Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Модель штучної імунної мережі для керування ігровими персонажами у комп’ютерній грі

(тема)

Виконав:

студент II курсу, групи СПм-21-2

Бурцев В.С.
(прізвище, ініціали)

Спеціальність

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

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

Освітня програма

Системне програмування
(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)

2023 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-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

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

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Бурцеву Владиславу Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Модель штучної імунної мережі для керування ігровими персонажами у комп'ютерній грі

затверджена наказом по університету від “ 04 ” квітня 2023 р. № 318 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 17 травня 2023 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 – 26 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	29.03.23-05.04.23	
2	Вибір та обґрунтування методики дослідження	06.04.23-16.04.23	
3	Вибір інструментальних засобів	17.04.23-22.04.23	
4	Проведення експериментів	23.04.23-29.04.23	
5	Оформлення матеріалів кваліфікаційної роботи	30.04.23-07.05.23	
6	Подання кваліфікаційної роботи керівникові та	12.05.23-13.05.23	
	її попередній захист		
7	Подання кваліфікаційної роботи на	14.05.23-17.05.23	
	рецензування		

Дата видачі завдання 4 квітня 2023 р.

Студент _____
(підпис)

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

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 86 с., 13 рис., 2 табл., 1 дод., 33 джерел.

C#, .NET, UNITY, КЛАСИФІКАЦІЯ, ШТУЧНА ІМУННА МЕРЕЖА АФІННІСТЬ, АНТИТІЛО, АНТИГЕН, КЛОНУВАННЯ, МУТАЦІЯ, СУПРЕСІЯ, ІГРОВИЙ ПЕРСОНАЖ

Метою кваліфікаційної роботи є розробка, реалізація та випробування імунного методу керування поведінкою ігрових персонажів у комп’ютерній грі. Слід зазначити, що метод керування поведінкою має бути розробеним на основі моделі штучної імунної мережі. Яка ще не використовувалася для організації системи штуного інтелекту в грових додатках.

В ході виконання кваліфікаційної роботи був проведений аналіз існуючих імунних моделей та методів, їх переваг та недоліків. Також було розглянуто та досліджено питання впливу особливостей та налаштувань імунних операторів, з яких формуються поширені на сьогодні імунні моделі та методи. Було реалізовано ігровий додаток типу action, в якому передбачається існування великої кількості ігрових персонажів із різними рисами характеру, які визначають той чи інший клас поведінки.

Розроблений програмний засіб виконує покладене на нього завдання керування поведінкою ігрових персонажів у комп’ютерній грі різними способами та з урахуванням можливостей роботи різних імунних операторів.

ABSTRACT

Master’s thesis: 86 pages, 13 figures, 2 tables, 1 appendices, 33 sources.

C#, .NET, UNITY, CLASSIFICATION, ARTIFICIAL IMMUNE NETWORK, AFHINNITY, ANTIBODY, ANTITIGENE, CLONING, MUTATION, SUPRESSION, GAME CHARACTER

The purpose of the qualification work is to develop, implement and test the immune method of controlling the behavior of game characters in a computer game. It should be noted that the behavior management method should be developed on the basis of the artificial immune network model. Which has not yet been used to organize the system of artificial intelligence in game applications.

In the course of the qualification work, an analysis of existing immune models and methods, their advantages and disadvantages was carried out. The question of the influence of the features and settings of immune operators, from which the immune models and methods common today are formed, was also considered and investigated. An action-type game application was implemented, which assumes the existence of a large number of game characters with different character traits that determine one or another class of behavior.

The developed software tool fulfills the assigned task of controlling the behavior of game characters in a computer game in various ways and taking into account the capabilities of different immune operators.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 8
ВСТУП 9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ПРОЕКТУВАННЯ 10
1.1 Огляд ігрових жнрів 10
1.2 Огляд програмних засобів для розробки ігор 16
1.3 Вибір мови програмування та засобів розробки 23
1.4 Біологічні принципи функціонування систем штучного інтелекту 26
1.5 Складання вимог та постановка задачі проектування 32
2 ОГЛЯД ВИКОРИСТАИХ ТЕХНОЛОГІЙ ТА ОСОБЛИВОСТІ ІГРОВОГО ДИЗАЙНУ 33
2.1 Принципи SOLID та їх вплив на розробку програмних засобів 33
2.2 Компонентно-орієнтована архітектура Unity-проектів 36
2.3 Особливості архітектурного партерну MVI 39
2.4 Штучні імунні системи та задача керування персонажами в іграх 43
2.5 Особлиості роботи імунних операторів в моделі aiNET 49
2.6 Особлиості дизайну ігрового додатку 53
3 АРХІТЕКТУРА ІГРОВОГО ПРОЕКТУ ТА МОДЕЛЬ ШТУЧНОЇ ІМУННОЇ МЕРЕЖІ ДЛЯ КЕРУВАННЯ ПЕРСОНАЖАМИ В ІГРАХ 59
3.1 Формат даних ігрового персонажу 59
3.2 Особливості методу baiNET та керування поведінкою персонажів 60
3.3 Реалізація архітектури MVI для ігрового проекту 66
3.4 Результати випробувань baiNET в ігровому проекті 70
ВИСНОВКИ 73
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 74

ДОДАТОК А Графічний матеріал кваліфікаційної роботи..... 77

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

АНТИГЕН – це зовнішній вплив (вірус, бактерія).

АНТИТІЛО – це компонент штучної імунної системи, внутрішня клітина, що відповідає можливому рішенню задачі

ІА – імунний алгоритм

ІМ – імунна модель

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

ПЗ – програмне забезпечення

ПОПУЛЯЦІЯ – це множина можливих рішень задачі.

СКВ – системи контролю версій

ШІС – штучна імунна система

aiNET – Artificial Immune Network

ВСТУП

На сьогоднішній день розвиток ігрових додатків оказав значний вплив на поведінку та звички суспільства. Велику роль у цьому відіграють багато-користувальницькі ігри, навколо яких формуються соціальні зв’язки, фан-групи а також бізнес-проекти. Пандемія COVID-19 виступила каталізатором росту популярності ігрових додатків в світі. Тому ігрова індустрія зараз має найстрімкіший розвиток. Тепер в популярні ігри можна грати і на персональних комп'ютерах, і в соціальних мережах, і на ігрових консолях, а також на мобільних пристроях – телефонах та планшетах.

Стрімкий розвиток інформаційних технологій обумовив появу нових можливостей для розробників та видавців ігрових програмних засобів, а також спрощення та вдосконалення же існуючих технологій. Завдяки цьому в області розробки ігрових додатків з'явилася велика кількість платформ, шаблонів розробки архітектури ігрових додатків, а також ігрових движків - середовищ, які дозволяють спростити та автоматизувати процес розробки та розгортання ігрових програмних засобів.

Окрім того великої популярності зараз набирають системи штучного інтелекту, які дозволяють автоматизувати вирішення складних практичних задач щодо інтелектуальної обробки та аналізу даних. На сьогодні штучний інтелект може використовуватися як повноцінний співбесідник, вчитель, юрист або адвокат. Суттєвий вплив розвиток систем розробки штучного інтелекту оказав на ринок ігрових програмних засобів.

У цій роботі будуть розглядатися проблеми створення штучного інтелекту ігрових персонажів, які не знаходяться під безпосереднім керуванням користувачів, задля імітації поведінки гравців багатокористувацької комп’ютерної гри та протидії користувачу або співпраці із ним.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ПРОЕКТУВАННЯ

1.1 Огляд ігрових жанрів

Зазвичай класифікація ігрових програмних засобів відбувається за їх жанром, а також за кількістю гравців. Слід зазначити, що критерії належності ігрового додатку до того чи іншого жанру важно однозначно визначити, через те, що на сьогодні не існує остаточної та універсальної систематизації класифікації ігор, і в різних джерелах дані про жанр конкретного проекту можуть відрізнятися. Однак, існує консенсус, до якого поступово прийшли корпорації, що займаються розробкою ігрових додатків, завдяки якому належність гри до того чи іншого жанру може бути визначена однозначно. На рисунку 1.1 наведено загальну класифікацію жанрів ігрових додатків.

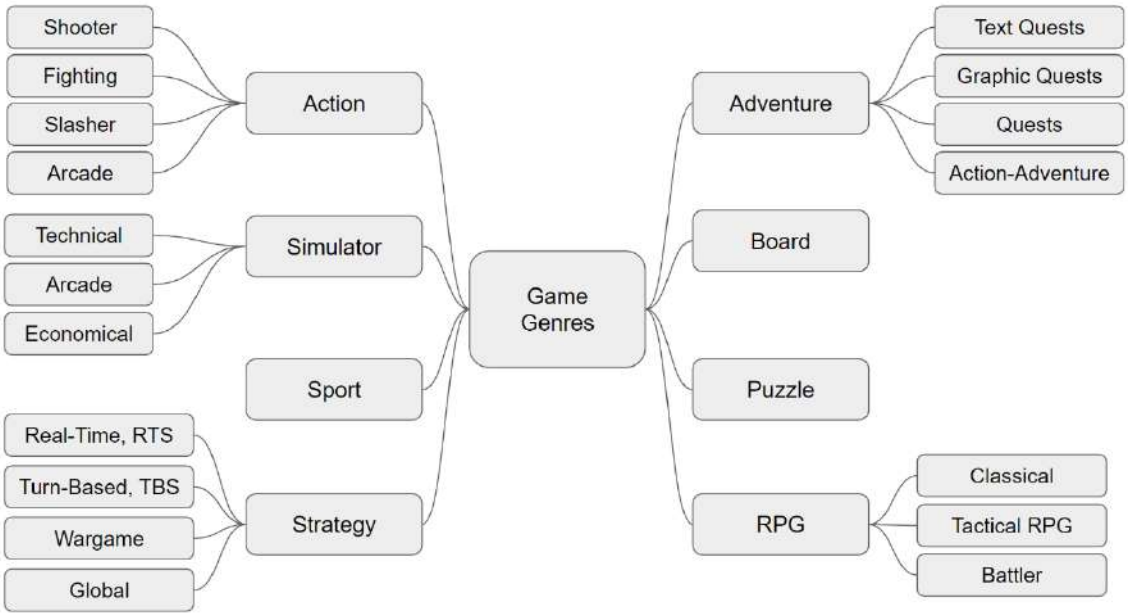

Рисунок 1.1 – Загальна класифікація ігрових жанрів

Ігровий жанр Action відрізняється тим, що гравець, як правило, діючи одноосібно, повинен знищувати за допомогою різних видів зброї різноманітних ворогів задля проходження певного рівню. Як правило. Після

досягнення заданих цілей гравець переходить на наступний рівень з іншим переліком завдань. Найбільш відомі ігри цього жанру є “Half-life”, “Doom”, “Return to Castle Wolfenstein”, “Max Payne” тощо. Окремим різновидом цього жанру є шутери від першої – First Person Shooter / FPS, або від третьої особи – Third Person Shooter, TPS. У шутерах від першої дії гравець ніби бачить ігровий процес очима свого персонажу. У шутерах від третьої особи гравець бачить персонажа збоку з фіксованої або довільної точкою зору. Найбільш відомими іграми цього жанру є “Doom”, “Quake”, тощо. Одними з найстаріших видів Action-ігор є файтинги, де геймплей складається виключно з поєдинків двох і більше супротивників із застосуванням рукопашного бою (“Mortal Kombat”, “Tekken”) та слешери - ігри від третьої особи, де основною частиною ігрового процесу є фехтувальний поєдинок із застосуванням холодної зброї. Окремим різновидом Action-ігор є аркади, тобто ігри, де гравцю доводиться діяти швидко, та покладатися насамперед на свої рефлекси та реакцію. Аркади характеризуються розвиненою системою бонусів, серед яких є нарахування очок, видавання специфічного ігрового контенту, тощо.

Головною особливістю ігрових додатків жанру Simulator є симуляція деякого процесу, де якомога повніше імітується фізична поведінка та керування будь-яким складним об’єктом технічної системи, наприклад: літаком, кораблем, автомобілем, тощо. Якщо аркадні ігрові додатки переважно фокусуються на демонстрації гравцю різних неможливих явищ, трюків та гостроти сюжету, то головний критерій якості симуляторів полягає в повноті та реалістичності моделювання його об’єкта. Спрощеним видом симуляторів є аркадні симуляції, які зазвичай оперують альтернативною фізикою ігрових процесів. Принциповою відмінністю аркадних симуляторів від аркад є саме наявність специфічної фізичної моделі. Найчастіше з такою фізичною моделлю розробляють симулятори підводних човнів та орбітальних космічних станцій. Окремим різновидом симуляторів є економічні симулятори. Де гравцеві надається можливість керувати

економічними системами різного ступеня складності, наприклад, містом (“SimCity”), острівною державою (“Tropico”), фермою (“SimFarm”), тощо. Найчастіше у якості об’єкта управління можуть виступати екзотичні системи, наприклад, мурашник або підземелля.

Особливостями жанру стратегічних ігор є те, що гра вимагає від гравця вироблення перемоги супротивника у розвитку та дослідженнях, або у військовій операції. В іграх цього жанру гравець керує не одним персонажем, а цілим підрозділом або містом, чи військовою базою. На сьогодні найбільш популярними видами ігор стратегічного жанру є покрокові стратегії (Turn-Based Strategy, TBS), де гравці по черзі роблять ходи, при цьому може передбачатися існування певних часових обмежень на виконання цього свого кроку гравцями, та стратегічні ігри в реальному часі (Real Time Strategy, RTS), де всі гравці виконують свої дії одночасно і не чекають завершення ходів один від одного. Поділ ігрового процесу на ходи у покрокових стратегіях відриває його від реального життя та позбавляє гру динаміки, внаслідок чого ці ігри користуються меншою популярністю, ніж стратегії реального часу. З іншого боку, у іграх TBS гравець має набагато більше часу на роздуми під час здійснення свого ходу, що дає можливість проявляти здібності до логічного мислення. Найбільш відомими TBS іграми є “Civilization”, “Heroes of Might and Magic”, “Galactic Civilizations”. Стратегії типу RTS з’явилися пізніше покрокових, а першою популярною грою у цьому жанрі була “Dune II” 1992 року розробки, сюжет якої заснований на популярному фантастичному фільмі. Ігри типу RTS відрізняються від TBS своєю динамікою та більше заохочують гравця до ігрового процесу. Найбільш відомими RTS іграми є “Warcraft”, “Starcraft”, “Age of Empires”, “Cossacks”, тощо.

Окремим різновидом стратегічних ігор є “Wargame”, які ведуть свою історію з шахів та шахмат. Це специфічні військові стратегічні ігри, у яких користувач від початку гри має якусь кількість ресурсів (підрозділів, окремих солдатів, військової техніки, тощо), які він поступово втрачає під час

проведення гри. Метою гри є знищення ресурсів противника допоки противник не знищив ресурси гравця. На розвиток комп'ютерних варгеймів значно вплинули настільні ігри, а також професійні військові ігри. Головною ідеєю комп'ютерного варгейма є моделювання бою від рівня взвод-рота до рівня фронт-група армій. Також варгейми, відповідно до специфіки ігрового світу, поділяють на історичні та фантастичні. Головний акцент у варгеймах роблять на: автентичності, реалістичності та історичності. Найбільш популярними варгеймами є "Total War", "Fallout Tactics", "Hearts of Iron".

Глобальні стратегії відрізняються від інших видів стратегічних ігор своїм масштабом, оскільки тут гравець керує не окремою базою чи містом, а державою. В руках гравця опиняється не лише військові справи, але й економіка, суспільні відносини, науковий прогрес, освоєння нових земель та дипломатія. У деяких видах глобальних стратегій відбувається розділення ігрових карт, де разом із глобальною використовуються тактична карта для битв, як у "Rome: Total War".

Спортивні комп'ютерні ігри, які ще мають назву "Sport Sim", відрізняються від інших тим, що вони імітують якусь спортивну гру спортивну гру реального світу, наприклад футбол, хокей, баскетбол теніс або гольф. Слід зазначити, що спортивні ігри, які імітують силові види спорту або поєдинок у якому-небудь бойовому мистецтві, типу карате, боксу, боротьби, тощо відносять до Sport-Action ігор.

Окремим ігровим жанром є Adventure та Quest які умовно поєднують у жанр пригод. Зазвичай, ігри цього жанру є оповіданнями, в яких керований гравцем персонаж просувається за сюжетом і взаємодіє з ігровим світом за допомогою застосування предметів, спілкування з іншими персонажами та вирішення логічних завдань, як у іграх "Space Quest", "Myst" та "Syberia". Текстові квести також можна умовно віднести до жанру Adventure. Вони були дуже популярні на початку розвитку індустрії персональних комп'ютерів через мале поширення графічних пристроїв відображення та нестачі ресурсів пам'яті, процесору та відеокарти комп'ютеру. Однак, із

розвитком комп'ютерної техніки, цей жанр втратив свою популярність. Головною відмінністю текстових квестів від графічних є те, що гравець взаємодіє з ігровим світом за допомогою командного рядка, а інформація про світ виводиться у вигляді текстів та малюнків із друкованих знаків. Слід зазначити, що у сучасних текстових квестах, наприклад у “Superhero league of Hoboken” графіка також активно використовується. Перші графічні квести почали з’являтися на початку 1980-х років. Серед відомих ігор цього жанру визначають “King’s Quest” та “Space Quest”.

Найпопулярнішим на сьогодні жанром квестів є Action-Adventure, який здебільшого заснований на реакції та рефлексах гравця, що притаманно жанру Action, але використовує елементи класичних квестів - предмети, взаємодія з навколишнім оточенням, текстова взаємодія із іншими персонажами ігрового світу. Відомими представниками цього жанру є ігри “Legend of Zelda” та “Resident Evil”.

