапазону [1,1; 1.25].

Важливою особливістю ігрового процесу є можливість гри у режимі PvE – Player via Enemy, тобто така гра, коли користувач грає проти команди, якою керує сама гра. Відповідно до цього, для керування кораблями команди системи у режимі PvE було адаптовано модифікований та адаптований під специфічні умови задач алгоритм ШІС клонального відбору ClonAlg. В такому разі назвою ігровий бот можна охарактеризуватися корабель команди противника гравця, який користується грою у режимі PvE. Команда кораблів, якою керує система в бою проти команди користувача буде складатися з набору ігрових ботів, кожним з яких керуватиме модифікований алгоритм клонального відбору, який моделює штучну імунну систему.

3 АДАПТАЦІЯ ІМУННОЇ МОДЕЛІ КЛОНАЛЬНОГО ВІДБОРУ ДЛЯ КЕРУВАННЯ ІГРОВИМИ БОТАМИ ТА АРХІТЕКТУРА ПРОЕКТУ

3.1 Організація керування ігровими ботами на основі моделі клонального відбору

Задача керування ігровими ботами у бою із командою кораблів користувача зводиться до моделювання поведінки кожного окремого корабля команди супротивника з урахуванням можливостей та характеристик кожного конкретного корабля команди гравця. Відповідно до цього ігровими ботами є тільки кораблі команди гри/системи із якими бореться команда користувача. Таким чином, керування ігровими на основі ШІС, а саме імунної моделі клонального відбору, полягає у розгляді ботів як антитіл системи, які взаємодіють із чужорідними для системи антигенами, якими для ботів є кораблі з команди користувача проекту. Окрім того, боти можуть взаємодіяти і між собою, виколюючи команду відновлення як для самих себе, так і для ботів своєї команди.

Метод ClonAlg є одним з найбільш поширених методів, які функціонують на основі моделі клонального відбору. Цей метод може бути використаний для вирішення задачі керування ігровими ботами, які протидіють команді кораблів користувача, що керує ними власноруч. При цьому спосіб, в який метод буде проводити керування ботами відіграє ключову роль. Класичний варіант імунного алгоритму ClonAlg передбачає можливість зміни параметрів антитіл в процесі їх взаємодії із популяцією чужорідних антигенів шляхом клонування, мутації та саморегуляції популяції антитіл через виконання клонального відбору та старіння. При цьому зміни параметрів антигенів або не передбачалися зовсім, або були незначними, окрім того, можливості зникнення антигену в процесі формування імунної відповіді системою не передбачалося як таке. Але у

нашому випадку, в ході протистояння команди кораблів користувача із командою ігрових ботів системи така можливість передбачається. Слід зазначити, що при цьому методу роботи імунного алгоритму в такому разі є саме повне знищення всіх антигенів, які уособлюють команду кораблів користувача. Тому оригінальний варіант імунного методу ClonAlg не боже бути застосований для вирішення даної задачі без низки змін. Відповідно до цього, в модифікованій версії цього алгоритму – clonalg-rpg, закладається декілька важливих особливостей. По-перше, алгоритм в процесі своєї роботи не може безпосередньо змінювати значення ознак антитіл (набору ігрових ботів), а може тільки приймати рішення щодо доречності виконання тієї чи іншої команди для кожного конкретного бота або його клону. По-друге, вибір рішення щодо виконання тієї чи іншої команди бота має відбуватися з точки зору максимальної ефективності цієї дії з точки зору знищення всієї популяції антигенів, або з точки зору відновлення показника міцності всієї популяції антитіл, або конкретного окремого її бота. По-третє, умовою припинення роботи модифікованого імунного алгоритму clonalg-rpg є повне знищення всієї популяції антигенів, представленої набором кораблів користувача гри, до того, як популяція антитіл, яка представлена ігровими ботами, припинить своє існування. По-четверте, динамічні зміни ознак ботів-антитіл та антигенів-кораблів з команди користувача, відбуваються постійно, після виконання кожної дії антигенів та антитіл системи. Таким чином, реалізація модифікованої версії імунного алгоритму clonalg-rpg має спиратися на чотири визначені принципи, як на основу роботи алгоритму. Окрім того, важливою особливістю моделі клонального відбору є те, що в процесі роботи антитіла фокусуються в першу чергу на взаємодії із антигенами, а потім використовує деякі можливості взаємодії із іншими антитілами в процесі саморегуляції популяції антитіл.

Слід зазначити, що задля спрощення можливостей реалізації імунних методів та алгоритмів, їх прийнято розділяти на специфічні імунні оператори, які виконують ту чи іншу дію із популяцією антитіл або антигенів. При

цьому черговість цих операторів в алгоритмі не може бути змінена в процесі його роботи. Імунний оператор концентрується на реалізації однієї специфічної функції, що відповідає принципу Single Responsibility з концепції SILOD. За характером своєї дії та можливостями налаштування та конфігурування, імунні оператори прийнято розділяти на універсальні та специфічні. При цьому універсальні оператори можуть використовуватися у різних імунних методах через те, що виконують дії, які є актуальними для різних імунних моделей. Серед універсальних імунних операторів відзначають оператор клонування, оператор мутації, оператор презентації популяції антигенів, тощо. Специфічними імунними операторами називають оператори, які виконують функцію, властиву виключно для окремої специфічної моделі, наприклад для моделі конального відбору, або для моделі негативного відбору, тощо. Специфічними імунними операторами визнають оператор відбору клонів, оператор супресії, оператор апоптозу, оператор позитивного відбору і т.д.

Роботу модифікованого методу clonalg-rpg, створеного для вирішення задачі керування ігровими ботами, можна умовно описати на рівні імунних операторів наступним чином:

$$clonalg - rpg = \left[\begin{matrix} Presentation(AB, AG) \rightarrow \\ Cloning(AB, CL) \rightarrow \\ Mutation(AB, CL) \rightarrow \\ ClonSelection(AG, CL) \rightarrow \\ SelfRegulation(AB, CL) \rightarrow \\ Termination(AB, AG) \end{matrix} \right]^{AG>0}, \quad (3.1)$$

де $Presentation(AB, AG)$ – оператор представлення набору антигенів AG популяції антитіл AB; $Cloning(AB, CL)$ – оператор клонування популяції антитіл; $Mutation(AB, CL)$ – оператор мутації сформованих клонів; $ClonSelection(AG, CL)$ – оператор відбору клонів; $SelfRegulation(AB, CL)$ –

оператор саморегуляції популяції антитіл; $Termination(AB, AG)$ – оператор перевірки можливості зупинки роботи алгоритму.