Одним з найбільш популярних на сьогодні є жанр рольових ігор - Role Playing Game, RPG. Його характерними ознаками є те, що у головного героя, інших персонажів, ворогів та супротивників присутня деяка кількість ключових параметрів (умінь. характеристик, навичок), які визначають їхню силу та здібності. Зазвичай при цьому головною характеристикою ігрового персонажу є рівень, який визначає загальну силу та перелік доступних навичок або предметів екіпірування. Всі ці параметри користувач може вдосконалювати шляхом виконання завдань, знищення ворогів та тривалим використанням цих самих навичок. В іграх жанру RPG активно використовуються елементи інших ігрових жанрів, наприклад Action та Adventure. Особливостями цих ігор є дуже різноманітний, відкритий і великий ігровий світ, окрема сюжетна лінія, розгалужені діалоги із великою кількістю варіантів відповідей, велика кількість різних персонажів із своїми цілями та рисами характеру. Також серед характерних особливостей ігор цього жанру є використання гравцем великої кількості різних ігрових предметів: екіпірування, ліків, артефактів, ресурсів, тощо. Найбільш

відомими іграми цього жанру є “Diablo”, “Fallout”, “Mount and Blade”, “Dungeon Siege”, “Baldur’s Gate”, тощо.

Окремим жанром комп’ютерних та мобільних ігор є логічні головоломки та Puzzle. У звичайній головоломці роль арбітра, який спостерігає за тим. щоб гравці дотримувалися правил, відіграє сам гравець, як у грі пасьянс, або деякий механічний пристрій, наприклад, кубік Рубіка. З появою комп’ютерів можливості головоломок розширилися, оскільки створити програму простіше, ніж створити механічний пристрій. Сучасні комп’ютерні головоломки, як правило, не вимагають від гравця прикладання над зусиль або розвинутої реакції, проте у багатьох Puzzle-іграх ведеться рахунок часу, витраченого гравцем на пошук рішення. Найбільш відомими головоломками є “Сапер”, “Sokoban”. “Portal” та інш.

Традиційними або настільними Board іграми є комп’ютерна реалізація якої-небудь відомої настільної гри. наприклад, шахи, шахмати, карти, “Монополія”, або ігри серії “Warhammer”.

За кількістю гравців ігри розподіляють на поодинокі single-player та розраховані на велику кількість користувачів multi-player. Головною рисою ігрових додатків типу single-player є те. що гравець проходить гру самотужки, та зазвичай у ролі супротивника користувачу виступає комп’ютер або штучний інтелект. Ігри, що розраховані на багато користувачів multi-player принципово відрізняються від single-player необхідністю постійного підключення до виділеного серверу або високими вимогами до якості інтернету. Слід зазначити, що зазвичай у multi-player гру зазвичай одночасно можуть грати декілька десятків гравців на одній ігровій карті. Окремим видом ігор є MMO - Massively Multiplayer Online або масові ігри в інтернеті. Такі ігри передбачають велику кількість гравців, що може грати online одночасно та відрізняються від інших видів ігор великою потужністю серверної частини. Зазвичай серверна частина MMO-ігор передбачає підтримку сотень тисяч запитів користувачів в секунду. Найбільш відомими MMO-іграми є “World of Warcraft”, “RAID Shadow Legends”, тощо.

1.2 Огляд програмних засобів для розробки ігор

Через велику популярність та поширеність комп’ютерних та мобільних ігор на сьогоднішній день в світі існує велика кількість середовищ для розробки ігор, які для спрощення називають ігровими движками. Вони розрізняються своїми можливостями та областями застосування, наприклад є універсальні ігрові движки, які дозволяють створювати як мобільні, так і desktop та web-ігрові додатки. Також є спеціалізовані ігрові движки, що дозволяють створювати виключно мобільні або виключно 3D чи 2D ігри. Розглянемо найбільш популярні на сьогодні середовища для розробки ігор.

Unreal Engine - одне з найбільш поширених універсальних середовищ для розробки ігрових додатків для стаціонарних комп’ютерів, мобільних пристроїв, ігрових консолей та браузерів. В ньому можна працювати з персонажами, анімаціями,візуальними ефектами, ігровою логікою, фізикою та графікою гри. Unreal Engine як програмний засіб було розроблено компанією Epic Games для своєї гри під назвою Unreal, і після цього став окремим проектом та набув популярності як універсальний засіб для розробки ігор. Його головна перевага перед іншими ігровими движками полягає у хорошій оптимізації при великих графічних можливостях.

Unreal Engine здебільшого використовується для розробки 3D-ігор для стаціонарних комп’ютерів та консолей. Можливість розробляти 2D-ігри у ньому з’явилася відносно нещодавно. Зазвичай його обираються для розробки ігор AAA-класу, тобто високо бюджетних програмних засобів, які мають хорошу графіку та розраховані на велику аудиторію користувачів. Однак, його можна застосовувати і для розробку indie-ігор у випадку, коли розробникам важлива якісна графіка та хороша оптимізація. Інструменти для розробки мобільних ігрових додатків в Unreal Engine з’явилися пізніше, тому в цьому сегменті ігрової розробки він поступається технології Unity. Слід зазначити, що сферами застосування Unreal Engine є не тільки розробка ігор, його активно використовують в області VR/AR-розробки, а також у

кінематографі для створення анімацій та комп’ютерної графіки. За допомогою Unreal Engine створюють відеоролики для кінотеатрів, які демонструють відео з оглядом 360 градусів, а також використовують на телебаченні для накладання ефектів на відео для прямого ефіру.

Рисунок 1.2 – Інтерфейс системи Unreal Engine

Серед головних особливостей Unreal Engine можна виділити орієнтованість на 3D-розробку - оскільки він створювався для розробки саме тривимірних ігрових додатків, підтримка 2D-проектів була слабкою, хоча згодом цей недолік було усунуто. Також це середовище розробки ігрових проектів має великі можливості щодо оптимізації коду ігрового додатку, оскільки у якості програмування для Unreal Engine використовується мова програмування C++, яка дозволяє якнайкраще оптимізувати навантаження на процесор, графічну карту та процес керування оперативною пам’яттю. Окрім того, Unreal Engine має технологію Blueprints, яка була створена для полегшення процесу написання коду ігрового додатку та реалізує можливості візуального програмування без необхідності написання програмного коду

власноруч. Ця технологія дозволяє створювати програми із спеціальних візуальних блоків та зв’язків між ними, що спрощує використання Unreal технічним спеціалістам, які не займаються програмування безпосередньо.

Суттєвою перевагою Unreal Engine над іншими середовищами розробки ігор є широкі графічні можливості для створення фото-реалістичної тривимірної графіки. В Unreal велика кількість текстур, візуальних ефектів та матеріалів, які можна застосовувати до об’єктів для налаштування якісного зовнішнього вигляду. Якісна анімація, яка базується на технології Blueprints Animation, що дозволяє використовувати готові анімаційні шаблони рухів для моделей різних видів, також значно відрізняє це середовище від інших.

Серед переваг Unreal Engine також можна виділити широкі можливості налаштування штучного інтелекту для не ігрових персонажів, які використовуються в ролі ворогів або противників до користувача ігрового додатку. Для цього в Unreal використовується специфічний інструмент Behavior Trees, або дерева дій – специфічні блок-схеми. які описують поведінку персонажів гри та їх реакції на основні дії гравця. Також в Unreal є окремий модуль для створення звукових ефектів, який дозволяє перемикати аудіо-файли у залежності від особливостей сцен та дій, змінювати їх гучність, змішувати один з одним, накладати ефекти та робити інші операції із звуками. Unreal Engine має модуль Sound Cue, який створено для роботи зі звуковими ефектами ігрового додатку. На розповсюдження Unreal значно вплинула можливість безкоштовного використання, починаючи з 2015 року. У випадку, якщо гра. розроблена на Unreal Engine, буде мати великий комерційний успіх – Epic Games буде отримувати 5% від доходів у вигляді роялті. Через те, що Unreal Engine є cross-platform, він дозволяє створювати гру для будь-якої платформи та операційної системи, а також для існуючих популярних консолей та мобільних пристроїв. Також Unreal дозволяє переносити ігрові додатки з однієї платформи на іншу.

Серед головних недоліків Unreal Engine можна виділити важкість мови програмування C++ – ця мова була створена у середині 1980 років, коли ще

не існувало поширених шаблонів програмування, сучасних поглядів на ООП та концепції SOLID, яка відіграла значну роль у розвитку теорії сучасного програмування. Тому зараз мова C++ не має більшості можливостей, які мають біль сучасні мови, типу C# або Java, хоча дозволяє робити біль глибинну оптимізацію. Суттєвим недоліком цього середовища є відносно високі системні вимоги до обчислювальної системи – через велику кількість можливостей роботи з графікою система Unreal Engine вимагає великої кількості ресурсів від комп'ютера. на якому вона буде застосовуватися.

Окрім того, велика ціна додаткових модулів у Unreal Engine також не сприяє його поширенню – це середовище є магазин, де можна придбати додаткові компоненти для розробки ігрових додатків, готові ефекти, 3D-моделі і т.д. Деякі розробники критикують цей магазин через занадто великі ціни та співвідношення ціна-якість.

Технологія Unity на сьогоднішній день є домінуючою технологією розробки ігрових мобільних додатків. Система Unity дозволяє розробляти cross-platform програмні засоби, які мають 2D та 3D графіку. Це середовище розробки ігрових додатків підтримує професійну платну ліцензію та аматорську ліцензію, яка дозволяє створювати невеликі за масштабами додатки безкоштовно. Однак у безкоштовній версії Unity значно скорочена кількість підтримуваних платформ для перенесення між операційними системами та платформами свого проекту. Велику популярність та поширеність система Unity отримала через простоту свого інтерфейсу. Зручність у використанні Unity не несе негативного впливу на функціональні можливості та забезпечує можливості роботи не тільки для професійних програмістів, а й для дизайнерів, спеціалістів з тестування та аніматорів. Робочій простір додатку Unity представлений вікнами Scene – вікно на якому розробник створює та розташовує ігрові об'єкти, Hierarchy – вікно, у якому розробник формує ієрархії та об'єднання ігрових об'єктів задля полегшення їх подальшого керування, Game – вікно, в якому розробник може побачити гру так, як її буде бачити гравець, Inspector – вікно, у якому розробник може

налаштовувати ігрові об’єкти сцени, додавати та керувати ігрові компоненти та Console – вікно, у якому розробник може спостерігати які помилки та повідомлення відбуваються під час запуску ігрового додатку.

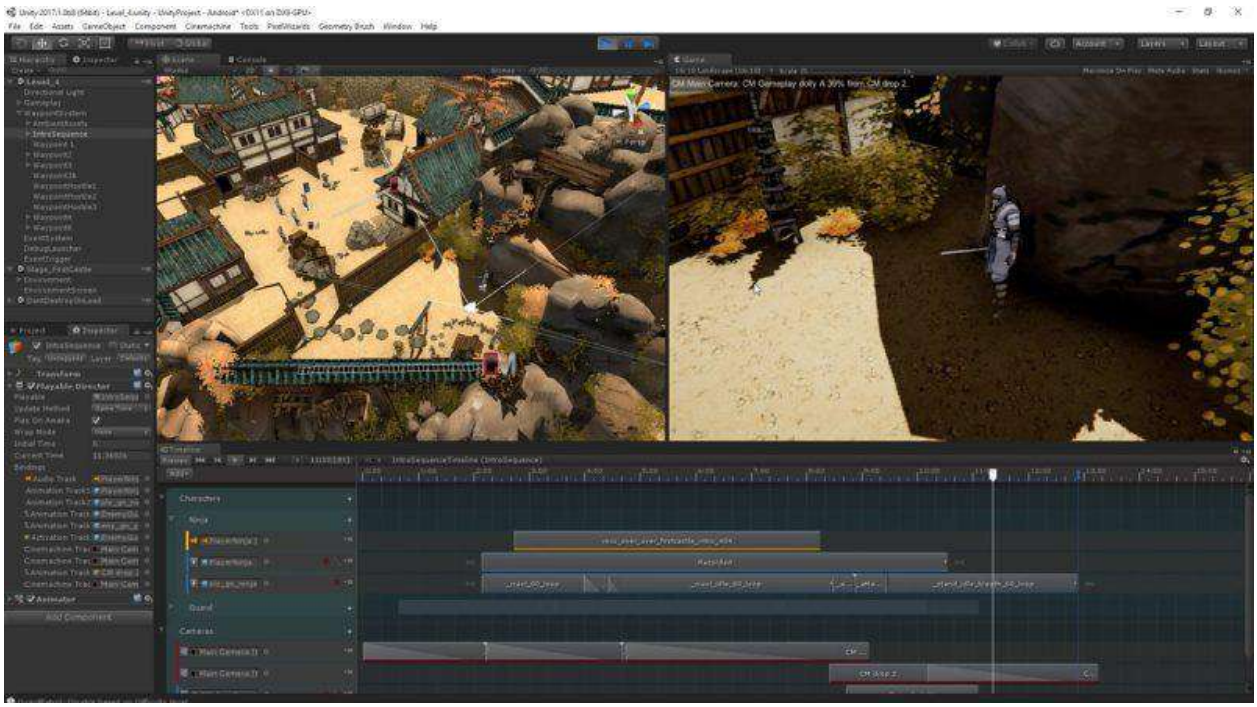

Рисунок 1.3 – Інтерфейс системи Unity3D

Середовище Unity підтримувало декілька популярних мов програмування – JavaScript та C#, але останні версії Unity більше налаштовані на роботу з мовою C#. Окрім того, Unity має свій повноцінний фізичний движок, за роботу якого відповідає технологія NVIDIA PhysX. Також цей ігровий движок надає багато можливостей щодо створення та налаштування анімації, формування шейдерів, налаштування матеріалів та текстур, роботу зі спрайтовими та 3D анімаціями, звуковими ефектами, тощо. За своїми базовими можливостями Unity не поступається Unreal Engine, але ця технологія зручніша у дослідженні та розробці ігрових додатків через використання більш сучасної мови програмування C#. Серед недоліків Unity можна виділити менші можливості щодо роботи з графікою, ніж у Unreal Engine, а також менші можливості щодо оптимізації та моніторингу навантаження ігрового додатку на обчислювальну систему.

Технологія CryEngine також є однією з найбільш поширених технологій розробки ігрових додатків. Це cross-platform движок, який дозволяє створювати проекти не тільки для стаціонарних комп'ютерів, але й для мобільних пристроїв, браузерів та ігрових консолей. Цей ігровий движок розповсюджується безкоштовно, але після подолання грою визначеного максимально допустимого прибутку передбачається оформлення платної підписки на розробку. Зовнішній вигляд інтерфейсу користувача середовища розробки ігрових додатків CryEngine представлено нижче.

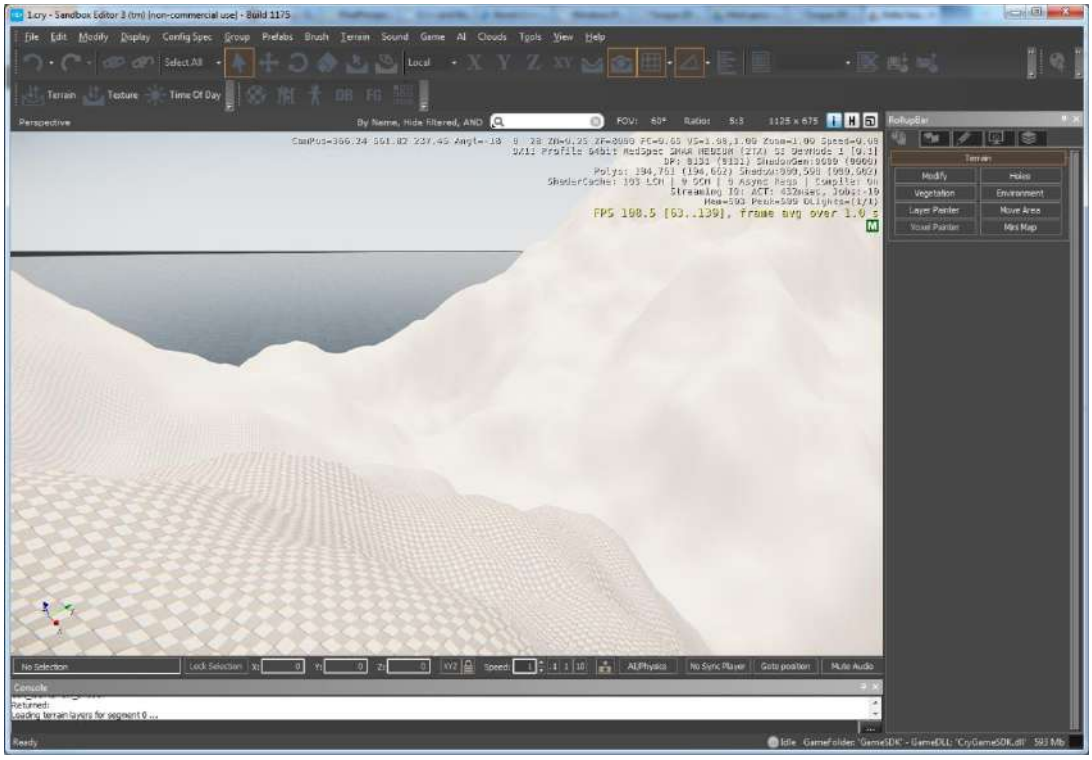

Рисунок 1.4 – Інтерфейс системи CryEngine

У програмному засобі CryEngine додано простий візуальний редактор ігрових елементів, який дозволяє розробнику побачити, як саме ігрові об'єкти будуть виглядати для кінцевого споживача. Він забезпечує можливості зручного редагування графічних елементів, імпорт моделей з популярних 3D-редакторів Maya та 3Ds Max. Система розробки коду ігрового додатку на CryEngine сформована на Lua, який використовувався у популярних іграх "STALKER" та "World of Warcraft". Для оптимізації

апаратного навантаження на графічний процесор передбачено застосування технології Geometry Instancing. Окрім того, серед переваг CryEngine можна виділити широкі можливості реалізації штучного інтелекту та систему звукового супроводу ігор, cross-platform, використання тривимірного конструктору об’єктів SandBox, підтримка фізики та гарна масштабованість. Серед недоліків CryEngine визначають відносно високий поріг входження у розробку, проблеми супроводу “безкоштовних” користувачів та менший рівень поширеності, порівняно з Unity та Unreal Engine, що визначається у недостатньому рівні документування багатьох можливостей движка.

Серед найбільш поширених технологій, що використовуються для розробки ігрових додатків можна виділити Godot. Godot – безкоштовна платформа для розробки ігрових додатків з відкритим початковим кодом. Цей програмний засіб було розроблено у 2007 році аргентинськими програмістами. На теперішній час Godot є найбільш популярним некомерційним ігровим движком, який підтримується та розробляється виключно за кошти волонтерів та пожертви інді-розробників. Він надає можливості розробки ігрових додатків у 2D та 3D режимах. Особливістю цього ігрового движка є те, що замість псевдо 2D-режиму, коли тривимірний ігровий світ представляється у двох вимірах - платформа працює у реальному 2D-просторі, який виражено у множині пікселів, що значно спрощує можливості розробки ігрових додатків.

Серед переваг цього ігрового движка можна визначити гарну оптимізацію для розробки 2D-ігор, спрощену систему роботи із текстурами та можливості підтримки піксель-арту, великі можливості щодо підтримки різних мов програмування: GDScript, C#, C++, а також можливості візуального програмування без використання специфічних мов. Окрім того значною перевагою цього програмного засобу є гнучка система налаштування анімації об’єктів та якісна система підтримки розробників за рахунок великого обсягу технічної документації про можливості розробки на

цьому ігровому движку. Зовнішній вигляд інтерфейсу користувача середовища Godot представлено нижче.

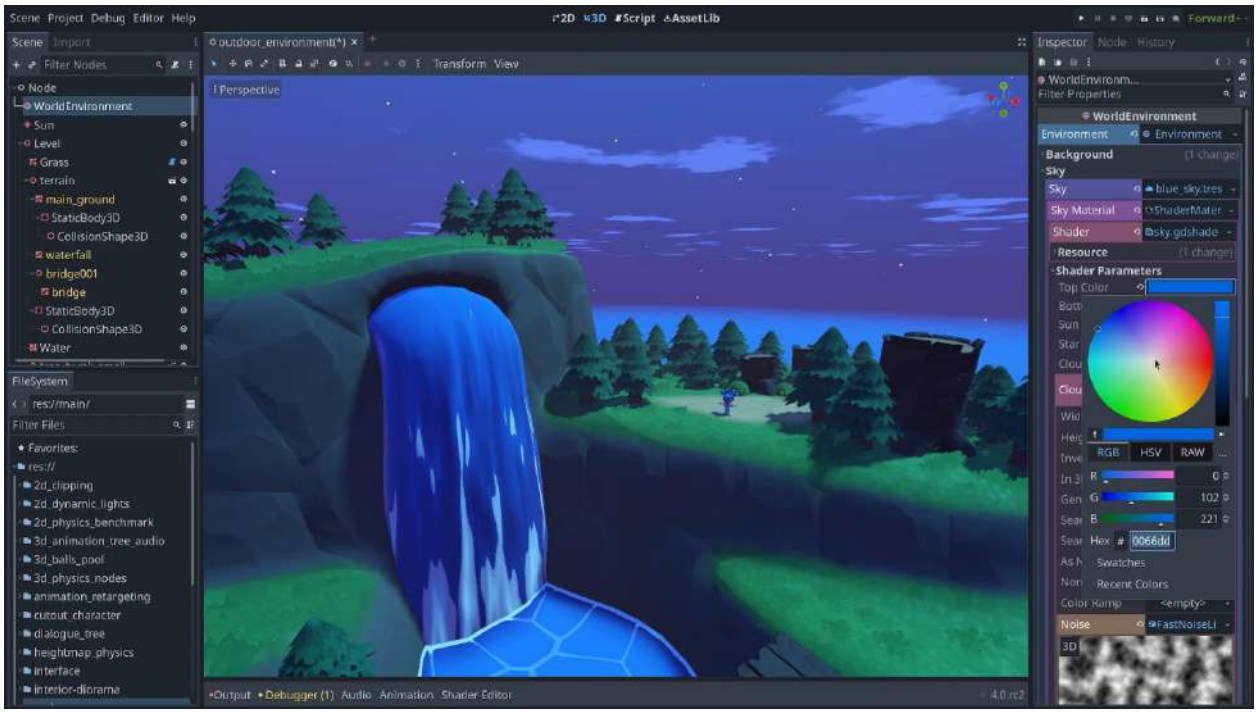

Рисунок 1.5 – Інтерфейс системи Godot

Серед недоліків Godot визначають недостатньо високу якість фізики для 2D-ігрових додатків, порівняно із Unity та Unreal Engine, деякі труднощі із програмним переміщенням ігрових об’єктів на сценах, а також відсутність можливості формувати консольні збірки проекту.

1.3 Вибір мови програмування та засобів розробки

Серед поширених мов програмування найбільш поширеними на сьогоднішній день є C++, C#, Java та JavaScript. Слід зазначити, що незважаючи на те, що саме Java є природною мовою для написання додатків під мобільні пристрої на платформі Android, вона не підтримується жодною поширеною технологією розробки ігрових додатків, описаною раніше. Відповідно до цього, у випадку обрання Java у якості основної мови програмування ігрового додатку – використання багатьох корисних

технологій, які надають розробникам поширені ігрові движки, буде неможливе. Це саме стосується і мови програмування JavaScript, яка дуже поширена для розробки web-додатків, але не підтримується жодним з поширених зараз ігрових движків. Тому ці мови програмування виключаються з переліку мов для написання сучасних ігрових додатків.

Мова C++ є однією з найстаріших мов програмування, вона була створена на початку 1980 років і досі не втратила своєї актуальності. Ця мова мала дуже великий вплив на розвиток багатьох мов програмування, оскільки вона була однією з перших об’єктно-орієнтованих мов програмування. Великою перевагою цієї мови є те, що її використовують технології Unreal Engine, CryEngine та Godot, що значно спрощує розробку ігрових додатків.

Мова програмування C# є повністю об’єктно-орієнтованою, на відміну від мови C++. Ця мова була розроблена у 2001 році корпорацією Microsoft як подальший розвиток мови C++, насиченої шаблонами розробки та великою кількістю бібліотек та можливостей швидкого написання коду. Вона має такі функції як сувора типізація, заборону множинного успадкування, можливості використання events та data bindings, повне об’єктно-орієнтоване програмування, а також реалізує можливості компонентно-орієнтованого програмування і автоматичне визначення та прибирання з пам’яті непотрібних об’єктів. Використання цієї мови програмування надає розробникам можливості швидкого отримання прототипу та його деталізації для отримання остаточного варіанту архітектури програмного засобу. Це робить розробку на C# більш перспективною для розробки невеликих за своїми функціональними можливостями ігрових додатків, ніж розробка мовою C++. За темпами розробки мова C# значно переважає C++ за рахунок вбудованих шаблонів розробки та концепцій ООП та SOLID і поступається їй виключно у можливостях оптимізації. Крім того, C# є базовою мовою програмування для технології Unity та Godot, що надає додаткових можливостей при розробці ігрових додатків. Слід зазначити що обидві мови можуть застосовуватися для написання cross-platform програмних засобів,

але за цим показником C++ значно переважає мову C#, хоча у випадку розробки на Unity це компенсується можливостями Unity у формування збірок проекту під різні платформи та операційні системи.

Значною перевагою C++ є можливість оптимізації unmanaged коду, що надає максимальної ефективності при використанні ресурсів обчислювальної системи, де буде запускатися ігровий додаток. Отже при вирішенні складних завдань, пов’язаних з обробкою великих обсягів даних, використання мови C++ показує значно вищі результати, ніж використання мови C#. Але варто зазначити, що при обиранні старого стилю програмування часткового ООП або у випадку не високого рівня майстерності володіння C++ ця перевага може бути повністю нівельована витоками пам’яті та виникненням критичних помилок. З точки зору зручності читання коду мова C++ значно поступається C#. Також C++ не має можливості зручного створення events та data bindings, і передбачає самостійну реалізацію розробником design-pattern Observer та інших шаблонів розробки, які реалізовані у C# за замовчанням. Також серед переваг C# перед C++ є її менша технічна складність за рахунок відсутності вказівників. що значно пришвидшує процес розробки програмних засобів. Тому для реалізації ігрового додатку у межах цієї роботи було обрано саме мову програмування C#. Оскільки мовою C# можна писати ігрові програмні засоби за допомогою технологій Unity та Godot – питання вибору ігрового движка також значно спрощується. Слід зазначити, що технологія Godot значно поступається Unity у можливостях роботи з тривимірною графікою, фізикою, візуальними і звуковими ефектами, а також іншими можливостями, які мають значний вплив на якість ігрового додатку. Окрім того, технологія Unity дозволяє створювати робочі версії ігрових додатків для різних платформ та операційних систем, що суттєво покращує можливості перенесення проекту на іншу платформу чи операційну систему, та масштабування програмного засобу. Тому у якості технології розробки було ігри було обрано саме технологію Unity та мову програмування C#.

1.4 Біологічні принципи функціонування систем штучного інтелекту

Обрана для розробки ігрового програмного засобу технологія Unity не має розвинутої системи організації штучного інтелекту для керування персонажами в ігрових додатках різних жанрів. Тому для вирішення цієї задачі можна використати поширені біологічні принципи організації систем штучного інтелекту. Серед них виділяють штучні нейронні мережі, генетичні алгоритми, або алгоритми мурашнику та штучні імунні системи.

Штучні нейронні мережі імітують діяльність мозку хребетних, зокрема мозку людини. Завдяки цьому вони здатні до самонавчання та самоорганізації. Незважаючи на велику кількість моделей побудови та функціонування штучних нейронних мереж вони мають спільні риси, що відрізняють їх з інших біологічно-орієнтованих систем організації інтелектуального прийняття рішень. Всі моделі штучних нейронних систем оперують. так само, як і мозок людини, одним загальним типом робочих елементів - нейронів. При цьому, незалежно від обраної моделі штучної нейронної мережі, побудову штучного нейрона завжди намагаються відтворити відповідно до побудови нейрону людини. Сьогодні методи та алгоритми. що функціонують на основі апарату штучних нейронних мереж, показують високі результати при вирішенні задач розпізнавання образів, класифікації, кластеризації, інтелектуального прийняття рішень, прогнозування, розпізнавання рукописного тексту, спілкування із людьми, розпізнавання різних мов та звукових повідомлень. Формування нових принципів побудови штучних нейронних мереж, заснованих на використанні вирішальних правил з кінцевими вибірками може призвести до отримання значних результатів при вирішенні широкого спектру практичних задач.

Штучні нейронні мережі оперують штучними нейронами, кожен з яких уособлює в собі спрощену модель біологічного нейрона. Головними функціями, які реалізує штучний нейрон є прийняття сигналів з багатьох входів та їхньої обробки єдиним чином разом із подальшим переданням

результату цієї обробки наступним нейронам з іншими вирішальними правилами. Так само, як і біологічні, штучні нейрони мережі пов’язані між собою аксонами, а місця їх перетину визначають як синапси. Саме у синапсах відбувається посилення або ослаблення електрохімічного сигналу, який шириться нейронною мережею. Відповідно до цього зв’язки між штучними нейронами мережі називають синоптичними. Головним параметром синапсу є ваговий коефіцієнт, відповідно до якого відбувається та чи інша форма зміни сигналу та інформації під час її передавання між нейронами. Завдяки цієї особливості функціонування синапсів, вхідна інформація обробляється та формується у вигляді остаточного результату.

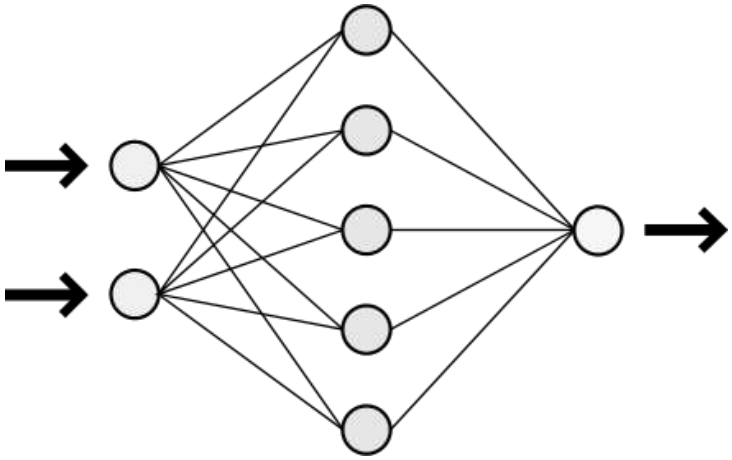

Рисунок 1.6 – Загальний вигляд штучної нейронної мережі

Слід зазначити, що нейронні мережі побудовані з декількох шарів нейронів, які постійно обмінюються між собою сигналами через свої синапси (рисунок 1.6). Відповідно до цього. нейрони першого, або вхідного шару отримують інформацію від зовнішніх сенсорів системи, обробляють її та передають результат у нейрони наступного шару. Нейрони наступного шару обробляють отриману інформацію та передають результат у нейрони наступного шару і так далі. Слід зазначити, що всі шари, окрім шару, що приймає інформацію зовні та шару, що видає сформований результат роботи є прихованими. Така проста нейронна мережа здатна до навчання та може формулювати прості рішення та знаходити зв’язки між даними та

алгоритмами їх обробки. Важливою особливістю є те. що кількість прихованих шарів мережі може змінюватись динамічно.

Генетичні алгоритми, так само, як і нейронні мережі відносяться до еволюційних видів формування та прийняття рішень. Зазвичай їх використовують для пошуку рішення в задачах з великою кількістю потенційних результатів та задачах з великою комбінаторною складністю. Принцип роботи еволюційних алгоритмів заснований на еволюційній теорії Чарльза Дарвіна. Слід зазначити, що зазвичай рішення у генетичному алгоритмі представляється у формі бінарної послідовності. Цей спосіб застосовується коли змін одного біта в рішенні може призводити до незначної зміни цільової функції.

У випадку вирішення завдань на зв’язаних графах рішенням є маршрут переміщення по графу у вигляді деякої послідовності його вершин. Таким чином зміна однієї вершини у сформованому маршруті може завдати кардинальних змін щодо обчислення значення цільової функції. Тому операції проводяться із послідовностями цілих чисел, які позначають номери вершин, а функцією пристосованості алгоритму є довжина цього маршруту. Робота генетичного алгоритму передбачає використання оператору мутації задля формування випадкових або псевдо-випадкових змін у послідовності вирішення завдання. При цьому оператор мутації крім безпосередньої зміни хромосоми об’єкту дослідження виконує декілька додаткових дій: видалення випадкового вузла з маршруту, включення випадкового вузла у випадкове місце маршруту, зміна місць двох випадкових вузлів у маршруті. Окрім використання оператору мутації, генетичні алгоритми користуються оператором кросовера, який виконує функцію формування нового маршруту на основі частин декількох інших маршрутів. Слід зазначити, що оператор кросовера не відрізняється високою ефективністю при роботі з маршрутами, які мають значні відмінності у переліку своїх вершин.

Однією з проблем, характерних для генетичних та еволюційних алгоритмів є проблема зациклення у процесі пошуку рішення у локальних

максимумах. При цьому рішення, знайдене у такому локальному максимумі, поступово витісняє всі інші варіанті розв’язання задачі в популяціях об’єктів. Задля вирішення цієї проблеми використовують паралельні потоки пошуку, які можуть взагалі не перетинатися між собою під час своєї роботи.

Алгоритми мурашнику засновані на моделюванні взаємодії між мурахами однієї колонії. Такі алгоритми демонструють високу ефективність у вирішенні комбінаторних задач та задач на графах, в тому числі й задачу комівояжера. При цьому алгоритми мурашнику оперують колонією як складною розподіленою децентралізованою системою. де кожна окрема мураха виконує прості одноманітні дії та володіє мінімальною інформацією та можливостями взаємодії з іншими мурахами колонії та невеликими ділянками зовнішнього середовища. Незважаючи на це, вся система як колонія мурах, може вирішувати складні задачі щодо оптимізації маршрутів та знаходити дуже близькі до оптимальних рішення складних логістичних задач, шляхом адаптації до зовнішніх умов, що динамічно змінюються.

Слід зазначити, що взаємодія між мурахами колонії відбувається за допомогою обміну інформацією прямими та непрямими шляхами. Під час прямого обміну даними відбувається безпосередній контакт між мурахами, а під час непрямого відбувається зміна зовнішнього середовища одним з мурах колонії та розпізнавання цієї зміни іншими. Моделювання алгоритмів мурашнику зазвичай відбувається саме завдяки непрямого обміну між окремими об’єктами колонії. Під час непрямого обміну мурахи розпилюють специфічний феромон, запах якого залишається на ґрунті деякий час, який можуть розрізняти інші мурахи колонії.