Робота оператору $Presentation(AB, AG)$ зазвичай полягає у визначенні афінностей між антитілами та антигенами, а також афінностей між популяцією антитіл ШІС. При цьому під афінністю визначають міру подібності ознак антитіла та антигену, а робота імунного алгоритму зводиться до того, щоб в процесі клонування та мутації клони-нащадки того чи іншого антитіла відтворили ознаки того чи іншого антигену. Така цільова дія системи можлива при вирішення задачі класифікації, кластеризації, розпізнавання образів, прогнозуванні і т.д., але є недоцільною при вирішенні задачі керування ігровими ботами у RPG-грі. Оскільки підвищення рівня афінності між антитілами та антигенами є однією з цільових задач ШІС, то враховуючи специфіку задачі, афінность має відображати, наскільки швидко ШІС знищує зовнішній антиген в процесі своєї роботи, а не розпізнає та класифікує його. Відповідно до цього під час визначення афінностей clonalg-rpg буде використовувати поточний стан антигенів із їх уявним оптимальним станом, який характеризується однією ознакою – нульовим значенням параметру міцності та мінімальним значенням параметру енергії у ворожого корабля (антигена). Таким чином у визначенні афінності будуть використовуватися лише дві ознаки ігрових об’єктів з дев’яти, описаних раніше, що може суттєво пришвидшити роботу імунного алгоритму за рахунок зменшення кількості обчислювальних операцій. Враховуючи це, а також враховуючи вимоги до діапазону значень афінності $(0; 1.0]$, формула визначення афінності антигенів буде мати наступний вигляд:

$$affAg_i = \frac{1}{1 + (Ds_i - Dc_i) + (Es_i - Ec_i)}, \tag{3.2}$$

де $affAg_i$ – афінность і-го антигену, Ds_i – збережений показник міцності і-го антигену на старті бою, Dc_i – поточний показник міцності і-го антигену на

поточній ітерації бою, Es_i – збережений показник енергії і-го антигену на старті бою, Ec_i – поточний показник енергії і-го антигену на ітерації бою.

Оператор клонування антитіл $Cloning(AB, CL)$ використовується задля створення набору клонів для кожного антитіла поточної популяції ШІС з метою їх подальшої мутації. Слід зазначити, що зазвичай в різних моделях ШІС найбільш поширене використання оператору статичного клонування, або оператору пропорційного клонування. При цьому особливістю статичного клонування є обмеження кількості клонів, які створюються в процесі роботі ШІС, заздалегідь визначеним значенням, яке не може бути змінено в процесі роботи. Використання оператору пропорційного клонування передбачає залежність кількості клонів від значення афінності або авідності кожного конкретного антитіла. Відповідно до цього, більший показник афінності антитіла дає більшу кількість клонів при пропорційному клонування. Таким чином, при пропорційному клонуванні кількість клонів може динамічно змінюватися в процесі роботи ШІС. Для випадку алгоритму clonalg-rpg використання оператору пропорційного клонування є недоречним через обмежену кількість варіантів мутації, так само, як і використання статичної мутації для всіх антитіл, незалежно від їх ознак та характеристик. Тому оператор клонування в clonalg-rpg також буде специфічним, а саме, адаптивним, тобто кількість клонів для кожного окремого антитіла буде визначатися виходячи з його ознак та особливостей. При цьому визначальний вплив на кількість клонів, які створюються в процесі клонування буде мати кількість варіантів атаки кожного окремого корабля (бота). Відповідно до цього кількість клонів, які створюються завдяки використанню оператору адаптивного клонування, визначається наступним чином:

$$n_i = C_i \times N_{AG}, \tag{3.3}$$

де n_i – кількість клонів і-го антитіла, C_i – кількість варіантів взаємодії між цим антитілом та будь-яким антигеном, N_{AG} – кількість активних антигенів.

Оператор мутації клонів $Mutation(CL)$ значним чином впливає на швидкість вирішення цільової задачі, поставленої перед ШІС. Це відбувається через те, що саме механізм мутації забезпечує внесення змін у імунні об’єкти та наближує ШІС до вирішення своєї задачі. За принципом своєї роботи вирізняють декілька найбільш розповсюджених операторів мутації: статична, пропорційна та зворотно-пропорційна. Головною особливістю оператору статичної мутації є використання деякого незмінного значення, яке буде додаватися або відніматися від ознак клонів в процесі мутації. Використання оператору пропорційної мутації передбачає формування залежності глибини змін в ознаках клонів, які відбуваються в процесі мутації, від їх афінності до всієї популяції антигенів або до визначеної обмеженої кількості антигенів. При цьому більше значення між клоном та антигенами порозводить до зростання мутацій в ознаках цих клонів. Використання зворотно-пропорційної мутації передбачає використання зворотної залежності величини змін в ознаках клонів від показника їх афінності до антигенів. Слід зазначити, що враховуючи особливості задачі керування ігровими ботами у RPG-грі, використання жодного із вказаних видів оператору мутації неможливе без суттєвих модифікацій. Тому в алгоритмі clonalg-rpg передбачається використання оператору адаптивної мутації як модифікації оператора статичної мутації із можливістю зміни об’єкту взаємодії та характеру дії. Особливістю роботи оператору адаптивної мутації є те, що він не змінює безпосередньо параметри клонів антитіл, а концентрується на виборі способу взаємодії із антигенами. Відповідно до цього під час мутації для кожного окремого клону відбирається окремий специфічний спосіб взаємодії із кожним активним антигеном. При цьому під активним антигеном в контексті задачі керування ігровими ботами у RPG-грі розуміємо окремий корабель з команди космічних кораблів користувача, а під специфічним способом взаємодії розуміємо той чи інший спосіб атаки кожного окремого корабля команди користувача, або спосіб атаки всієї команди кораблів тим чи іншим ботом.

Оператор відбору клонів $ClonSelection(AG, CL)$ відіграє дуже важливу роль в роботі імунного алгоритму clonalg-rpg, заснованого на принципах клонального відбору. Саме завдяки ньому з переліку мутованих клонів відбирається один імунний об’єкт, або один спосіб взаємодії з тим чи іншим активним антигеном, або всім набором антигенів, який дає найбільший внесок у вирішення цільової задачі. Відповідно до цього в процесі клонального відбору обирається той клон, чия дія призвела до максимізації афінності якого-небудь окремого антигену, тобто переводу його у неактивний стан, або максимізації середньої афінності у популяції антигенів. Ці умови клон може досягнути за рахунок виконання своєї специфічної дії, яка призводить або до мінімізації значення міцності у якомусь конкретному антигені, або до суттєвого зменшення міцності у всьому наборі активних антигенів. Слід зазначити, що у разі, якщо той чи інший антиген в процесі роботи імунного алгоритму повністю вичерпує значення свого показника міцності – він переходить у неактивний стан, що проявляється у знищенні космічного корабля в команді користувача, із яким асоційовано цей антиген. Відповідно до цього знищений корабель з команди користувача стає неактивним антигеном, який більше не враховується в роботі модифікованого імунного алгоритму clonalg-rpg. Таким чином робота оператору клонального відбору в алгоритмі clonalg-rpg відповідає принципам роботи оператору елітарного клонального відбору, який користується максимальними показниками афінності з антигенами та використовується у таких поширених алгоритмах як CLONALG та BCA.

Оператор саморегуляції популяції антитіл $SelfRegulation(AB, CL)$ не специфічний для алгоритму CLONALG, та доданий у алгоритм clonalg-rpg замість оператору апоптозу, чия дія була спрямована на зменшення популяції імунних об’єктів за рахунок заміни клонованих антитіл обраними в результаті клонального відбору клонами. Слід зазначити, що завдяки особливостям організації оператору апоптозу, реалізованого у CLONALG, мінімізується кількість операцій обчислення афінностей між антитілами та

клонами, але такі операції не виключаються повністю. В алгоритмі clonalg-rpg замість цього оператору використовується оператор саморегуляції популяції антитіл, який передбачає визначення афінностей або визначення якихось інших метрик між популяцією антитіл та обраними в процесі клонального відбору клонами. Відповідно до цього між кожним клоном визначається афінность, яка визначає рівень взаємодії із іншими активними антитілами ШІС. Слід зазначити, що під активними антитілами розуміємо антитіла, які асоціюються із тим чи іншим кораблем з набору ігрових ботів, якими керує ШІС. При цьому якщо корабель, який є ігровим ботом, повністю втрачає всі одиниці показника міцності в ході ігрового бою, він стає неактивним, не приймає участі в ігровому процесі і не може бути відновленим з боку інших ігрових ботів. Такий ігровий бот перестає вважатися для ШІС антитілом, і відповідно до цього, він не більше не підлягає дій операторів клонування, мутації і т.д., та умовно вважається клітиною пам’яті, яка не приймає участі у подальших процесах ШІС. Головною задачею імунного оператора саморегуляції є визначення клонів, які можуть замінити активні антитіла та перевести ІШС та популяцію антитіл у новий стан, який наближує реалізацію задачі імунного алгоритму. Слід зазначити, що після проведення відбору клонів із всієї множини клонів, створених від кожного активного антитіла, обирається один, який характеризується максимальним значення афінності до антигенів. Після чого в процесі саморегуляції ШІС для цього клону приймається рішення щодо визначення можливої дії із активними антитілами. Такими діями може бути або виклик команди відновлення всього набору ігрових ботів, або виклик команди відновлення одного конкретного бота, який асоційований із цим клоном. Визначення можливості таких дій відбувається після розрахунку внутрішніх афінностей між клоном та активними антитілами. Внутрішня афінності визначаються декількома способами, в залежності від способу взаємодії: взаємодія із одним антитілом, або взаємодія із всією популяцією. Відповідно до цього першим варіантом взаємодії є взаємодія із одним

антитілом, від якого було створено цей клон, в такому разі формула визначення афінності буде мати наступний вигляд:

$$aff'Ab_i = \frac{1}{1+\left|Ds_i - \left(Dc_i + R_i\right)\right|}, \quad (3.4)$$

де $aff'Ab_i$ – афінность клона з і-антитілом, від якого цей клон було створено, Ds_i – збережений показник міцності і-го антитіла на старті бою, Dc_i – поточний показник міцності і-го антитіла на поточній ітерації бою, R_i – показник відновлення і-го антитіла.

Визначення показника внутрішньої афінності клону до всієї популяції антитіл ґрунтується на (3.3) та узагальнює цей вираз наступним цином:

$$aff''Ab_i = \frac{1}{S}\sum_{S}^{i=1}\left(1+\left|Ds_i - \left(Dc_i + \frac{1}{S}R\right)\right|\right)^{-1}, \quad (3.4)$$

де $aff''Ab_i$ – афінность клону із всією популяцією активних антитіл, в тому числі і антитілом, від якого цей клон було створено, S – розмір популяції активних антитіл, Dc_i – поточний показник міцності і-го антитіла на поточній ітерації бою, R – показник відновлення клону.

Спосіб можливої дії щодо відновлення характеристики міцності для одного антитіла, всієї популяції антитіл, обирається за значенням більшої афінності, тобто у випадку, якщо значення $aff'Ab_i$ є більшим ніж значення афінності $aff''Ab_i$, обирається перший спосіб відновлення. В іншому випадку обирається другий спосіб відновлення, який може провести ігровий бот під час виконання свого ходу впродовж гри. Слід зазначити, що у випадку якщо ігровий бот має тільки один спосіб відновлення, афінность між ним та активними антитілами визначається тільки один раз.