На початковому етапі роботи алгоритму мурахи обирають свої напрямки пересування випадковим чином задля досягнення мети - пошуку їжі, позначаючи специфічними феромонами свої шляхи. Після чого, мурахи, що досягли мети повертаються назад раніше всіх своїм шляхом, збільшивши вміст феромонів на цьому маршруті. Після декількох ітерацій з усієї кількості маршрутів переміщення мурах буде обрано один найкращій маршрут, який

має найбільшу кількість феромонів. У випадку, якщо на цьому шляху з’являється перешкода - мураха випадковим чином обирає інший шлях та формує новий маршрут до їжі. Таким чином через деякий час формується інший оптимальний маршрут до цілі. Слід зазначити, що алгоритми мурашника набагато швидше адаптуються до змін зовнішнього світу, ніж генетичні алгоритми.

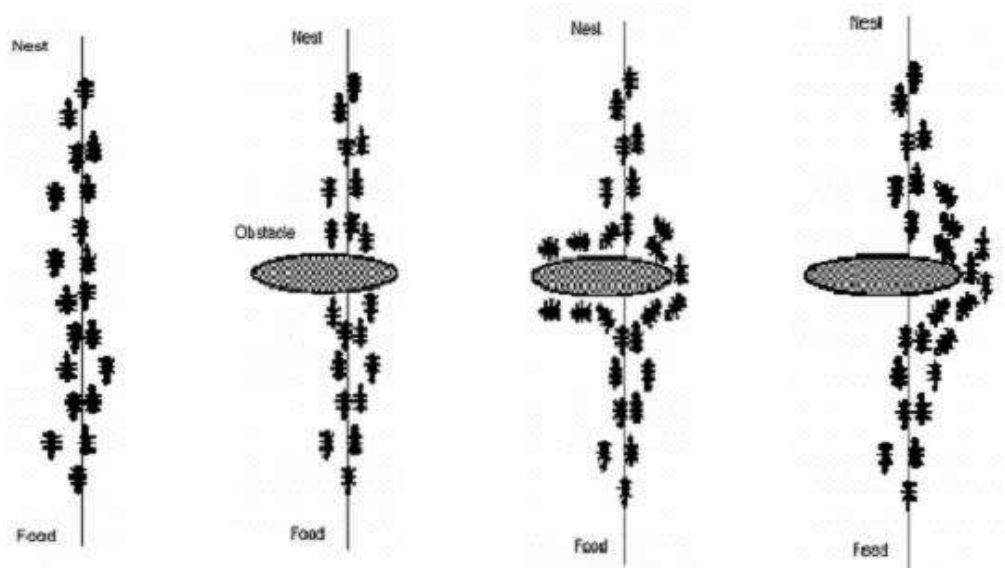

Рисунок 1.7 – Приклад роботи алгоритму мурашнику

Штучні імунні системи, так само, як і штучні нейронні системи моделюють поведінку біологічних систем функціонування організму людини. Головними елементами штучних імунних систем є антитіла, які можуть взаємодіяти між собою під час вирішення складної практичної задачі, а також з антигенами, як конкретними проявами зовнішнього світу з яким взаємодіє імунна система. На сьогоднішній день існує декілька основних моделей штучних імунних систем, які відрізняються між собою способом побудови та принципами взаємодії між об’єктами системи. У деяких моделях взаємодія між антитілами виключається повністю (модель клонального відбору), у деяких - така взаємодія відіграє більшу роль, ніж взаємодія з антигенами (модель негативного відбору). Завдяки своїй розподіленості та можливості до самонавчання штучні імунні системи успішно

використовуються для вирішення практичних завдань будь-якої складності, здебільшого пов’язаних із класифікацією та кластеризацією даних, розпізнаванням образів і рукописного тексту, розпізнаванням речі людини, прогнозуванням, тощо. Здатність штучної імунної системи до самонавчання визначається особливостями процесу формування імунної відповіді на появу антигенів та особливостями саморегуляції системи.

Так само, як і генетичні алгоритми, штучні імунні системи оперують переліком специфічних імунних операторів під час своєї роботи. Серед таких операторів можна визначити оператори представлення антигенів, клонування, мутації, селекції клонів, супресії мережі, позитивного відбору, негативного відбору, досягнення критерію зупинки імунного навчання, тощо. Головка особливість роботи штучної імунної системи полягає в тому, що її антитіла можуть взаємодіяти із зовнішніми антигенами. Характер цієї взаємодії вимірюється значенням афінності між цими об’єктами. У деяких випадках одного значення афінності для прийняття рішення щодо визначення класу або кластеру для окремого антитіла буває недостатньо, тому використовують поняття афінності до популяції об’єктів, наприклад зовнішньо-популяційна афінность або внутрішньо-популяційна афінность. Також задля пришвидшення роботи системи може відбуватися зменшення кількості імунних об’єктів, до яких відбувається визначення афінності – тоді оперують поняттям авідності антитіла до деякої групи об’єктів, сере яких можуть бути як антигени, так і антитіла. Після презентації об’єктів антитілам за рахунок клонування, мутації та селекції клонів або супресії мережі імунна система намагається відтворити параметри одного з визначених антигенів задля набуття стану повної специфічності (подібності) до нього. Завдяки цьому з усіх можливих варіантів прийняття рішень тієї чи іншої практичної задачі обирається те, що характеризується специфічним антитілом, або найбільшою афінность до одного з переліку ймовірних рішень.

1.5 Складання вимог та постановка задачі проектування

Метою роботи є розробка моделі штучної імунної мережі для керування ігровими персонажами у комп’ютерній грі. Відповідно до зазначеної мети виникає необхідність вирішення наступних завдань:

1 аналіз завдання, пошук необхідної інформації;

2 проектування та реалізація ігрового додатку на основі одного з сучасних архітектурних шаблонів розробки – MVI;

3 проектування та реалізації моделі штучної імунної мережі для прийняття рішення щодо поведінки персонажу у грі;

4 налаштування імунних операторів для оптимізації процесу керування персонажами гри;

5 адаптація реалізованої моделі штучної імунної мережі у гру для керування персонажами, враховуючи особливості ігрового дизайну.

2 ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ ТА ОСОБЛИВОСТІ ІГРОВОГО ДИЗАЙНУ

2.1 Принципи SOLID та їх вплив на розробку програмних засобів

Принципи SOLID було сформовано наприкінці 2000 років як розвиток ідей об’єктно-орієнтованого програмування. Ці концепції оказали значний вплив на розвиток технологій програмування та формування архітектури програмних засобів. Само по собі слово SOLID можна перекласти з англійської як “твердий” та може бути використано задля підкреслення того факту, що код написано згідно із загальноприйнятими стандартами та правилами, а також одним стилем.

Суть об’єктно-орієнтованого програмування полягає в тому, що будь-який код уособлює концепцію взаємодії між окремими інформаційними об’єктами, які мають свою впорядкованість, тобто належать до якихось класів, які у свою чергу, поєднуються у ієрархію успадкування. Принципи ООП набули швидкого розповсюдження через те, що завдяки ним розробникам програмних засобів стало значно легше реалізовувати великі та складні проекти великими командами розробки, моделювати алгоритми та керувати великими обсягами інформації.

Значний вклад в формування SOLID зробив Роберт Мартін - один з найбільш досвідчених програмістів сьогодення, який почав програмувати ще в 1970 роках та став першим, хто зібрав всі ці концепції у одну збірку і почав розповсюджувати їх серед розробників. Термін SOLID був вигаданий Майклом Фізерсом - відомим автором книг по програмуванню. Слід зазначити, що концепції SOLID є абстрактними сутностями, які не мають жорстких зв’язків із будь-якою технологією, платформою або мовою програмування. Концепції поєднані в SOLID носять виключно рекомендаційних характер для розробників з метою покращення якості їх проектів, що особливо важливо для крупних програмних засобів.

Принцип єдиної відповідальності - Single-Responsibility Principle, або SRP (літера S з абревіатури SOLID) означає, що будь-який тип даних повинен мати одну зону відповідальності і повністю реалізовувати її. Наприклад, у випадку, якщо в коді додатку є клас із декількома методами, один з яких підготовляє дані, а інший - записує їх - такий клас порушує принцип SRP та має бути розділений на два класи, кожен з яких має виконувати свою функцію. Використання концепції SRP призводить підвищення якості коду та зменшення ймовірності виникнення конфліктів злиття гілок початкового коду додатку, які виникають у випадку одночасної розробки проекту кількома розробниками.

Принцип відкритості-закритості - Open-Closed Principle, або OCP (літера S абревіатури SOLID) означає, що типи додатку можна розширювати за рахунок наслідування або extension-методів, але їх не можна модифікувати безпосередньо. Будь-які зміни у класах можливі лише за умови додавання нової функціональності або виправлення помилок, знайдених під час тестування програмного засобу. Таким чином, відповідно до цього принципу будь-які програмні сутності мають бути відкритими для розширення шляхом додавання нової функціональності, але при цьому закриті для зміни. Такі обмеження обумовлені зменшенням кількості можливих помилок, які можуть виникати в результаті частих змін функціональності великих проектів та знизити вартість розробки в цілому. Використання цього принципу на практиці призводить до зменшення обсягів роботи спеціалістів з тестування програмних засобів через те, що їм не потрібно проводити регресивне тестування всього додатку після кожного внесення змін розробниками.

Принцип підстановки Барбари Лісков – Liskov Substitution Principle, або LCP (літера L абревіатури SOLID) стосується доцільності та логічності успадкування типів даних. Відповідно до нього у випадку заміни будь-якого базового класу класом-спадкоємцем логіка роботи програмного засобу не має бути пошкодженою. Цей підхід допомагає виявляти проблемі абстракції та приховані зв’язки між сутностями, а також робить поведінку модулів більш

передбачуваною та вводити обмеження на успадкування. Принцип Барбари Лісков стверджує, що поведінка типів-спадкоємців не може мати протиріч до поведінки їх базових типів, а сутності, які використовують батьківський тип мають працювати так само, як і базові з дочірніми класами. При цьому в роботі програмного засобу нічого не має ламатися. Таким чином, клас-спадкоємець може доповнювати поведінку свого базового типу, а не заміщувати її своєю реалізацією.

Принцип розділення інтерфейсів – Interface Segregation Principle, або ISP (літера I абревіатури SOLID) стосується зменшення кількості методів / відповідальностей в інтерфейсах. Суть цього принципу в тому, що замість того, щоб формувати великі універсальні абстракції - краще створювати велику кількість спеціальних інтерфейсів, які декларують невелику кількість методів. Таким чином ця концепція перекликається із концепцією SRP для класів. Під час успадкування тип-спадкоємець може отримати велику кількість непотрібної функціональності, яка не буде в ньому використовуватися. Задля запобігання цієї ситуації інтерфейси необхідно розділити за їх функціональними можливостями. Серед переваг концепції ISP визначають: зниження залежностей між модулями, зменшення непотрібного функціоналу, який не потрібно реалізовувати у типах-спадкоємцях, під час внесення правок у функціональність типу даних звертаються тільки до потрібних частин, а не до всіх залежних модулів.

Принцип інверсії залежностей – Dependency Inversion Principle, або DIP (літера D абревіатури SOLID) декларує ідею того, що програмні модуля, які знаходяться на більш високих рівнях архітектури проекту не повинні залежати від модулів, які знаходяться на низьких рівнях архітектури проекту, а можуть залежати лише від деяких абстракцій. При цьому деталі типу не мають впливати на абстракції, скоріш абстракції мають впливати на деталі. Окрім того, цей принцип описує один з найважливіших принципів ООП – модулі верхніх рівнів не мають залежати від модулів нижніх рівнів. Обидва типи можуть залежати лише від абстракцій.

Слід зазначити, що використання принципів SOLID в розробці гарантує легке оновлення. розширення, внесення змін та усунення помилок програмного засобу. Це відбувається за рахунок того, що код додатку буде легко читати та розуміти розробникам.

2.2 Компонентно-орієнтована архітектура Unity-проектів

Однією з особливостей ігрових додатків, створених за допомогою технології Unity є широке використання сцен ігрових об’єктів. Сцени використовуються для відображення користувачу якого-небудь аспекту гри, наприклад сцени головного вікна, сцени битви із противниками, сцени покращення параметрів героя або бази користувачем. Інколи сцени використовуються для розміщення інформації щодо ігрових рівнів, які проходить користувач в процесі використання ігрового додатку. Будь-яка сцена Unity-проекту зберігає в собі ієрархії ігрових об’єктів. Слід зазначити, що в Unity ігровий об’єкт представляє той чи інший об’єкт, яким безпосередньо керує гравець, або з яким він буде взаємодіяти впродовж певного періоду користування ігровим додатком. Для представлення ігрового об’єкту у коді використовується специфічний тип даних GameObject. Особливістю ігрових об’єктів в Unity є те, що вони не можуть взаємодіяти між собою безпосередньо, а також мають дуже обмежені функціональні можливості. Всі функції та можливості взаємодії між об’єктами, а також можливості візуального відображення 3D-моделей разом з текстурами та матеріалами, реалізуються за допомогою компонентів, які додаються до ігрових об’єктів сцени та редагуються у спеціальному вікні Unity.

Таким чином, ігрові об’єкти фактично є носіями компонентів, кожен з яких реалізує ту чи іншу функціональність. Без компонентів ігрові об’єкти неможливо визначити на сцені та вони майже не несуть ніякої користі, окрім можливості формування ієрархій.

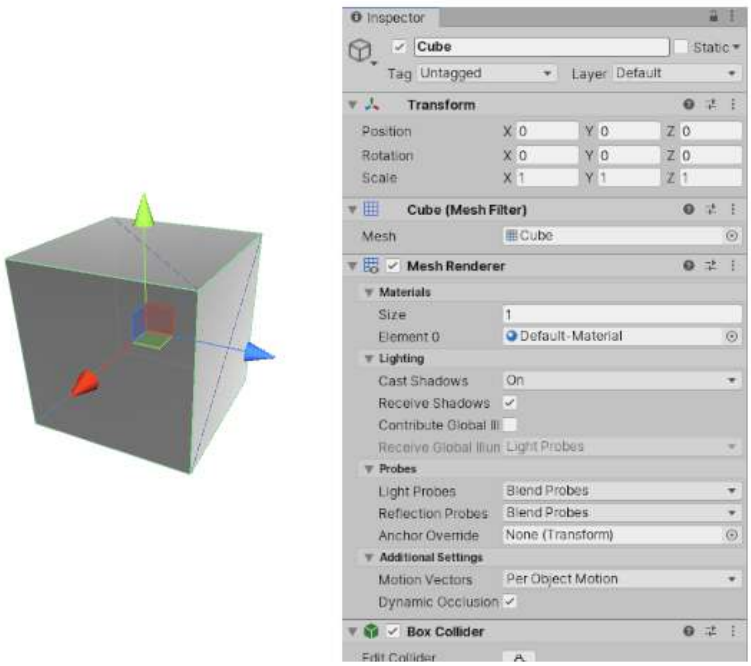

Рисунок 2.1 – Ігровий об’єкт та його компоненти

Компонент Transform є обов’язковим компонентом для будь-якого ігрового об’єкту сцени, оскільки він не тільки зберігає інформацію щодо положення цього об’єкту на сцені, його кутів та масштабу, але й дозволяє формувати складні ієрархії об’єктів та визначати роль того чи іншого об’єкту в ієрархії. Розміщення ігрових об’єктів на сценах відбувається в залежності від налаштування вимірності 2D та 3D. Відповідно до цього змінюється й кількість ознак, які характеризують положення, кути та масштаб об’єкту. Слід зазначити, що кожна одиниця в значенні координат X, Y або Z співвідноситься зі значенням 1 метр реального світу.

Також одним з найпоширеніших компонентів, які використовуються ігровими об’єктами в Unity є MeshRenderer, який додається до об’єкту разом з компонентом MeshFilter. Слід зазначити, що під поняттям Mesh в Unity розуміють 3D-модель, яка буде відображатися на сцені. Сам по собі MeshFilter дозволяє обрати 3D-модель, яка буде відображатися ігровим об’єктом з переліку ресурсів проекту, а компонент MeshRenderer дозволяє накласти на цю модель матеріал із обраною текстурою, які також мають зберігатися у якості ресурсів Unity-проекту. Крім того MeshRenderer

відповідає ще й за вплив джерел світла на ігровий об’єкт, процес візуалізації тіней, застосування того чи іншого шейдеру для візуалізації, тощо.

Задля налаштування фізичних параметрів ігрового об’єкта використовують специфічні компоненти типу Collider. Система Unity використовує велику кількість colliders, для поширених форм 3D-моделей, наприклад сфери, куби, капсули, поверхні, тощо. Ці компоненти створені для визначення точок перетину мешів ігрових об’єктів, а також для унеможливлення фізичного проходження одного ігрового об’єкту скрізь іншого. За допомогою colliders можна налаштовувати реакції об’єктів на випадки перетинів поверхонь, що буде мати вплив на візуалізацію сцени.

Зазвичай разом із colliders в Unity також використовують компонент RigidBody, який дозволяє налаштовувати фізичні властивості поверхні ігрового об’єкту та може використовувати специфічний фізичний матеріал. За допомогою цього компоненту зазвичай налаштовують гнучкість та пружність ігрових об’єктів.

У випадку використання програмного коду об’єктами ігрової сцени в Unity передбачається специфічний компонент Script. Цей компонент дозволяє приєднати до ігрового об’єкту файл з C#-кодом, з якого можна викликати ту чи іншу функціональність впродовж ігрового процесу. Задля забезпечення таких можливостей будь-який script в Unity-проекті успадковує клас MonoBehaviour.

Крім безлічі специфічних компонентів, якими можуть користуватися розробники, Unity забезпечує можливість роботи з шаблонами створення об’єктів, які уособлюють деяку ієрархію ігрових об’єктів разом із переліком компонентів та специфічними налаштуваннями. Такими шаблонами в Unity є Prefabs. За допомогою prefabs можна статичним чи динамічним чином створювати складні анімаційні об’єкти, які можуть користуватися scripts та взаємодіяти між собою під час ігрового процесу. Prefab є специфічний ігровий об’єкт, який відображається на сцені, але може виступати у якості шаблону за яким додаток буде створювати однотипні об’єкти під час

вирішення тих чи інших задач. Цей підхід дозволяє економити час на динамічному формуванні об’єктів і заснований на використанні фабричного методу або інших шаблонів створення об’єктів.

2.3 Особливості архітектурного патерну MVI

Розробка сучасних програмних засобів базується на використанні архітектурних шаблонів розробки, які визначають загальні правила додавання, реалізації, видалення та фіксації функціоналу. На сьогоднішній день існує велика кількість шаблонів розробки, які можна використовуватися для розробки звичайних та клієнт-серверних додатків. Серед цих шаблонів окремою групою можна виділити так звані MVx шаблонів, які призначені для створення класичних додатків або клієнтської частини клієнт-серверного програмного засобу. Базовим шаблоном розробки архітектури, з якого почався розвиток цих трирівневих шаблонів є Model-View-Controller, який було сформовано наприкінці 1970 років. Незважаючи на велику поширеність на популярність, цей архітектурний шаблон було створено до того, як було запропоновано design-patterns на початку 2000 років. Відповідно до цього, у класичному варіанті MVC не було реалізовано events, data bindings та багатьох інших design-patterns, які використовуються для автоматизації низки процесів обміну даними між рівнями архітектури додатку.

Архітектурний шаблон Model-View-Intent з’явився як усучаснена модифікація шаблонів Model-View-Controller та Model-View-Presenter. Концепція цього шаблону побудови архітектури програмних засобів була сформована у 2019 році в блогосфері серед Java та JavaScript розробників. Інколи цей архітектурний шаблон визначають як повернення розвитку MVx до класичної концепції MVC, але з використанням events, bindings, з урахуванням концепцій SOLID та інших сучасних підходів та методів розробки програмних засобів.

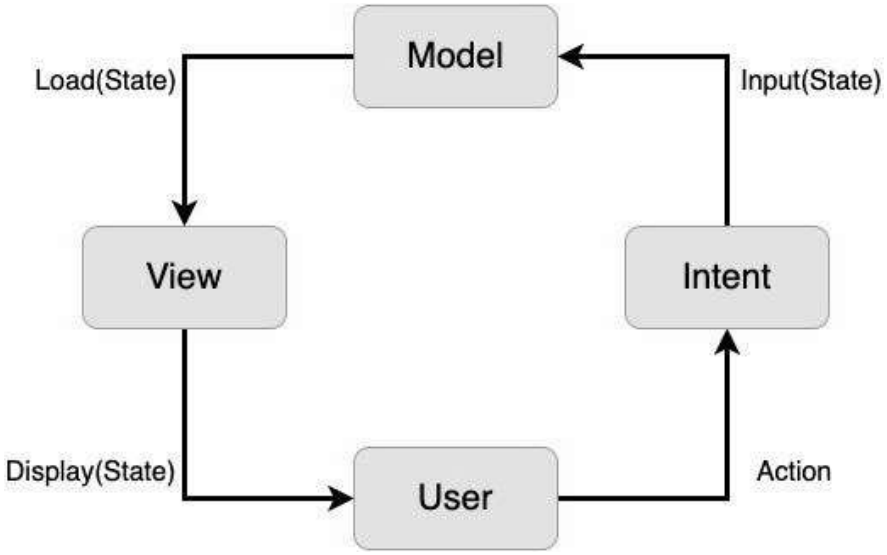

Рисунок 2.2 – Загальна схема роботи шаблону MVI

Цей архітектурний шаблон передбачає використання рівня намірів користувача щодо виклику тієї чи іншої функціональності, який реалізується як функціональна сутність Intent. Головними особливостями MVI є розділення програмного засобу на три (в окремих варіаціях - чотири) рівня: Model, View, Intent, State.

На рівні Model знаходиться реалізація основних типів даних та бізнес-логіки проекту. В цьому модель MVI схожа на варіацію MVC з активною моделлю. В деяких модифікаціях модель є незмінною, а також вона використовує перелік станів, які описують параметри та поточні значення типів моделі після відпрацювання того чи іншого наміру – Intent користувача. Слід зазначити, що на цьому рівні не тільки представляються типи даних, які описують інформацію про користувача та службові дані проекту, але й більшість функціональних можливостей програмного засобу. Модель відповідає на запити Intents, виконує у відповідь на ці запити свої методи, які формують новий змінений стан моделі та передають його на рівень View за допомогою станів State. Також цей рівень представляє методи встановлення та отримання своїх даних, а також перевірки вхідних значень, які можуть викликати Intents. Архітектура MVI передбачає те, що будь-який тип даних рівня Model може взаємодіяти з будь-яким типом з рівня Intent.

Модель в цьому архітектурному шаблоні повинна мати децентралізовану розподілену структуру, що повністю відповідає принципу SRP концепції SOLID. Окрім того рівень Model доступний для типів даних з рівня View тільки на читання змінених даних.

Рівень View використовується для забезпечення взаємодії з користувачем. так само як і в інших MVx архітектурних шаблонах розробки. На цьому рівні представлено перелік діалогових вікон, елементів керування користувача, вікна для інформування користувача та інші елементи інтерфейсу, з якими буде взаємодіяти користувач. Слід зазначити. що цей рівень не відповідає за перевірку вхідних даних, які вводить користувач під час взаємодії із програмним засобом. Типи, представлені на цьому рівні можуть безпосередньо взаємодіяти з events, викликаючи їх методи, а також отримувати інформацію зі станів зміненої моделі. Також архітектура MVI передбачає можливість оновлення даних, які відображаються користувачу за допомогою набору діалогових вікон, за рахунок підписки на events типів даних, реалізованих на рівні Model. Крім того, типи даних рівня View не можуть мати бізнес-логіки, окрім можливостей перемикання діалогових вікон інтерфейсу користувача.

Рівень Intent реалізує наміри користувача або програмного засобу для виконання специфічних команд щодо зміни стану моделі. Цей рівень повинен мати розподілену структуру, що відповідає принципам SOLID, тобто для кожної окремої команди формується окремий intent. Також рівень Intent надає елементам діалогових вікон з рівня View можливість формувати та викликати intents на виконання або їх методи для спроби зміни стану моделі, або для виклику деякої функціональності з моделі, яка буде обгорнута методами intent. Слід зазначити, що деякі модифікації архітектурного шаблону MVI передбачають формування абстрактних фабрик на рівні intents, фабричних методів або інших варіантів creational patterns, які використовуються для створення специфічного intent. Але intents також можуть бути реалізовані у вигляді низки класів, що організовані по принципу

singletons або статичних типів.

Окремої уваги у архітектурному шаблоні MVI виділяють стани моделі або states. Відповідно до цього кожен state представляє локалізовану частину даних зміненої в процесі ви конання intent моделі. Таким чином за своїм функціональним призначенням state подібні до контекстів архітектурного шаблону MVVM. Відповідно до цього states реагують на events від змінених моделей та формують новий стан того чи іншого типу у моделі. Кожен окремий state має доступ до рівня View та може взаємодіяти із діалоговими вікнами та елементами інтерфейсу користувача. Слід зазначити, що states або зовсім не мають функціоналу, або мають мінімальний набір методів, які використовуються виключно для отримання даних діалоговими вікнами з рівня View. Важливою особливістю MVI є можливість існування декількох states для однієї моделі.

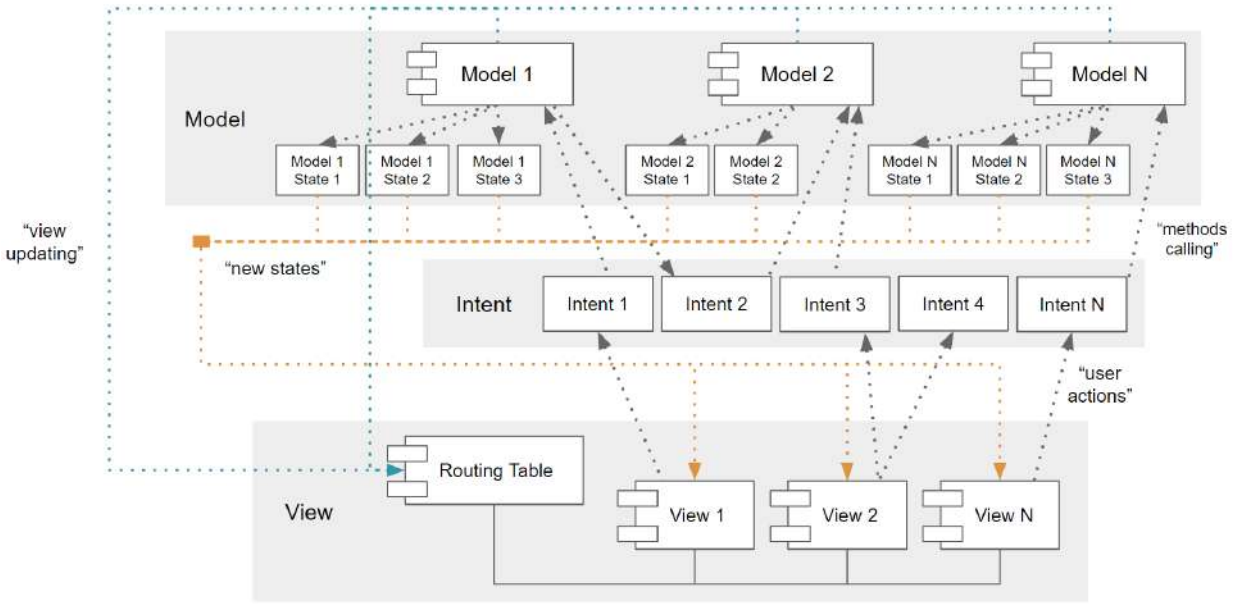

Рисунок 2.3 – Деталізована схема роботи шаблону MVI

Серед переваг архітектурного шаблону MVI можна виділити простоту реалізації та розгортання, що дозволяє спростити процес залучення нових технічних спеціалістів до команди розробки, а також процес їхньої технічної адаптації на проекті. Через децентралізовану структуру рівнів MVI забезпечує можливості роботи над одним проектом великої команди

технічних спеціалістів з низькою ймовірністю виникнення merge-конфліктів. Крім того цей шаблон розробки достатньо простий та інтуїтивно зрозумілий, оскільки базується на базових принципах MVC.

2.4 Штучні імунні системи та задача керування персонажами в іграх

На сьогоднішній день теорія штучних імунних систем користується великою науково-дослідницькою увагою та використовується для вирішення великої низки практичних задач - від задач класифікації та кластеризації даних до захисту інформації, прогнозування та розпізнавання голосових повідомлень. Тому серед існуючих систем штучного інтелекту теорія штучних імунних систем займає важливе місце поряд із генетичними алгоритмами, нейронними мережами, тощо.

Так само, як і більшість систем інтелектуальної обробки інформації та прийняття рішень, теорія штучних імунних систем користується специфічним термінологічним апаратом, який оперує поняттями лімфоцитів, антитіл, антигенів, фагоцифів, афінностей. авідностей, тощо. при цьому антитілами вважаються об’єкти самої імунної системи, якими вона оперує з метою формування специфічної імунної відповіді на розпізнавання антигенів, які представляють об’єкти зовнішньої середи (бактерії, віруси, тощо). Слід зазначити, що під специфічної імунною відповіддю розуміють такий стан імунної системи, коли кожне антитіло шляхом клонування та мутації відтворило параметри одного з зовнішніх антигенів та набуло специфічності до нього. Теорія штучних імунних систем оперує поняттям афінності, як мірою подібності антитіл та антигенів в багатовимірному просторі ознак. В залежності від особливості імунної моделі афінності можуть визначатися не тільки між антитілами та антигенами, але й між антитілами однієї популяції. Таким чином, на основі значення афінності в штучних імунних системах визначається, наскільки антитіло подібне до того чи іншого антигену або

антитіла системи. Значення афінності знаходиться у 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.

Серед найбільш поширених імунних моделей штучних імунних систем, які використовуються для вирішення наукових та практичних задач можна виділити модель клонального відбору, модель штучної імунної мережі, модель позитивного відбору та модель негативного відбору антитіл.

Слід зазначити, що модель клонального відбору є однією з найперших імунних моделей, основні засади якої було описано ще у 1959 році дослідником М. Берентом, який є автором терміну “клональний відбір”. В основі цього принципу лежать особливості функціонування В-лімфоцитів (антитіл) після визначення присутності чужорідних об’єктів імунною системою. Це передбачає перетворення популяції антитіл шляхом їх клонування, мутації та відбору клонів після проведення мутації задля створення нової популяції антитіл, що мають більші афінності до антигенів. За рахунок поступових перетворень в популяціях антитіл відбувається досягнення стану специфічності антитіл до антигенів з метою розпізнавання їх структури та особливостей задля того, щоб у випадку наступного потрапляння подібного антигену в імунну систему процес його розпізнавання був максимально швидким. Цей процес і називають процесом формування специфічної імунної відповіді системою на зовнішні подразники. Слід зазначити, що у випадку, якщо антитіло за рахунок поступового клонування, мутації та клонального відбору досягло стану специфічності до одного з антигенів - воно перестає приймати участь в процесі формування імунної відповіді. Принцип роботи моделі клонального відбору представлений нижче та умовно розподіляється на декілька основних етапів.

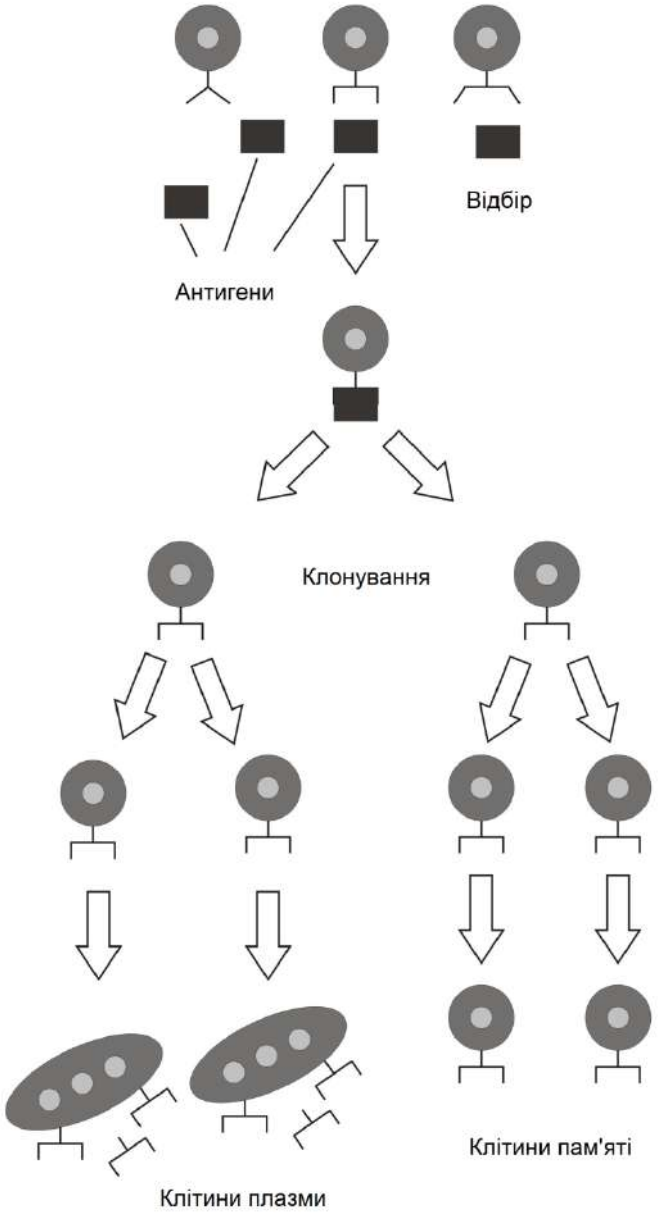

Рисунок 2.4 – Принцип роботи моделі клонального відбору

Відповідно до цього після визначення системою факту присутності антигенів відбувається визначення афінностей між антитілами системи та чужорідними антигенами. Після цього запускається процес формування специфічної імунної відповіді за рахунок клонування, мутації та клонального відбору до тих пір, поки не буде сформована специфічна популяція антитіл. На сьогоднішній день найбільш поширеними методами, що функціонують на основі моделі клонального відбору є Clonalg та BCA. Ці методи відрізняються один від одного особливостями клонування та специфікою

процесу відбору клонів після проведення мутації. Слід зазначити, що ключовою особливістю моделі клонального відбору, яка суттєво відрізняє її серед інших поширених імунних моделей є те, що вона не передбачає взаємодії між антитілами системи в процесі формування імунної відповіді. Це означає, що афінності визначаються лише між антитілами та антигенами, але не визначаються безпосередньо в популяції антитіл. Така особливість значно звужує області практичного використання цієї моделі. Головною відмінністю між методами Clonalg та BCA під час клонування є те, що Clonalg оперує всією популяції антитіл, тобто відбувається популяційне клонування, яке вимагає від обчислювальної системи значних обсягів оперативної пам’яті для розміщення всіх клонів. У методі BCA клонування відбувається послідовно, що значно зменшує навантаження на обчислювальну систему, де працює цей метод. Окрім того, Clonalg передбачає популяційну мутація та популяційний відбір клонів, в ході якого серед всіх антитіл та їх клонів обирається об’єкти, які характеризуються високими значеннями афінностей до зовнішніх антигенів. На відміну від цього, в методі BCA ці етапи відбуваються послідовно, окремо для кожного антитіла системи. В цілому, модель клонального відбору користується значною популярністю через простоту реалізації та модифікації, на відміну від інших моделей штучних імунних систем.