На завершальному етапі роботи оператора саморегуляції ШІС в clonalg-rpg для кожного окремого клону приймається фінальне рішення щодо вибору способу його дії – атаки того чи іншого супротивника, або групи супротивників, представлених набором активних антигенів, чи відновлення одного чи групи ботів, представлених активними антитілами ШІС. Остаточне рішення щодо вибору дії приймається після співставлення афінностей, які характеризують взаємодію цього клону із набором активних антигенів та взаємодію із набором активних антитіл. У разі, якщо афінность, яка характеризує взаємодію із антитілами більша афінності, що описує взаємодію із антигенами, для бота, асоційованого із даним клоном, приймається рішення щодо відновлення антитіл. В іншому випадку ігровий бот буде атакувати обраний корабель команди гравця, або всі його кораблі, в залежності від типу атаки. Після цього обрання ігровий бот реалізує обраний спосіб дії, змінюючи ознаки антигенів, або антитіл. Слід зазначити, що оскільки зміни видуваються тільки в ознаках антигенів та антитіл, а клон символізує той чи інший спосіб можливої дії, подальше збереження клонів для формування імунної пам’яті є недоцільним. Тому після виконання обраної дії клон видаляється з об’єктів системи.

Оператор перевірки можливості зупинки бою $Termination(AB, AG)$ використовується для перевірки можливості зупинки процесу ігрового бою. Найбільш поширеними типами операторів завершення роботи ШІС є статичний, критеріальний, повний та комбінований. При використанні статичного оператору зупинки роботі імунного алгоритму використовується значення максимальної кількості популяцій клонів, які формуються ШІС в процесі своєї роботи. При використанні критеріального оператору припинення роботи ШІС передбачається досягнення кожним активним антитілом системи стану умовної специфічності, який характеризується визначеним значення максимальної афінності, до того чи іншого антигену. Відповідно до цього ШІС зупиняє свою роботу у випадку, якщо в популяції активних антитіл всі імунні об’єкти достягають стану повної або умовної

специфічності до того чи іншого антигену. Використання оператору зупинки ШІС, який використовує комбінований підхід, передбачає як можливість досягнення стену повної або умовної специфічності, так і використання деякого визначеного ліміту на кількість популяцій клонів, які створює ШІС в процесі своєї роботи під час вирішення тієї чи іншої практичної задачі. Цей ліміт на формування популяцій клонів є своєрідним запобіжником, який унеможливлює імовірність виникнення нескінченного циклу клонування імунних об’єктів, їх подальшої мутації, відбору тощо.

Слід зазначити, що в алгоритмі cloanlg-rpg використовується критеріальний оператор зупинки роботи ШІС, який передбачає зупинку роботи імунних операторів у разі відсутності активних антигенів або активних антитіл. Відповідно до цього, якщо в системі є хоча б одне активне антитіло, або ШІС взаємодіє хоча б з одним активним антигеном, система буде продовжувати свою роботу.

3.2 Загальна архітектура проекту та реалізація clonalg-rpg

Програмна реалізація ігрового програмного засобу в жанрі RPG, із керуванням ігровими ботами на основі імунної моделі клонального відбору проводилася на Unity 2019 при використанні платформи .NET та мови програмування C#. Для формування архітектури ігрового проекту було використано концепцію багаторівневої архітектури. Відповідно до цього архітектура проекту формується з двох рівнів: рівень Model та рівень View. Такий варіант розділення коду проекту було обрано через те, що використання універсальних трирівневих патернів типу MVC, MVP або MVVM недоречна для невеликого проекту, метою створення якого є реалізація та випробування модифікованого імунного алгоритму клонального відбору clonalg-rpg, створеного для керування ігровими ботами. Недоцільність використання універсальних трирівневих патернів ґрунтується на необхідності створення певних об’ємів інфраструктурного коду, який

формується у вигляді чистих абстракцій, таких як інтерфейси або абстрактні класи, може призвести до ускладнення проекту, для якого не передбачається подальшої масштабованості. Слід зазначити, що при цьому повна відмова від використання архітектури проекту може призвести до зростання його складності та кодифікованості. Тому для ігрового проекту було створено дворівневу архітектуру типу Model-View. При цьому в проекті передбачається автоматизація обміну даних між рівнем Model та рівнем View за рахунок використання технології односторонньої прив’язки даних, запозиченої з архітектурного шаблону MVVM. Відповідно до цього ігровий додаток умовно поділяється на декілька основних програмних модулів:

- модуль інтерфейсу користувача, реалізований на рівні View;
- модуль представлення ігрових даних, реалізований на рівні Model;
- модуль прив’язки даних, реалізований на рівні View;
- модуль імунного алгоритму та операторів, реалізований в Model;
- модуль керування поведінкою ігрових ботів, реалізований в Model;
- модуль файлових операцій, реалізований на рівні Model;
- модуль логування даних, реалізований на рівні Model.

Таким чином рівень View відповідає за модуль інтерфейсу користувача та модуль, який забезпечує реалізацію односторонньої прив’язки даних з рівня Model на рівень інтерфейсу користувача у View. У модулі інтерфейсу користувача знаходиться перелік типів даних, які використовуються для організації взаємодії із користувачем та відповідають за візуальне відображення ходу ігрового процесу RPG-гри. Саме цей модуль зберігає не тільки частину коду, яка асоціюється з тим чи іншим вікном ігрового додатку, але й менеджер вікон, який забезпечує навігацію між різними екземплярами відображення проекту, що надають користувачу можливості щодо вибору та редагування команд кораблів, вибору команди супротивника (яка складається з набору ігрових ботів під керуванням ШІС), вікна проведення битви, вікна із результатами битви та вікна із інформацією про дані та результати бою різних користувачів, представлені набором

облікових записів, які зберігаються у специфічному файлі. Окрім того, на рівні View знаходиться кодова частина для всіх видів прив’язки даних (data bindings), які асоціюють дані з типів, представлених на моделі із візуальною частиною інтерфейсу користувача для різних вікон ігрового додатку. Слід зазначити, що для організації прив’язки даних у проекті використовується механізм одностороннього біндінгу, який реалізовано шляхом використання івентів та рефлексії типів.