Наступною за популярністю імунною моделлю є модель штучної імунної мережі, яка була запропонована В. Ерне наприкінці 1970 років. Слід зазначити, що головною відмінністю цієї моделі від моделі клонального відбору є можливості взаємодії між імунними об’єктами всередині популяції антитіл, а не тільки між антитілами та антигенами. Через це, модель штучної імунної мережі надає більше можливостей та використовується для вирішення більш складних практичних та дослідницьких задач, ніж інші імунні моделі. Головним недоліком цієї моделі є важкість її реалізації та модифікації. Особливістю цієї моделі можливість взаємодії між клонами та антитілами під час формування специфічної імунної відповіді. Принцип

роботи цієї моделі базується на гіпотезі про те, що лімфоцити можуть взаємодіяти між собою, утворюючи загальну мережу.

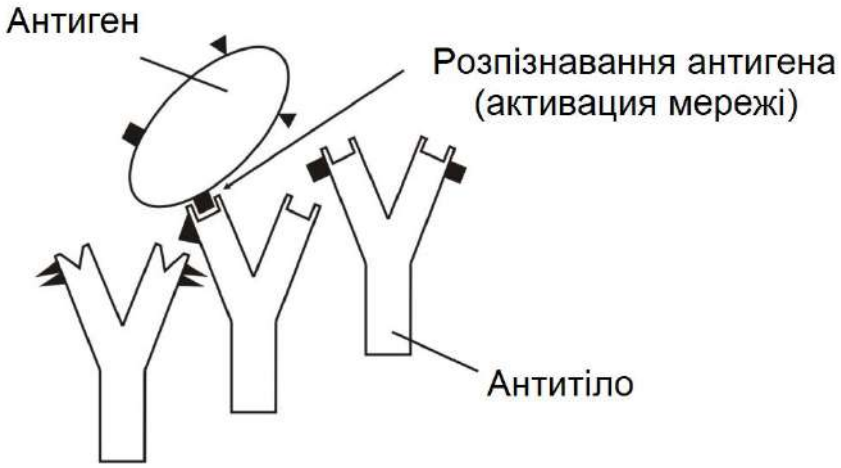

Рисунок 2.5 – Принцип взаємодії антитіл імунної мережі

Встановлення факту наявності антигенів призводить до стимуляції роботи імунної мережі. В результаті цього низка антигенів, які мають високі афінності до виявлених антигенів формує множину клонів в процесі формування специфічної імунної відповіді. Так само, як і в моделі клонального відбору сформовані клони підлягають мутації, під час якої відбувається зміна їх ознак задля набуття стану специфічності до деякого антигену. Важливою відмінністю моделі штучної імунної мережі від моделі клонального відбору є процес редагування популяції імунних об’єктів, яка утворюється шляхом поєднання початкових антитіл мережі та сформованої множини клонів. Цей процес в моделях штучної імунної мережі реалізується завдяки механізму супресії. Під час супресії стискання мережі до початкової кількості антитіл шляхом видалення змінених під час мутації клонів та початкових антитіл, які характеризуються низьким значенням афінності до антигенів, а також низьким значенням афінності до антитіл, які набули стан специфічності до деякого зовнішнього антигену. Серед існуючих методів, організованих на основі теорії штучних імунних мереж можна виділити методи aiNET, opt-aiNET та RLAIS. Ці методи зазвичай використовуються

для вирішення задач класифікації. кластеризації та розпізнавання образів, але після деякої модифікації можуть застосовуватися для вирішення інших практичних задач.

Моделі негативного та позитивного відборів відрізняються своєю вузькою спеціалізацією та зазвичай використовуються для вирішення специфічних задач, пов'язаних з організацією захисту від комп'ютерних мереж та файлів від вірусів. Ці моделі функціонують на основі принципів поділу і розпізнавання клітин системи та зовнішніх антигенів, з якими контакту система під час формування специфічної імунної відповіді. Методи негативного відбору шляхом клонування та мутації формують набори антитіл, з яких видаляються ті об’єкти, які характеризуються високими значеннями афінності до клітин самої імунної системи, і низькими до чужорідних антигенів. Найбільш відомою є робота Форреста зі створення імунних методів, які використовуються для виявлення аномалій. В результаті цього був розроблений один методів негативного відбору, який отримав назву NSA.

Задля спрощення реалізації та модифікації різних імунних методів та моделей, їх розподіляють на перелік специфічних функцій, які називають імунними операторами. Під імунним оператором розуміють функціональний елемент імунного методу рішення деякої практичної задачі, який виконує специфічні операції над набором імунних об’єктів – антитіл, або популяцією зовнішніх антигенів. Слід зазначити, що серед імунних операторів виокремлюють універсальні оператори від спеціалізованих. Відповідно до цього, універсальними імунними операторами є оператор презентації, оператор клонування, оператор мутації, а спеціалізованими є оператори клонального відбору, супресії імунної мережі, позитивного та негативного відбору антитіл. Тобто, універсальні імунні оператори використовуються більшістю існуючих моделей та методів, а спеціалізовані – тільки окремою імунною моделлю чи методом. Незважаючи на це, саме імунні оператори оказують найбільший вплив на швидкодію імунного алгоритму та точність

обчислень і прийняття рішень під час вирішення практичних задач.

Задача керування поведінкою персонажів в ігрових програмних засобах може розглядатися як окремий випадок задачі класифікації. Де можливі варіанти дій ігрового персонажу формують набір класів, які представляються популяцією антигенів. При цьому ознаками класу є декілька параметрів, які мають специфічні незмінні значення. Ці параметри беруться з обмеженого переліку характеристик, які визначають поведінку ігрового персонажу. Так шляхом клонування, мутації, взаємодії з іншими персонажами та супресії мережі для кожного окремого персонажу обирається тип поведінки.

2.5 Особливості роботи імунних операторів в моделі aiNET

Імунний метод aiNET є одним з найбільш поширених методів, які функціонують на основі моделі штучної імунної мережі. Саме цей метод частіше за все використовують задля вирішення задач автоматичної класифікації з неконтрольованим навчанням, прогнозування та розпізнавання символів та тексту. На швидкість та точність роботи цього методу суттєво впливають його імунні оператори, серед яких можна виділити оператор представлення популяції антигенів антитілам імунної системи – $Presentation(AB, AG)$, а також презентації антитіл системи між собою задля обчислення мережевих коефіцієнтів; оператор клонування антитіл – $Cloning(AB, CL)$, який дозволяє створювати тимчасові об’єкти для мутації ознак під час формування специфічної імунної відповіді мережею; оператор мутації – $Mutation(CL)$, який дозволяє формувати безліч змін для тимчасових клонів, сформованих для кожного окремого антитіла під час клонування; оператор супресії мережі – $Supression(AB, CL)$, який контролює розмір популяції антитіл та не дає імунній мережі витрачати надмірну кількість пам’яті на зайві непотрібні для подальшої роботи імунні об’єкти; оператор перевірки можливості припинення процесу формування специфічної імунної відповіді – $Termination(AB, AG)$, який дозволяє зупиняти процес

безперервного клонування, мутації та супресії імунних об’єктів мережі та визначає умови завершення процесу імунного самонавчання. Слід зазначити, що час формування імунної відповіді має лінійну залежність від розміру популяції антитіл та популяції антигенів, які представляють класи поведінки ігрових персонажів та формують навчальну вибірку.

Під час роботи оператору $Presentation(AB, AG)$ для кожного антитіла мережі відбувається обчислення зовнішніх та внутрішніх афінностей. Під терміном зовнішня афінность розуміють афінность між антитілом та антигеном, а термін внутрішня афінность розуміють як афінность між антітіло та іншим антитілом штучної імунної мережі. Таким чином, на цьому етапі процесу формування специфічної імунної відповіді кожне антитіло визначає афінності до кожного антигену та кожного антитіла з обох популяцій імунних об’єктів. Через те, що цей процес займає багато часу – оператор представлення антигенів антитілам використовується тільки один раз за весь час формування імунної відповіді системою.

Оператор клонування антитіл $Cloning(AB, CL)$ використовується з метою створення множини клонів для кожного антитіла, яке не набуло стану специфічності до деякого антигену. Для імунних алгоритмів найбільш поширеними є статичне та пропорційне клонування. У випадку статичного клонування – кількість клонів, яка формується для кожного антитіла, не залежить від його афінностей до антигенів і є вхідним параметром імунного алгоритму. У випадку використання пропорційного клонування кількість клонів антитіла пропорційно залежіть від його афінностей з одним або декількома цільовими антигенами. Слід зазначити, використання надмірно великої або малої кількості клонів може мати однаково негативний вплив на швидкість процесу формування специфічної імунної відповіді.

Оператор мутації $Mutation(CL)$ відіграє велику роль в процесі формування специфічної імунної відповіді мережею на виявлення зовнішніх антигенів. Це відбувається через те, що саме він забезпечує набуття стану повної або умовної специфічності між клонами антитіла та антигенами

завдяки тому, що робота даного оператора призводить до зміни ознак клонів, що може наблизити їх до ознак того чи іншого антигену. За принципом роботи відрізняють декілька видів оператора мутації: статична, пропорційна та зворотно-пропорційна мутація. При використанні оператору статичної мутації зміни ознак клону антитіла відбуваються у межах одного чітко вказаного діапазону значень, який використовується у якості вхідного аргументу. У випадку використання пропорційної мутації зміни ознак клону відбуваються пропорційно його афінності до одного або групи антигенів:

$$\mu = rand(0{,}0;\ aff_{ij}], \tag{2.2}$$

де μ – коефіцієнт мутації, а aff_{ij} – значення афінності між i антитілом, від якого було сформовано клон для подальшої мутації, та j антигеном.

Особливістю зворотно-пропорційної мутації є те, що зміни ознак клонів збільшуються у випадку низького рівня його специфічності до одного або групи цільових антигенів, та зменшуються у випадку високого рівня специфічності. Відповідно до цього визначення коефіцієнту мутації відбувається наступним чином:

$$\mu = rand(0{,}0;\ 1 - aff_{ij}]. \tag{2.3}$$

Задля підвищення швидкості формування імунної відповіді може використовуватися направлена мутація. У більшості існуючих імунних алгоритмах використовується випадкова мутація, тобто після визначення коефіцієнту мутації параметри клону або збільшуються, або зменшуються випадковим чином. Особливістю направленої мутації є те, що рішення про збільшення або зменшення ознаки клону приймається після порівняння ознак цього клону та ознак одного з цільових антигенів або антитіл.

Оператор супресії мережі $Supression(AB, CL)$ дозволяє скоротити кількість антитіл та змінених під час мутації клонів до початкового рівня.

Головною умовою високої швидкості класифікації є зменшення кількості імунних об’єктів для яких необхідно проводити клонування, мутацію та презентацію, але за рахунок формування нових антитіл під час клонування відбувається значне збільшення кількості імунних об’єктів у мережі. Отже робота цього оператору дозволяє скоротити кількість антитіл до тієї кількості, яка була до початку клонування. Відповідно до цього, для кожної групи, яку утворює клоноване антитіло та його клони, відбувається вибір одного імунного об’єкту, який в подальшому буде використовуватися алгоритмом. Таким чином, з кожної такої групи буде обрано по одному антитілу, що має найбільший рівень афінності до антигенів. За принципом роботи відрізняють декілька видів оператора супресії: популяційний та цільовий. У випадку популяційної супресії кожному антитілу або клону, ознаки якого було змінено під час мутації, представляється вся популяція антигенів, тобто до кожного з існуючих антигенів цей клон визначає афінность. З цих афінностей потів визначається авідність як середня афінность антитіла чи клону до всієї популяції антигенів. Після співставлення авідностей в кожній групі антитіла та його клонів обирається один імунний об’єкт, що має найбільший рівень авідності, який і замінює антитіло, від якого він був клонований. У випадку цільової супресії антитілам та клонам представляється не вся популяція, а тільки їх цільові антигени, тобто вони визначають афінності не до всіх антигенів, а до їх невеликої кількості, що визначається на початку роботи імунного алгоритму. Головною умовою обрання імунного об’єкту, який замінить клоноване антитіло є максимальна афінность до одного з цільових антигенів, або набуття стану специфічності до одного з них.

Оператор $Termination(AB, AG)$ використовується для перевірки можливості зупинки процесу формування імунної відповіді мережі на антигени (імунного навчання). За принципом роботи відрізняють декілька видів оператору зупинки формування імунної відповіді: статичний, критеріальний, повний та комбінований.

Використання статичного оператору зупинки формування імунної відповіді використовується значення максимальної кількості поколінь антитіл, які формуються за рахунок клонування, мутації та супресії. Ця кількість поколінь визначається перед початком роботи імунного алгоритму та використовується у якості його вхідного аргументу.

Використання критеріального оператору зупинки передбачає досягнення кожним антитілом мережі стану умовної специфічності до групи цільових антигенів або до окремого антигену. Це унеможливлює вплив на зупинку процесу імунного навчання, що може призвести до збільшення часу на класифікацію об’єктів.

Особливістю повного оператору зупинки імунного навчання є досягнення стану повної специфічності між кожним антитілом імунної мережі та одним з антигенів. Це також унеможливлює вплив на зупинку процесу формування імунної відповіді, що призводить до збільшення часу класифікації.

Комбінований оператор зупинки імунного навчання передбачає досягнення повної або умовної специфічності між кожним антитілом мережі та одним або групою антигенів, та використання максимальної кількості поколінь імунних об’єктів у якості вхідного аргументу для алгоритму класифікації. Цей аргумент використовується у якості запобіжника від нескінченного формування популяцій антитіл та збільшення часу класифікації об’єктів.

2.6 Особливості дизайну ігрового додатку

Ігровий додаток моделює футуристичний космічний світ, де гравець взаємодіє з іншими персонажами у вільному ігровому просторі. Гравець, так само як і інші ігрові персонажі з автоматичним керуванням керує власним космічним кораблем, який може переміщуватися просторами ігрового світу, знаходити та видобувати ігрові ресурси, обмінювати їх у визначених точках

карти ігрового світу або шляхом торгівлі з іншими персонажами задля розвитку власного корабля. На карті ігрового світу передбачено декілька локацій, де гравець разом із іншими персонажами може обмінювати зібрані ресурси, ремонтувати свій корабель та підвищувати його рівень та характеристики. Крім того на карті ігрового світу періодично генеруються місця потенційного видобутку ігрових ресурсів різних видів. Серед видів ігрових ресурсів, якими може керувати гравець та інші ігрові персонажі можна визначити наступні види:

- енергія;
- крипто-валюта;
- руда;
- метал;
- кристали.

Енергія є головним ресурсом, який витрачають космічні кораблі гравця та ігрових персонажів з автоматичним керуванням під час переміщення між локаціями карти ігрового світу. Також цей ресурс використовується задля відновлення міцності корабля після битв з ігровими персонажами. Окрім того, енергія витрачається космічними кораблями на постріли під час битви. Слід зазначити, що космічні кораблі у грі не можуть самостійно генерувати енергію, і можуть купувати її у центрах обміну ресурсів.

Крипто-валюта є універсальним ресурсом для обміну ресурсів у грі, ремонту космічного корабля та підвищення його рівня. Цей вид ресурсу можна отримувати лише на пунктах обміну ресурсами. обмінюючи на неї руду, метал або кристали. За крипто-валюту на точках обміну ресурсів гравець та ігрові персонажі, які керують іншими кораблями, можуть придбати енергію для своїх кораблів.

Руда є найменш цінним ресурсом у грі, який може видобувати гравець та інші ігрові персонажі на своїх космічних кораблях. Руду можна знайти на спеціальних рудовищах на карті ігрового миру. Будь-який космічний корабель під керуванням гравця, або персонажу, може збирати руду з

рудовища до її повного вичерпання. Зібрану руду можна продати за крипто-валюту або перетворити на метал, ціна якого буде значно вище.

Метал є ресурсом, який можна отримати тільки за рахунок переплавлення зібраної руди космічним кораблем гравця або кораблями інших ігрових персонажів. Виплавляння металу з руди потребує великої кількості енергії та самої руди, але цей ресурс приносить гравцям значно більшу кількість крипто-валюти, ніж звичайна руда.

Кристали є найбільш рідким та цінним ресурсом, який можна зібрати на рудовищі разом зі звичайною рудою. Імовірність знаходження кристалів дуже невелика, тому цей ресурс цінується найбільше. Кристали можна обміняти на крипто-валюту в точках обміну ресурсів.