У модулі представлення ігрових даних знаходиться перелік типів, які використовуються для опису облікових записів користувачів ігрового додатку, космічні кораблі, які приймають участь у ігровому бою, допоміжні типи, які використовуються для організації ігрового бою двох команд із кораблями, а також типи даних, які використовуються для опису ігрової статистики та логування даних про ігровий процес, та черговість ходів космічних кораблів різних команд у ігровому бою. Відповідно до обраної дворівневої архітектури, всі ці типи даних розміщуються на рівні Model. Окрім того, на рівні Model знаходяться типи, які забезпечують роботу імунного алгоритму керування ігровими ботами clonalg-rpg, описують представлення антигенів, антитіл, а також ієрархію типів, які реалізують поведінку різних імунних операторів, зібраних у алгоритмі clonalg-rpg. При цьому для спрощення реалізації антигенів, антитіл та клонів було використано ряд абстракцій, в тому числі абстрактний базовий клас клітини. Також слід зазначити, що внутрішня організація типів ігрового проекту відповідає основним концепціям об’єктно-орієнтованого програмування та базовим принципам SOLID. Модуль імунного алгоритму, реалізований на рівні Model також визначає специфічний тип, який має реалізацію методу clonalg-rpg, який використовує специфічний інтерфейс для виконання набору своїх імунних операторів, що відповідає принципу інверсії залежностей та підвищує гнучкість до модифікації ігрового проекту.

Модуль керування поведінкою ігрових ботів знаходиться на рівні Model та забезпечує взаємодію між операторами імунного алгоритму, які

оперують специфічними типами антитіл, антигенів і т.д. та представленням кораблів та ігрових ботів а також типом-менеджером ігрового бою, який забезпечує взаємодію кораблів користувача та ігрових ботів. Окрім цього, даний модуль також відповідає за логічне представлення карти ігрового бою, та розміщення команд із кораблями користувача та ігровими ботами, які знаходяться під керівництвом ШІС та алгоритму clonalg-rpg.

На рисунку 3.1. відображається файлова структура ігрового проекту, розділена на два основних рівня у відповідності до прийнятого рішення щодо реалізації дворівневої архітектури.

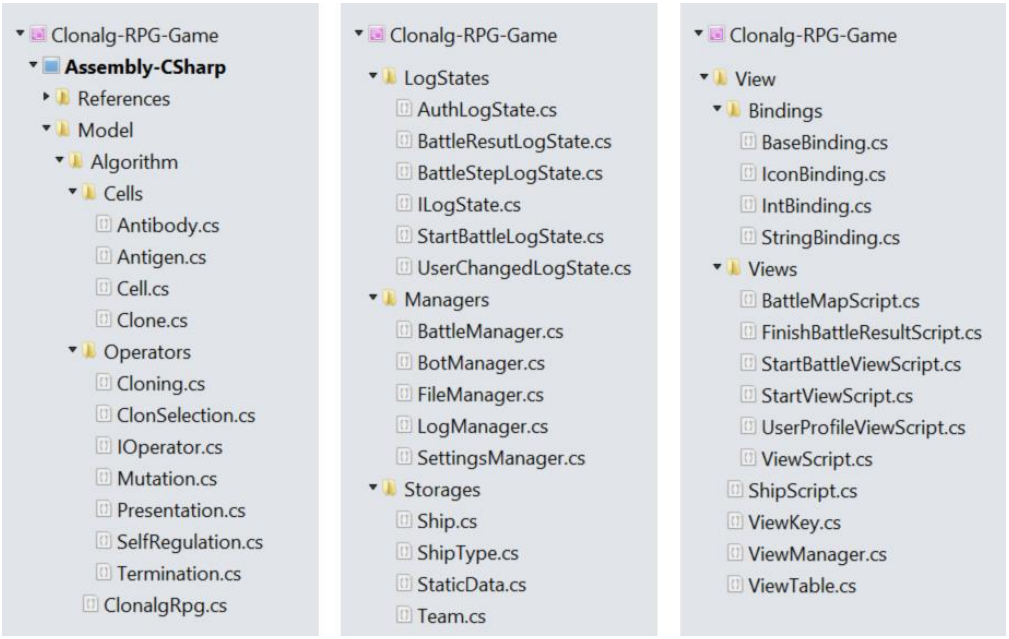

Рисунок 3.1 – Файлова структура ігрового проекту

Слід зазначити, що важливу роль у дослідженні роботи імунного алгоритму clonalg-rpg відіграють модулі файлових операцій та логування даних ігрового бою також реалізовані на рівні Model. Вони використовуються для завантаження та зберігання даних того чи іншого користувача шляхом json-серіалізації у специфічний файл із обліковими даними гравців. Система логування даних ігрового бою використовується для того, щоб забезпечити можливість дослідження ходів кораблів

користувача та ігрових ботів, що знаходяться під керуванням алгоритму clonalg-rpg. Дані про хід ігрового бою двох команд автоматично зберігаються у csv-файл із датуванням часу початку битви. Завдяки цьому можна дослідити черговість ходів та пріоритетність у прийнятті рішень clonalg-rpg.

3.3 Результати випробувань clonalg-rpg

Для аналізу ефективності імунного методу clonalg-rpg, створеного для керування ігровими ботами у RPG-грі, було проведено декілька ігрових сесій із різним складом команд кораблів користувача та команд ботів. Певний вплив на результати бою відігравав вибір фракції, що призводило до використання бонусів у різних параметрах кораблів. Задля того, щоб користувач міг відбирати не один і той самий тип космічних кораблів, які характеризуються великими значеннями у основних характеристиках, що значним чином впливає на хід ігрового бою, у користувача було обмеження у ресурсах, а саме у коштах, які він може витрачати на формування своєї команди при повному доступі до всіх типів кораблів з кожної фракції. Відповідно до цього, користувачу надається можливість комбінувати кораблі різних фракцій в своїй команді. Слід зазначити, що команди ігрових ботів можуть генеруватися випадковим чином без будь-яких обмежень по вартості, або завантажуватися зі специфічного файлу із налаштуваннями специфіки бою. У випадку генерації команд ботів випадковим чином відбувається формування як дуже сильних дорого вартісних команд, так і до формування команд із невеликою вартістю та можливостями. Окрім того, значний вплив на хід ігрового бою, а також на швидкість прийняття рішення алгоритмом clonalg-rpg, мали кількість способів атаки кораблів, а також кількість способів відновлення.

При дослідженні ефективності алгоритму clonalg-rpg було зроблено біля сотні випробувань із різними складами команд користувача та команди ігрових ботів різних типів та різних фракцій. Слід зазначити, що під час цих

випробувань команди формувалися не випадковим чином, а через специфічні налаштування, враховуючи, що космічні кораблі за своїми параметрами та можливостями можна розділити на три класи, де перший клас формують найдешевші кораблі типу support та scout, другий клас формують кораблі типу cruiser та frigate, а третій найбільш дорого вартісний клас формують важкі кораблі, які займають по два місця на карті ігрового бою – destroyer та dreadnought. При цьому параметри цих кораблів, такі як сила атаки, енергія, міцність і т.д. у ігровому балансі налаштовані таким чином, що кораблі першого класу поступаються приблизно 25% кораблям другого класу, і приблизно на 50% кораблям третього класу. Відповідно до цього, кораблі другого класу по своїм основним показникам поступаються кораблям третього класу на 25%. Однак задля балансування гри кораблі першого ласу переважають всі інші кораблі по показнику швидкості і роблять свій хід під час ігорного бою в першу чергу.

Випробування алгоритму clonalg-rpg проводилися на одному наборі вхідних даних про команди кораблів користувача та ігрових ботів трьома способами: без використання ШІС, коли користувач керує обома командами безпосередньо (це нетиповий спосіб роботи проекту, який налаштовується специфічним чином), без використання ШІС шляхом обрання способу дії ігрового бота випадковим чином, та із використанням алгоритму clonalg-rpg, який керує поведінкою ігрових ботів в бою. Головним показником ефективності імунного алгоритму керування ігровими ботами є кількість перемог ШІС над людиною у ігрових боях, в яких приймають участь співставні або рівні по своїх основних параметрах команди кораблів. Окрім того, при дослідженні ефективності враховувався не тільки відсоток перемог ШІС над користувачем, але й кількість кораблів, яка залишилася в команді переможця. Відповідно до цього було виділено наступні показники:

- у переможця залишився 1 корабель;
- у переможця залишилося 2 корабля;
- у переможця залишилося 3 і більше кораблів.

Використання такої деталізації дозволить дослідити не тільки сам факт перемоги, але й наскільки переможець переважав свого опонента під час ігрового бою, що може свідчити про рівень майстерності гри. Слід зазначити, що більшість випробувань було проведено на двох типах команд – команди у складі 5 у користувача та 5 кораблів у супротивника, а також команди у складі 4 кораблів у користувача та 4 кораблів у супротивника. При цьому кораблі було обрано однакові в усіх командах задля уникнення можливих переваг по параметрах різних кораблів з різних фракцій. Окрім того, важливий вплив відіграє і сам скла команди, а саме, у командах типу 5 на 5 використовується наступний перелік кораблів: support та scout на позиціях подалі від команди супротивника, cruiser, frigate та destroyer формують першу лінію супротиву, враховуючи те, що destroyer займає 2 місця на полі ігрового бою. У випадку використання команди типу 4 на 4, було використано два кораблі типу destroyer, один support у другій лінії від команди супротивника, та один frigate на першій лінії від команди супротивника. У таблиці 3.1. наведено відсоток перемог команд користувача над командами його опонента з урахуванням різних способів керування командою супротивника, визначених раніше.

Таблиця 3.1 – Частота перемог користувача над опонентом

Типи команд	Ручне керування без ШІС	Випадкове керування	Керування clonalg-rpg
5 на 5	50 %	85,15 %	50,35 %
4 на 4	50 %	96,75 %	49,15 %

Слід зазначити, що для кожного типу керування командою ігрових ботів було проведено по 20 ігрових битв задля отримання статистичних даних щодо результатів битв та вибору типу дій ботами під час ігрового бою. Відповідно до результатів наведених вище, найменш ефективним є керування ігровими ботами, зроблене випадковим чином, що майже завжди призводило до гарантованої поразки. Натомість організація керування

ігровими ботами на основі алгоритму clonalg-rpg показало результати співставні із результатами отриманими шляхом залучення людини для керування кораблями супротивника. Також з результатів, отриманих в процесі випробування імунного алгоритму, видно, що при зменшенні кількості кораблів в команді шляхом використання важких кораблів, які мають більшу варіативність можливих дій, незначним чином підвищується шанс перемоги ігрових ботів, що може свідчити про те, що у випадку збільшення кількості варіантів атаки та відновлення створений імунний метод clonalg-rpg буде показувати більшу ефективність ніж користувач. Для того щоб оцінити майстерність гри команди переможця було зафіксовано кількість кораблів, які залишилися в команді переможця після проведення ігрового бою, ці результати наведені у таблиці 3.2.

Таблиця 3.2 – Кількість кораблів, які залишилися в команді переможця

Типи команд		Ручне керування без ШІС	Випадкове керування	Керування clonalg-rpg
5 на 5	1	100,00 %	12,40 %	93,75 %
	2	0,00 %	82,35 %	6,25 %
	3+	0,00 %	5,25 %	0,00 %
4 на 4	1	100,00 %	3,60 %	95,00 %
	2	0,00 %	88,25 %	5,00 %
	3+	0,00 %	8,15 %	0,00 %

Відповідно до наведених результатів, можна зробити висновок, що використання способу керування ігровими ботами випадковим чином створює для користувача найлегші умови, тому що в більшості випадків саме користувач одержував перемоги над командою ботів, яка керується у випадковий спосіб прийняття рішень, при цьому в команді користувача майже завжди залишається по 2 корабля, навіть у випадку використання команд з 4 кораблів. На відміну від цього, згідно з отриманими результатами можна зробити висновок, що використання алгоритму clonalg-rpg створює

для користувача складні умови для отримання перемоги над командою ігрових ботів, при цьому в команді переможця зазвичай залишається лише один корабель, в той час як інші втрачаються в процесі ігрового бою.