Таким чином гравець разом з іншими космічними кораблями можуть переміщатися картою ігрового світу, винаходити нові рудовища, збирати звідти руду, продавати її на пунктах обміну ресурсів або виплавляти з неї метал. Окрім того, гравець та ігрові персонажі можуть нападати один на одного задля отримання руди, металу та кристалів щоб потім обміняти ці ресурси на крипто-валюту в точках обміну ресурсів. Також персонажі можуть захищати один одного від нападів та обмінюватися ресурсами між собою. Поведінка ігрових персонажів, які керують космічними кораблями на карті ігрового світу визначається значеннями його основних рис особистості. які можуть періодично змінюватися в ході ігрового процесу. Серед головних рис характеру, які впливають на поведінку ігрових персонажів є:

– агресивність;
– жадібність;
– законослухняність;
– переможність;
– комунікація;
– видобування;
– дослідження;
– обмін;

– ідеалізм.

Кожна з цих рис має однаковий діапазон значень [0; 100], що дозволяє спростити процес вимірювання афінностей між рисами ігрового персонажу та класом його поведінки. Слід зазначити, що під час додавання нового ігрового персонажу на карту відбувається встановлення цих ознак випадковим чином шляхом розподілення 300 балів між цими 9 базовими ознаками рис характеру ігрового персонажу. Поведінка ігрового персонажу визначається найближчим за характеристиками класом з переліку:

– грабіжник;
– захисник;
– робітник;
– розвідник;
– торговець;
– інженер.

Відповідно до цього у поведінці персонажів, які ближче до класу “грабіжник” основними будуть агресивні дії по відношенню до інших персонажів, особливо, якщо вони мають менш розвинений космічний корабель. У класі поведінки “грабіжник” домінують риси: агресивність, жадібність, переможність, всі інші риси можуть мати мінорне значення.

У поведінці персонажів класу “захисник” основними діями будуть проявлення агресії до “грабіжників” під час спроб нападу ними на інші космічні кораблі. У класі поведінки типу “захисник” домінують риси: агресивність, законослухняність, ідеалізм, при цьому всі інші риси можуть мати мінорне значення.

У поведінці персонажів з класу “робітник” будуть домінувати дії, направлені на видобуток руди, та її переробку на метал. Відповідно до цього клас поведінки типу “робітник” має домінуючі риси: законослухняність, видобування, комунікація, в той час, як інші риси мають мінорні значення.

Клас поведінки ігрових персонажів типу “розвідник” має основні дії, які направлені на пошук нових рудовищ на карті ігрового світу та видобутку

кристалів. Відповідно до цього в даному класі домінуючими рисами є: дослідження, законослухняність, жадібність, в той час як всі інші риси характеру ігрових персонажів будуть мати мінорні значення.

У поведінці персонажів класу “торговець” домінуючими діями є обмін ресурсів з іншими ігровими персонажами за меншою ціною, ніж у точках обміну ресурсами. Відповідно до цього основними рисами характеру ігрових персонажів цього класу є обмін, комунікація, жадібність. Інші риси характеру ігрових персонажів цього класу будуть мати мінімальні значення.

Клас поведінки ігрових персонажів типу “інженер” має основні дії, направлені на пошук нових рудовищ, пошук кристалів та виплавку металів, а також розвиток свого корабля. Головними рисами цього класу поведінки є дослідження, видобування та ідеалізм, в той час, як інші риси характеру будуть мати мінімальні значення.

Оскільки у грі користувач, так само як і кожен ігровий персонаж керує власним космічним кораблем – можливості розвитку корабля відіграє важливу роль в ігровому процесі. Відповідно до цього космічні кораблі, якими керує гравець та інші ігрові персонажі, мають низку специфічних модулів, які виконують різні задачі:

– двигун;
– батарея;
– гармата;
– радар;
– бур;
– плавильня;
– вантажний модуль для руди;
– вантажний модуль для металу;
– вантажний модуль для кристалів;
– ремонтний модуль.

Двигун дозволяє кораблю переміщатися у просторі ігрового світу додатку. Під час переміщення двигун використовує енергію з батареї

корабля. Підвищення рівня двигуна призводить до зменшення кількості енергії, яку використовує двигун під час переміщення. Батарея використовується кораблем для зберігання енергії, яку витрачає корабель під час виконання своїх основних функціональних можливостей. Підвищення рівня батареї призводить до збільшення кількості енергії, яку вона може зберігати. Гармата використовуються кораблем на нанесення пошкоджень іншим кораблям у випадку бойових зіткнень. Підвищення рівня гармати призводить до збільшення урону, який вона наносить супротивникам у боях. Радар дозволяє знаходити рудовища на карті ігрового світу під час переміщення корабля. Підвищення рівня радару дозволяє з більшою ймовірністю знаходити кристали у рудовищах під час видобутку. Бур дозволяє видобувати руду та кристали на рудовищах. Підвищення рівня буру дозволяє видобувати більше ресурсів за один цикл видобутку. Плавильня використовується для отримання металу з руди. під час якого витрачається певна кількість енергії. Підвищення рівня плавильні дозволяє зменшити кількість енергії, яка споживається цим модулем під час виготовлення металу з руди. Вантажні модуля для різних типів ресурсів дозволяють зберігати певну кількість ресурсів на кораблі для їх подальшого транспортування в місця продажу. Підвищення рівнів вантажних модулів збільшує кількість ресурсів, яку можна зберігати. Ремонтний модуль дозволяє відновлювати корабель від пошкоджень, отриманих під час зіткнень з іншими кораблями. На це відновлення витрачається певна кількість енергії, яка зменшується після підвищення рівня цього модуля.

Важливою особливістю ігрового процесу є періодична зміна стану рис характеру персонажу за одиницю ігрового часу – раз на хвилину. При цьому відбувається випадкове додавання або віднімання 1 пункту з випадкової риси характеру кожного персонажу. Після чого кожний ігровий персонаж визначає найближчі за значенням афінності класи поведінки та класи поведінки сусідніх персонажів.

3 АРХІТЕКТУРА ІГРОВОГО ПРОЕКТУ ТА МОДЕЛЬ ШТУЧНОЇ ІМУННОЇ МЕРЕЖІ ДЛЯ КЕРУВАННЯ ПОВЕДІНКОЮ ПЕРСОНАЖАМИ В ІГРАХ

3.1 Формат даних ігрового персонажу

Задля забезпечення високої якості роботи штучної імунної системи на початку її моделювання необхідно формалізувати вид даних та діапазони їх можливих значень. Формалізація типів, які використовуються для опису даних антитіл та антигенів – один з важливих етапів розробки та реалізації штучної імунної системи. Слід зазначити, що на швидкість визначення класів поведінки ігрових персонажів та на точність прийняття цього рішення значний вплив має перелік ознак, які використовуються під час визначення афінностей між імунними об’єктами різних видів, які використовуються системою під час формування специфічної імунної відповіді. Можна говорити про лінійну залежність між кількістю ознак імунного об’єкту та швидкістю формування імунної відповіді системою. Це обумовлюється тим, що збільшення кількості ознак імунних об’єктів призводить до збільшення кількості обчислювальних операцій під час обчислення афінності під час роботи операторів презентації та супресії мережі. Слід зазначити, що зараз найбільш поширеними видами опису ознак імунних об’єктів є:

– вектор ознак;
– матриця ознак;
– зважений масив векторів ознак.

У випадку використання вектору ознак для опису імунних об’єктів, всі його ознаки поєднуються у один масив. Головною умовою використання такої форми представлення характеристик імунних об’єктів системи є використання однакового діапазону можливих значень всіма поєднаними у загальний вектор ознаками. В іншому випадку робота імунних алгоритмів може призводити до формування великої кількості помилок.

У випадку використання матриці ознак для опису характеристик імунних об’єктів, всі ці параметри розподіляються поміж низки груп, які мають однаковий діапазон можливих значень. Слід зазначити, що матричний формат має низку особливостей: необхідність нормування груп ознак для формування матриці та відсутність ваг у рядків матриці. Ці особливості можуть призвести до появи великої кількості надлишкових даних, що негативно на швидкість обчислення афінності між об’єктами за рахунок збільшення кількості обчислювальних операцій.

Використання зваженого масиву векторів ознак передбачає формування багатовимірного масиву, групи ознак якого урівноважують заздалегідь визначені вагові коефіцієнти, які суттєво впливають на значення афінності під час презентації імунних об’єктів системою. Цей підхід передбачає неоднорідність ознак імунних об’єктів та використання різних діапазонів можливих значень для них, а також великого обсягу статистичних даних задля коректного визначення ваги для кожної окремою групи.

Відповідно до цього, у даній роботі буде використовуватися перший з визначених варіантів опису імунних об’єктів, а саме – вектор ознак. Це обумовлено тим, що всі ознаки характеру ігрового персонажу мають однаковий діапазон можливих значень, отже їх не потрібно нормувати та додатково підлаштовувати один до одного задля того, щоб використовувати для визначення класу поведінки ігрового персонажу у ігровому додатку.

3.2 Особливості методу baiNET для керування поведінкою персонажів

Незважаючи на те, що метод aiNET є одним з найбільш розповсюджених імунних методів, його не модна використовувати для вирішення задачі керування ігровими персонажами без додаткових модифікацій. Відповідно до цього, було зроблено модифікований метод behavioral-aiNET або baiNET, який може використовуватися для вирішення визначеної практичної задачі. Роботу модифікованого методу baiNET можна

умовно розділити на два основні етапи: етап формування специфічної імунної відповіді шляхом відтворення набору антигенів, які визначають класи поведінки ігрових персонажів, в процесі імунного навчання, та етап визначення об’єктів, з якими ці антитіла будуть взаємодіяти для реалізації особливості поведінки обраного класу. Таким чином метод baiNET на рівні імунних операторів визначається наступним чином:

$$baiNET = \left[\begin{matrix} Presentation(AB, AG) \rightarrow \\ Cloning(AB, CL) \rightarrow \\ Mutation(AB, CL) \rightarrow \\ Supression(AB, CL) \rightarrow \\ Termination(AB, AG) \end{matrix}\right]^{aiNET} \rightarrow \left[\begin{matrix} Selection(AB, AB', AB'') \rightarrow \\ Specification(AB', AB'') \end{matrix}\right]^{Act}, \tag{3.1}$$

де $Presentation(AB, AG)$ – оператор представлення популяції мережі антитіл до антигенів, з яких сформована навчальна вибірка, а також антитіл між собою; $Cloning(AB, CL)$ – оператор клонування популяції антитіл; $Mutation(AB, CL)$ – оператор мутації сформованих клонів; $Supression(AB, CL)$ – оператор супресії антитіл та клонів мережі; $Termination(AB, AG)$ – оператор перевірки зупинки процесу формування імунної відповіді мережі; $Selection(AB, AB', AB'')$ – оператор визначення антитіл, з якими буде взаємодіяти те чи інше антитіло, реалізуючи логіку свого класу поведінки; $Specification(AB', AB'')$ – оператор визначення об’єктів з якими можна взаємодіяти тому чи іншому антитілу в рамках свого класу поведінки, та антитіл, з якими слід уникати взаємодії.

Оскільки найбільший вплив на швидкість роботи та якість прийняття рішення методом baiNET мають його імунні оператори – слід окремо розглянути особливості їх роботи та налаштування.

Оператор $Presentation(AB, AG)$, який використовується в baiNET для представлення популяцій імунних об’єктів, також відповідає за визначення

цільових об’єктів для кожного антитіла. Слід зазначити, що зазвичай цей оператор викликається тільки один раз за весь процес формування специфічної імунної відповіді через велику кількість обчислень. Однак, враховуючи особливості ігрового дизайну цього проекту і той факт, що популяція антигенів зовсім невелика, та нараховує лише шість об’єктів – цей оператор використовується разом із операторами клонування та мутації антитіл під час формування кожної нової популяції об’єктів. Крім того, через невелику кількість ігрових персонажів на карті – передбачається, що їхня кількість вимірюється кількома сотнями, всі антитіла, які описують поведінку ігрових персонажів, можуть взаємодіяти один з одним без необхідності використання обмежень у вигляді NAT або якихось інших критеріїв. Однак, з точки зору корисності так взаємодія між всіх імунними об’єктами в популяції антитіл не є доцільною. Вплив на поведінку ігрового персонажу можуть оказувати лише персонажі, які знаходяться поблизу нього н карті ігрового світу. Тому під час роботи імунного оператору презентації для кожного окремого антитіла виділяється коло найближчих до нього антитіл за координатами на карті, а не за рисами характеру. Відповідно до цього для визначення найближчих персонажів буде використовуватися манхетеннсьа відстань по координатах у двовимірному просторі, замість евклідової відстані, розрахунок якої потребує більше обчислювальних операцій. Потім серед всіх для кожного антитіла буде обиратися 5 найближчих на карті об’єктів, із якими він буде взаємодіяти впродовж формування імунної відповіді.

Оператор клонування $Cloning(AB, CL)$ використовується для формування певної кількості клонів для кожного неспецифічного антитіла. Слід зазначити, що в модифікованому методі baiNET використовується статичне клонування, яке передбачає використання деякого зовнішнього параметру, за допомогою якого буде встановлено кількість клонів, які формуються для кожного окремого антитіла, яке не набуло стану специфічності до одного з зовнішніх антигенів. Відповідно до цього,

кількість клонів, яка буде використовуватися в baiNET під час клонування буде дорівнювати 20 для кожного імунного об’єкта.

Оператор мутації $Mutation(AB, CL)$ зазвичай використовується для внесення змін в параметри сформованих раніше клонів. В модифікованому методі baiNET цей оператор змінює не тільки клони, але й саме антитіло. Слід зазначити що спосіб мутації клонів та антитіла, на основі якого ці клони були сформовані під час клонування, значно відрізняються між собою. Для клонування антитіла використовується випадкове статичне клонування із коефіцієнтом мутації μ, який буде дорівнювати 1. Для зміни параметрів клонів буде використовуватися зворотно-пропорційна мутація на основі афінності між антитілом та антигеном з обмеженням мінімального значення коефіцієнта мутації відповідно до наступного виразу:

$$\mu = rand\left[\, 0{,}25 \times \left(1 - aff_{ij}\right);\ 1 - aff_{ij}\right], \tag{3.2}$$

де μ – коефіцієнт мутації, а aff_{ij} – значення афінності між i антитілом, від якого було сформовано клон для подальшої мутації, та j антигеном.

Слід зазначити, що використання такого способу обчислення коефіцієнту мутації може суттєво пришвидшити процес досягання стану специфічності між антитілом та одним з антигенів за невелику кількість популяцій імунних об’єктів.

Оператор супресії $Supression(AB, CL)$ використовується з метою приведення кількості антитіл та клонів імунної мережі в початковий стан для того, щоб не відбувалося неконтрольованого росту системи та використання зайвих обчислювальних ресурсів. Відповідно до цього серед всіх клонів, чиї ознаки було змінено під час мутації, обирається клон, що має найбільше значення афінності до одного з антигенів, які представляють класи поведінки ігрових персонажів. Це передбачає що на першому етапі супресії кожен клон визначає афінності до кожного з антигенів – так відбувається через те, що кількість класів поведінки ігрових персонажів є дуже невеликим значенням,

так само, як і кількість клонів. Окрім того, треба відзначити послідовний процес обробки антитіл, тобто виконання всіх імунних операторів, наведених у виразі (3.1), відбувається послідовно над кожним окремим антитілом, а не над всією популяцією разом, окрім оператору зупинки процесу формування специфічної імунної відповіді. Відповідно до цього для кожного окремого антитіла відбувається презентація антигенів та інших антитіл, клонування, мутація та супресія, після чого це повторюється для наступного антитіла імунної мережі до тих під поки не будуть оброблені всі антитіла. Тільки після цього відбудеться перевірка можливості зупинки формування імунної відповіді мережею. Отже під час супресії кожен клон визначає афінності до всіх антигенів, після чого залишає лише афінность з максимальним значенням, на основі якої буде обиратися клон, який має найвищу афінность до того чи іншого антигену. Після чого дані цього клона та його афінность до визначеного антигену будуть збережені у антитіло, на основі якого цей клон був сформований. Відповідно до цього в кожен окремий час роботи імунної системи для кожного антитіла можна визначити найімовірніший по максимальному значенню афінності клас поведінки. Наприкінці виконання супресії всі клони видаляються з мережі.

Модифікований метод baiNET використовує критеріальний оператор зупинки формування специфічної імунної відповіді $Termination(AB, AG)$. Тобто кількість повторень операторів клонування, мутації та супресії для досягнення стану умовної специфічності та припинення процесу формування імунної відповіді, визначається за допомогою вказаного числа популяцій антитіл, яке використовується у якості вхідного параметру алгоритму. Відповідно до цього кількість популяцій антитіл, яка буде створюватися під час формування специфічної імунної відповіді мережею не перевищує одну популяцію імунних об’єктів.

Оператор вибору групи антитіл для подальшої взаємодії – $Selection(AB, AB', AB'')$ використовується для визначення антитіл, із якими будуть відбуватися взаємодії в процесі реалізації ігровим персонажем

специфічного виду поведінки, обраного в процесі формування імунної відповіді мережею. Слід зазначити, що цей оператор послідовний процес обробки антитіл, а також використання інших даних про антитіла, ніж дані в процесі формування імунної відповіді. Під час виконання цього оператору відбувається визначення афінності на основі координат антитіл на карті ігрового світу. Слід зазначити, що цей оператор також використовує зовнішній параметр, який обмежує кількість взаємодій з іншими антитілами до значення в 10 об'єктів. Тобто після закінчення роботи цього оператору для кожного антитіла буде обрано по 10 інших антитіл, із якими воно буде співпрацювати в подальшому задля реалізації особливостей своєї поведінки.