Окремим напрямком спостереження виступив тип дії, яку обирає та виконує ігровий бот під час свого ходу. Цей вибір відіграє важливу роль, оскільки саме він призводить команду ігрових ботів до перемоги над командою користувача або поразки. Слід зазначити, що при отриманні можливості зробити свій хід в порядку черги, яка формується на основі швидкості всіх активних кораблів з обох команд, які приймаються участь у ігровому бою, можна обрати або атаку супротивника, або відновлення. Пропустити хід неможливо ані ігровому боту, ані кораблю, який знаходиться під безпосереднім керуванням з боку користувача. В таблиці 3.3 наведені статистичні дані відносно частоти вибору того чи іншого способу дії ігровими ботами, які знаходяться під різними типами керування, визначеними раніше.

Таблиця 3.3 – Частота вибору способів дій в боях із командами 5 на 5

Варіанти дій корабля	Ручне керування		Випадкове керування		Керування clonalg-rpg	
	5 на 5	4 на 4	5 на 5	4 на 4	5 на 5	4 на 4
I спосіб ближньої атаки	19,33 %	9,82 %	24,56 %	18,89 %	20,29 %	11,01 %
II спосіб ближньої атаки	9,83 %	10,42 %	11,22 %	10,56 %	10,86 %	8,57 %
III спосіб ближньої атаки	3,15 %	7,89 %	2,22 %	5,56 %	3,90 %	9,76 %
I спосіб дальньої атаки	22,00 %	6,05 %	17,89 %	10,56 %	20,00 %	6,25 %
II спосіб дальньої атаки	5,83 %	10,62 %	6,22 %	10,56 %	4,86 %	8,57 %
III спосіб дальньої атаки	3,15 %	7.89 %	2,22 %	5,56 %	3,90 %	9,76 %
I спосіб відновлення	19,87 %	18,65 %	24,56 %	18,89 %	19,67 %	18,75 %
II спосіб відновлення	14,02 %	22,42 %	8,89 %	13,89 %	14,52 %	22,32 %
III спосіб відновлення	2,50 %	6,25 %	2,22 %	5,56 %	2,00 %	5,00 %

Виходячи з результатів дослідження рішень, прийнятих ігровими ботами відповідно до різних способів керування можна зробити висновок, що

імунний алгоритм clonalg-rpg надає більшу перевагу III способу атаки як на ближньої та і на дальньої дистанції, мінімізуючи при цьому використання II способу розподіленої атаки кораблів команди супротивника. Відповідно до цього у випадку рішення ігрового бота щодо проведення атаки супротивника під керівництвом clonalg-rpg, найбільшою увагою користуються способи концентрованої атаки визначеного корабля, при цьому, за можливістю, із нанесенням додаткової шкоди енергії корабля з команди супротивника, що суттєво впливає на його подальші можливості в бою. При цьому у випадку відновлення перевага також надається саме способу концентрованого відновлення одного обраного корабля. Таку поведінку можна охарактеризувати як стратегію концентрованого знищення найслабших кораблів з команди супротивника при максимальному концентрованому відновленні пошкоджених кораблів своєї команди.

Слід зазначити, що запропонований метод clonalg-rpg використовується для керування ігровими ботами в PRG-проектах вперше, тому на сьогоднішній день достатньо важко робити порівняння із іншими імунними та неімунними методами, які використовуються для організації системи керування ігровими ботами і при цьому є висвітленими у літературу на рівні, достатньому для їх відтворення з метою подального порівняння. Також важливо відмітити, що запропонований імунний метод простий у реалізації та модифікації, що робить можливим його подальше використання для керування ігровими ботами у іграх інших жанрів.

ВИСНОВКИ

У кваліфікаційній розглянуто вирішення актуального завдання керування ігровими ботами на основі теорії штучних імунних систем та моделі клонального відбору. При виконанні поставленого завдання було проведено аналіз найбільш поширених імунних моделей та методів, які можуть використовуватися для вирішення різних задач інтелектуальної обробки інформації, класифікації та прийняття рішень.

Проведений аналіз імунних підходів та принципів функціонування дозволив визначити їх основні особливості, переваги, недоліки та можливості модифікації та адаптації для вирішення різних практичних задач. Крім того, для моделі клонального відбору було проаналізовано існуючи поширені алгоритм, які можуть бути взятими за основу з метою їх подальшої модифікації та вирішення задачі керування ігровими ботами у ігрових проектах жанру RPG. Також було визначено універсальні імунні оператори, які можна використовувати під час розробки імунного методу керування поведінкою ігрових ботів, та проведено їх адаптацію для вирішення поставленої задачі дослідження.

В результаті проведення низки експериментів з моделювання імунних методів, реалізованих на основі моделі клонального відбору, було створено метод clonalg-rpg, який функціонує на основі поширеного методу CLONALG та використовує принцип клонального відбору та принцип саморегуляції у популяції антитіл, як найбільш підходящий для вирішення задачі керування ігровими ботами. Використання даного методу для вирішення поставленої задачі обумовлюється простотою реалізації та модифікації, а також максимальною швидкістю в межах та умовах специфіки ігрового балансу RPG-проекту. Розроблений метод clonalg-rpg характеризується високою швидкістю обчислення та зручністю реалізації та модифікації для подальшого застосування в ігрових проектах інших жанрів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Фомічов О. О., Бурцев В. С. Дослідження імунних операторів в моделі штучної імунної мережі, Системи управління, навігації та зв'язку, Випуск 2 (72), 2023, С 143-150.

2. M. Read, P.S. Andrews, J. Timmis, An Introduction to Artificial Immune Systems. In Handbook of Natural Computing, Shringer, Berlin, Germany (2012) 1575–1597.

3. K.B. Bahekar, Classification techniques based on Artificial immune system algorithms for Heart disease using Principal Component Analysis. International Journal of Scientific Research in Science, Engineering, and Technology, IJSRSET, Vol. 7, Iss. 5 (2020) 150-160.

4. S. Shekhar, D.K. Sharma, D.K. Agarwal, Y. Pathak, Artificial Immune Systems-Based Classification Model for Code-Mixed Social Media Data. IRBM, Vol. 43, Iss. 2, (2022) 120-129.

5. R.M. Mikherskii, Analysis of the Use of Artificial Immune Systems. In: IOP Conference Series: Materials Science and Engineering, (2021) 1-6.

6. R. Diestel, Extreaml Graph Theory. In: Graph Theory. Graduate Texts in Mathematics, Vol 173, Springer, Berlin, Heidelberg, 2017.

7. R.O. Duda, P.E. Hart, Pattern classification. Wiley & Sons, 2010.

8. D. Dasgupta, L.F. Nino, Immunological computation, Theory, and applications. Taylor & Francis Group, 2009.

9. D. Dasgupta, S. Yu, L.F. Nino, Recent Advanced in Artificial Immune Systems: Models and Applications. Applied Soft Computing, Elsevier (2011) 1574-1587.

10. H. Park, J.E. Choi, D. Kim, S.J. Hong, Artificial immune system for fault detection and classification of semiconductor equipment. Electronics, Vol. 10, No. 8, 944 (2021) 1-14.

11. S.S.F. Souza, F.P.A. Lima, F.R. Chavarette, A New Artificial Immune System Based on Continuous Learning for Pattern Recognition. Revista de Informatica Teorica e Aplicada, RITA. Vol. 27, No. 04, (2020) 34-44.

12. A.T. Haouari, L. Souici-Meslati, F. Atil, D. Meslati, Empirical comparison and evaluation of artificial immune systems in inter-release software fault prediction. Applied Soft Computing Journal, 96, (2020) 1–18.

13. V. Cutello, G. Nicosia, Multiple learning using immune algorithms. In: Proceedings of 4th International Conference on Recent Advances in Soft Computing, RASC (2022) 102–107.

14. G. Samigulina, Z. Samigulina. Biologically Inspired Unified Artificial Immune System for Industrial Equipment Diagnostic. International Conference on Machine Learning, Optimization, and Data Science. Lecture Notes in Computer Science book series LNCS, vol. 13811 (2023) 77-92.

15. M. Korablyov, O. Fomichov, N. Axak, Classification of objects based on a tree-shaped artificial immune network model. Advances in Intelligent Systems and Computing V, Springer, 2021, 160-172.

16. L. Eldén, Matrix Methods in Data Mining and Pattern Recognition, Second Edition, SIAM, 2019.

17. Mirkin B.G. Clustering for Data Mining. A Data recovery Approach / B.G. Mirkin. – Taylor & Francis Group, 2005. – 278 p.

18. Han J. Data Mining Concepts and Techniques Second Edition / J. Han, M. Kamber. – Elsevier, 2006. – 772 p.

19. de Oliveira J.V. Advances in fuzzy clustering and its applications / J.V. de Oliveira, W. Pedrycz. – John Willey & Sons, 2007. – 460 p.

20. Gan G. Data clustering theory, algorithms, and applications / G. Gan, C. Ma – Society Industrial and Applied Mathematics, SIAM, 2007. – 490 p.

21. Igawa K. Discrimination-based artificial immune system: modeling the learning mechanism of self and non-self discrimination for classification / K. Igawa, H. Ohashi // Journal of Computer Science, № 4, 2007. – P. 204-211.

22. Leung K. Generating compact classifier systems using a simple artificial immune system / K. Leung, F. Cheong, C. Cheong // IEEE, Transactions on Systems, Man, and Cybernetics, № 5, 2007. – P. 1344-1356.

23. Литвиненко В.І. Вирішення задачі класифікації з використанням механізмів ідіотипічної мережі / Литвиненко В.І. // Наукові праці: науково-методичний журнал, серія «Комп'ютерні технології» – МДТУ ім. П.Могили, Вип. 44, 2006. – С. 136-146.

24. Duda R.O. Pattern classification Second Edition / R.O. Duda. – Willey-Interscience, 2000. – 738 p.

25. Mittal A. Addressing the Problems of Bayesian Network classification of Video Using High Dimensional Features / A. Mittal, // IEEE Transactions on Knowledge and Data Engineering-04, Issue 2, 2004. – P. 230-244.

26. Han J. Data Mining Concepts and Techniques Second Edition / J. Han, M. Kamber. – Elsevier, 2006. – 772 p.

27. C. Lan, H. Zhang, X. Sun, Z. Ren, An intelligent diagnostic method based on optimizing B-cell pool clonal selection classification algorithm. Turkish Journal of Electrical Engineering and Computer Sciences, 28 (2020) 3270–3284.

28. Fielding A.H. Cluster and classification techniques for the biosciences / A.H. Fielding. – Cambridge University Press, 2007. – 260 p.

29. Gordon A.G. Classification Second Edition / A.G. Gordon. – CRC Press, 1999. – 248 p.

30. Garain U. Recognition of handwritten indic script using clonal selection algorithm / U. Garain, M.P. Chakraborty, D.Dasgupta // Springer, Lecture Notes in Computer Science, № 4163, 2006. – P. 256-266.

31. Secker A. AISEC: an artificial immune system for e-mail classification / A. Seckler, A.A. Freitas, J. Timmis // IEEE, Proc. The Congress on Evolutionary Computation, CEC-03, 2003. – P. 131-139.

32. Кукарцев, В. В. Порівняння систем контролю версій: Git, Mercurial, CVS і SVN [Текст] / В. В. Кукарцев, С. А. Бадарчи // Синергия наук. – 2018. – №19. − С. 538-548.

33. Вирт, Н. Алгоритми та структури даних. Нова версія для Оберона [Текст] / Н. Вирт. – М.: ДМК Пресс, 2010. – 410 с.

34. Puntambekar, A. A. Software Engineering [Текст] / A. A. Puntambekar. Technical Publications, 2009. – 332 p.

35. Худяков, А.А. 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).