Оператор специфікації об'єктів $Specification(AB', AB'')$ є останнім імунним оператором, який передбачається в методі baiNET. Робота цього оператору полягає в тому, щоб визначити для кожного з антитіл, які представляють поведінку ігрових персонажів, з якими антитілами воно буде намагатися взаємодіяти під час реалізації поведінки з урахуванням особливостей обраного класу, а яких антитіл слід уникати. Слід зазначити, що робота цього оператору виконується послідовно та окремо для кожного з антитіл мережі. Відповідно до цього для кожного антитіла, серед обраних 10 найближчих до нього на карті ігрового світу антитіл визначаються об'єкти, з якими воно буде намагатися співпрацювати, та об'єкти, яких воно має уникати. Визначення цих об'єктів відбувається завдяки використанню специфічних коефіцієнтів взаємодії, які додаються до головних рис характеру персонажу та впливають на його поведінку. Для більшості ознак ці коефіцієнти мають значення 1, однак для таких ознак, як агресивність та переможність коефіцієнти будуть мати значення -4. Для таких ознак, як законослухняність, видобування, комунікація та обмін ці коефіцієнти будуть мати значення 1,5. Відповідно до цього визначення афінностей між антитілами на цьому етапі роботи штучної імунної системи буде відбуватися за допомогою зміненої формули обчислення афінності:

$$aff_{ij} = \left(1 + \sqrt{k_1(p_{1i} - p_{1j})^2 + k_2(p_{2i} - p_{2j})^2 + \ldots + k_n(p_{ni} - p_{nj})^2}\right)^{-1}, \quad (3.3)$$

де aff_{ij} – значення афінності між i та j антитілами; k – коефіцієнт взаємодії на основі рис характеру; p – ознаки імунних об'єктів.

Таким чином, взаємодія між об'єктами буде у випадку, якщо афінность між ними, обчислена відповідно до (3.3) буде перевищувати 0,5. Інакше антитіло, що представляє ігрового персонажа, буде намагатися уникати взаємодії або переміщатися по карті ігрового світу подалі від цього об'єкту. Слід зазначити, що у випадку, якщо дії цього оператору підлягає антитіло з класом поведінки "грабіжник" – він буде переслідувати та атакувати кораблі, які мають найменш розвинутий модуль "гармата". Через те, що всі кораблі мають однакову швидкість на карті ігрового світу, персонажі з типом поведінки "грабіжник" не можуть наздогнати інші об'єкти під час їх пересування, та атакують лише нерухомі кораблі. Нерухомим корабель стає в одному з трьох випадків: видобування руди, обмін з іншим кораблем, підвищення рівня одного з модулів. Слід зазначити, що під час перебування корабля у пункті обміну ресурсів він не може бути атакованим.

3.3 Реалізація архітектури MVI для ігрового проекту

Програмна реалізація ігрового програмного засобу, в якому керування ігровими персонажами здійснюється на онові штучної імунної мережі та методу baiNET проводилася на Unity 2017 при використанні платформи .NET та мови програмування C#. Крім того, програма реалізована на основі поширеного архітектурного шаблону програмування MVI. Розроблений ігровий додаток є достатньо складним програмним засобом та умовно поділяється на декілька основних модулів, що мають різне функціональне призначення:

– модуль інтерфейсу користувача, реалізований на рівні View;

– модуль представлення даних, реалізований на рівні Model;
– модуль імунного алгоритму та операторів, реалізований в Model;
– модуль поведінки ігрових персонажів, реалізований на рівні Model;
– модуль карти ігрового світу, реалізований на рівні Model;
– модуль файлових операцій, реалізований на рівні Model;
– модуль інфраструктурної взаємодії, реалізований на рівні Intent.

У модулі інтерфейсу користувача представлені типи даних, які дозволяють організовувати взаємодію із гравцем та займається візуалізацією ігрового процесу комп’ютерної гри. Саме цей модуль забезпечує взаємодію між користувачем та функціональними можливостями ігрового додатку, а також процес візуалізації його діалогових вікон.

У модулі даних знаходяться класи, що використовуються для роботи з даними користувача, даними ігрових персонажів, імунними об'єктами, представленими набором антитіл, навчальних антигенів, клонів та класів поведінки ігрового персонажу. Слід зазначити, що при цьому відповідно до основних положень теорії штучних імунних систем класи антигенів, антитіл і клонів влаштовані однаково і успадковуються від базового класу клітини. Окрім того, організація цих типів формувалася з урахуванням принципів ООП та концепцій SOLID.

Модуль імунного алгоритму та його операторів містить реалізацію методу behavioral-aiNet, класи з методами обчислення евклідової відстані, обчислення афінності, коефіцієнтів мутації, а також реалізацію всіх імунних операторів, які використовуються в цьому методі. При цьому кожен імунний оператор реалізований в окремому класі, який організовано на основі шаблону програмування singleton, що спрощує можливості його використання в імунному алгоритмі, а також можливості модифікації в подальшому у випадку масштабування запропонованого імунного методу.

Модуль поведінки ігрових персонажів реалізує логіку роботи ігрових персонажів, які керують космічними кораблями під час ігрової сесії. Слід зазначити, що саме ці ігрові персонажі знаходяться під керуванням

модифікованої імунної мережі baiNET. Здебільшого поведінка персонажів виражена набором intents, які реалізують наміри того чи іншого ігрового персонажу реалізувати поведінку, обумовлену особливостями того чи іншого класу поведінки ботів ігрового додатку.

Модуль карти ігрового світу відповідає за організацію та імплементацію глобальних та загальних подій на карті, такі як появу нового рудовища, або вичерпання старого рудовища, додавання нових ботів замість ігрових персонажів, чиї космічні кораблі були знищені грабіжниками або захисниками, в результаті зіткнень з іншими ігровими персонажами.

Модуль файлових операцій містить класи, що дозволяють працювати зі стандартними типами фалів, представленими у форматах CSV та XML. У даних форматах зберігається вся інформація про зареєстрованих користувачів, а також про ігрові сесії кожного з зареєстрованих accounts. Слід зазначити, що інформація про зареєстрованих користувачів ігрового додатку зберігається у файлі з форматом XML шляхом серіалізації даних. Відповідно до цього на початку роботи ігрового додатку відбувається десеріалізація цього файлу, шляхом якої відбувається завантаження даних про користувачів та їхні рейтинги. Також важливо особливістю ігрового додатку є збереження інформації щодо дій користувача та ботів на карті ігрового світу. Відповідно до цього ігровий додаток реалізує повне логування дій впродовж всієї ігрової сесії користувача.

Модуль інфраструктурної взаємодії представлений переліком типів intent, які викликаються задля забезпечення взаємодії між рівнем інтерфейсу користувача та моделлю. Відповідно до правил архітектурного шаблону MVI об’єкти типу intent дозволяють користувачеві викликати ті чи інші операції та робити ті чи інші дії на карті ігрового світу під час гри. Кожна функціональна можливість, яка передбачена у грі має свій intent, яки дозволяє змінювати стан тієї чи іншої частини даних, представленої на рівні Model. Слід зазначити, що сам по собі intent може і не мати повної реалізації всієї необхідної функціональності, яку намагається запустити користувач.

Цю логіку intent викликає з типів, реалізованих на рівні Model у коректній послідовності, що спричиняє зміну станів моделі.

На рисунку 3.1 наводиться файлова структура програми, розділена на три основні рівня, у відповідності до принципів архітектурного шаблону MVI. Таким чином папка Intents містить перелік типів даних, які реалізують той чи інший функціональний намір користувача.

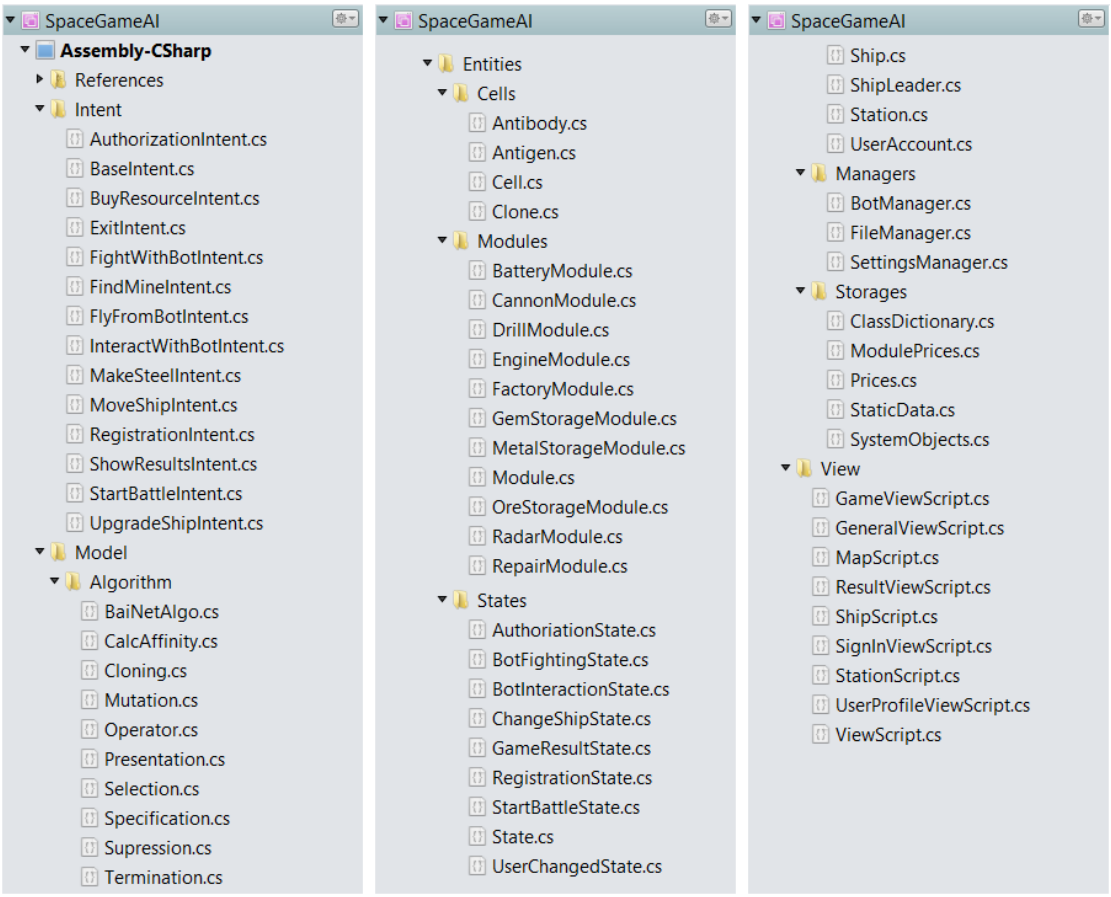

Рисунок 3.1 – Файлова структура застосунку

Папка View містить перелік скриптів, які додаються до UI задля реалізації логіки роботи інтерфейсу користувача. Слід зазначити, що всі ці скрипти успадковуються від одного файлу ViewScript, що реалізовує логіку віконного класу, надає можливості перемикання вікон ігрового додатку. Папка Model має децентралізовану та розподілену структуру, відповідно до основних принципів SOLID, що забезпечує можливості її швидкого масштабування, редагування та вирішення проблем у разі виникнення

помилок в роботі додатку. Слід зазначити, що у папці Model окремою частиною реалізована папка States, яка зберігає перелік універсальних станів моделі додатку разом із локалізованою інформацією щодо даних, які описують користувача та окремих ботів.

3.4 Результати випробувань методу baiNET в ігровому проекті

Для аналізу запропонованого методу керування ігровими персонажами у комп'ютерній грі було проведене декілька ігрових сесій із різною кількістю ігрових персонажів на різних за розмірами картах ігрового світу. Важливим фактором при цьому були налаштування карт, які регламентували кількість рудовищ, які одночасно існують на карті ігрового світу, їх видобув альні можливості та імовірності видобутку на цих рудовищах кристалів замість звичайної руди, з якої потім можна отримати метал для подальшого продажу. Характеристики наборів даних наведено у таблиці 3.1.

Таблиця 3.1 – Набори даних для налаштування карт ігрового світу

Ідентифікатор	Кількість			Об'єм рудовищ	Імовірність видобутку кристалів
	Персонажів	Точок обміну	Рудовищ		
Карта 1	50	2	10	100	10%
Карта 2	100	3	5	50	1%
Карта 3	100	3	25	200	20%
Карта 4	200	4	10	50	1%
Карта 5	200	4	50	300	10%

Слід зазначити, що генерація космічних кораблів, якими керують ігрові персонажі відбувається на різних точках обміну ресурсів. На цих точках кораблі не можуть атакувати один одного навіть, якщо вони класифікуються як "грабіжник" та "захисник". Окрім того, на цих точках кораблі не можуть здійснювати обмін між собою – обмін може відбуватися тільки із системою,

представленою кількома точками обміну.

Під час порівняння роботи алгоритму baiNet в різних умовах використання відбувалося дослідження поведінки імунних об’єктів, які формували популяцію антитіл та їх відношення до визначеної кількості класів, які було представлено набором антигенів. В процесі дослідження за ботами увага фокусувалася на закономірностях та особливостях поведінки ігрових персонажів в різних ролях, а також на динаміці зміни цих ролей в процесі роботи ігрового додатку. Важливим аспектом в роботі ігрового додатку є реалізація своєї ролі кожним з ігрових персонажів. Під реалізацією ролі ігрового персонажу будемо розуміти виконання ігровим персонажем дій, які було передбачено відповідним класом поведінки. Результати випробувань алгоритму baiNET для керування ігровими персонажами наведені нижче.

Таблиця 3.2 – Реалізація ролей ігровими персонажами на основі baiNET

Ідентифікатор	Карта 1	Карта 2	Карта 3	Карта 4	Карта 5
Грабіжник	78%	93%	73%	95%	65%
Захисник	50%	56%	65%	60%	77%
Робітник	90%	80%	95%	80%	95%
Розвідник	90%	90%	95%	85%	95%
Торговець	80%	75%	90%	70%	70%
Інженер	35%	20%	50%	18%	48%

Відповідно до наведених результатів можна скласти вивід про те, що деякі класи дуже рідко реалізуються в повному обсязі до того, як корабель буде знищено грабіжником, або клас поведінки не буде змінено в процесі роботи ігрового додатку. Одночасно з цим можна виділити класи, переважно реалізуються у випадках великої кількості ігрових ресурсів на карті ігрового світу, а також класи, які переважно реалізуються у випадку маленької кількості ресурсів на карті ігрового світу.

Як видно з результатів випробувань, наведених в табл. 3.2., найменше

реалізується клас “інженер”, особливо в умовах браку ресурсів на карті ігрового світу через те, що у більшості випадків обрання цього класу ігровим персонажем супроводжується недостатньою кількістю ресурсів для постійного підвищення рівня модулів космічного корабля. Також можна зазначити, що незалежно від особливостей карти ігрового світу, з високим рівнем імовірності реалізуються класи, сфокусовані на розвідці рудовищ та видобуванні ресурсів. Також серед класів поведінки ігрових персонажів можна виділити класи “торговець” та “захисник”, імовірність реалізації яких зростає у випадку використання карт з великою кількістю ресурсів та великою кількістю ігрових персонажів. Імовірність реалізації класу поведінки “грабіжник” безпосередньо залежить від відсутності ресурсів – чим менша кількість ресурсів на карті, та чим більше на карті при цьому ігрових персонажів – тим вища імовірність реалізації поведінки цього класу.

Слід зазначити, що запропонований метод behavioral-aiNet характеризується для керування ігрових персонажів вперше, так само як і модель штучної імунної мережі. Тому на сьогоднішній день достатньо важко робити порівняння із іншими імунними та неімунними методами, які використовуються для організації системи керування ігровими персонажами. Також важливо відмітити, що запропонований імунний метод простий у реалізації та модифікації, що робить можливим його подальше використання та модифікацію для керуваннями ігрових персонажів у іграх інших жанрів.

ВИСНОВКИ

У кваліфікаційній розглянуто вирішення актуального завдання керування ігровими персонажами на основі теорії штучних імунних систем та моделі штучної імунної мережі aiNET. При виконанні поставленого завдання було проведено аналіз найбільш поширених імунних моделей та методів, які можуть використовуватися для вирішення різних задач класифікації, кластеризації та розпізнавання образів.

Проведений аналіз імунних моделей дозволив визначити їх основні особливості, переваги та недоліки. Крім того, для кожної виділеної імунної моделі було визначено універсальні алгоритми, що використовуються для вирішення різних практичних завдань. Також було визначено універсальні імунні оператори, які можна використовувати під час розробки імунного методу керування поведінкою ігрових персонажів у комп’ютерних ігрових додатках. В ході дослідження особливостей імунних операторів було вивчено основні підходи, що використовуються для організації їх функціонування.

В результаті проведення низки експериментів з моделювання імунних методів, реалізованих на основі різних імунних моделей, був виділений метод behavioral-aiNet, який функціонує на основі штучної імунної мережі та її поширеної моделі aiNET, як найбільш підходящий для вирішення задачі керування ігровими персонажами. Використання даного методу для вирішення поставленої задачі обумовлюється необхідністю мережевої взаємодії під час визначення типу поведінки антитіл, які використовуються для представлення характерів ігрових персонажів.

При виконанні даної кваліфікаційної роботи було розроблено та реалізовано імунний метод керування ігровими персонажами behavioral-aiNet, що характеризується високою швидкістю обчислення та зручністю реалізації та модифікації для подальшого застосування в ігрових проектах інших жанрів та динаміки ігрового процесу.

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

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

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

5. 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.

6. 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.

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

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

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

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. 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.

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

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

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

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

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

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

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. 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.

28. 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.

29. 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.

30. 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.

31. 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.

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.