МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____202_ р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

ІНТЕЛЕКТУАЛЬНА СИСТЕМА МОДЕЛЮВАННЯ ПОВЕДІНКИ ПЕРСОНАЖІВ В КОМП'ЮТЕРНИХ ІГРАХ

Спеціальність 122 «Комп'ютерні науки»

122 – МКР – 601.21610418

Студент _____ А. Д. Под'ячев
«__» _____ 2022 р.

Консультант _____ О. П. Гожий
доктор технічних наук, професор
«__» _____ 2022 р.

Миколаїв – 2022

Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Освітньо-кваліфікаційний рівень **магістр**
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____20__р.

ЗАВДАННЯ
на магістерську кваліфікаційну роботу

ПОД'ЯЧЕВУ Андрію Денисовичу

1. Тема магістерської кваліфікаційної роботи «Інтелектуальна система моделювання поведінки персонажів в комп'ютерних іграх».

Керівник роботи Гожий Олександр Петрович д-р техн. наук, професор.

Затв. наказом Ректора ЧНУ ім. Петра Могили від «___» _____20__р. № 287

2. Строк подання студентом роботи «___» ___ 20__р.

3. Вхідні (початкові) дані до роботи: для виконання роботи не потрібні початкові дані.

Очікуваний результат роботи: Інтелектуальна система моделювання поведінки персонажів в комп'ютерних іграх.

4. Зміст пояснювальної записки (перелік питань, які потрібно розглянути):

- аналіз сучасного стану ігрового штучного інтелекту;
- огляд існуючих методів штучного інтелекту;
- розробка інтелектуальної системи моделювання поведінки персонажів в комп'ютерних іграх

5. Перелік графічного матеріалу: презентація.

6. Завдання до спеціальної частини: Основні вимоги техніки безпеки і промислової санітарії.

_____

_____

_____.

7. Консультанти:

Розділ	**Прізвище, ініціали та посада консультанта**	**Підпис**
Спеціальна частина з охорони праці		
Методична частина		

Керівник роботи д.т.н, професор, Гожий О. П.

(наук. ступінь, вчене звання, прізвище та ініціали)

_____

(підпис)

Завдання прийнято до виконання Под'ячев А.Д.

(прізвище та ініціали)

_____

(підпис)

Дата видачі завдання «____» _____ 20__ р.

КАЛЕНДАРНИЙ ПЛАН
Виконання магістерської кваліфікаційної роботи

Тема: Інтелектуальна система моделювання поведінки персонажів в комп'ютерних іграх.

№	Найменування роботи	Початок	Закінчення	Примітки
1	Визначення керівника і теми МКР. Подання заяви на затвердження теми МКР	01.09.2021	10.10.2021	
2	Отримання завдання на виконання МКР	19.10.2021	22.10.2021	
3	Складання календарного плану на період виконання МКР	23.10.2021	26.10.2021	
4	Огляд літератури за темою дослідження	27.10.2021	10.11.2021	
5	Проходження переддипломної практики, збір та аналіз матеріалів до МКР	22.11.2021	11.12.2021	
6	Аналіз предметної області та розробка технічного завдання. Моделювання результатів	16.12.2021	12.01.2022	
7	Опис фахової частини МКР	13.01.2022	25.01.2022	
8	Розробка спеціальної частини з охорони праці та методичної частини	26.01.2022	30.01.2022	
9	Попередній захист МКР на засіданні комісії кафедри	31.01.2022	31.01.2022	
10	Корегування роботи за результатами попереднього захисту	01.02.2022	03.02.2022	
11	Остаточне оформлення пояснювальної записки та слайдів доповіді для захисту	04.02.2022	06.02.2022	
12	Подання МКР рецензенту	09.02.2022	10.02.2022	
13	Рецензування МКР	11.02.2022	12.02.2022	
14	Подання МКР, її електронної копії та інших документів (відгуку, рецензії) до захисту	14.02.2022	15.02.2022	
15	Захист МКР перед екзаменаційною комісією (ЕК)	21.02.2022	22.02.2022	

Розробив студент Под'ячев А. Д. _____
(*підпис*)

Керівник роботи д.т.н, професор, Гожий О. П. _____
(підпис)

«____» _____ 202__р.

АНОТАЦІЯ

до магістерської кваліфікаційної роботи
студента групи 601 ЧНУ ім. Петра Могили

Под'ячева Андрія Денисовича

на тему: **«ІНТЕЛЕКТУАЛЬНА СИСТЕМА МОДЕЛЮВАННЯ ПОВЕДІНКИ ПЕРСОНАЖІВ В КОМП'ЮТЕРНИХ ІГРАХ»**

Актуальність даного дослідження полягає у необхідності створення більш непередбачуваного та людиноподібного ігрового штучного інтелекту. Це дозволить збільшити рівень поглиблення в гру, та підвищити якість ігрового процесу.

Об'єктом поведінка персонажів в комп'ютерних іграх.

Предметом дослідження є методи ігрового штучного інтелекту, та моделювання поведінки персонажів.

Метою дослідження є покращення ігрового досвіду і підвищення якості фінального продукту, за рахунок підвищення рівня поглиблення в гру, та створення більш непередбачуваного та людиноподібного ігрового штучного інтелекту.

В результаті виконання роботи було досліджено сучасні підходи до створення ігрового штучного інтелекту. Були досліджені та описані основні принципи та технології, що використовуються у ігровому ШІ, оцінена перспективність цієї галузі та проведено аналіз основних напрямків розвитку.

Дана робота складається з шести розділів. Кожен розділ відповідно присвячений: аналізу предметної області, математичним моделям і методам, використаним у магістерській роботі, моделюванню і проектуванню системи планування і оптимізації маршрутів, аналізу отриманих результатів, охороні праці, методичній частині магістерської роботи. Загальний обсяг роботи – 110 сторінки. Магістерська кваліфікаційна робота містить 2 додатки, 22 рисунків, 3 таблицю і посилання на 106 літературних джерел.

Ключові слова: штучний інтелект, ігри, дерева поведінки, нечітка логіка.

ABSTRACT

to the master's qualification work by the student of the group 601 of Petro Mohyla Black Sea National University

Podiachev Andrii

“INTELLECTUAL SYSTEM FOR MODELING THE BEHAVIOR OF CHARACTERS IN COMPUTER GAMES”

The relevance of this study lies in the need to create a more unpredictable and human-like gaming artificial intelligence. This will increase the level of immersion in the game, and improve the quality of the gameplay.

An object of research is the behavior of characters in computer games.

A subject of the research is the methods of game artificial intelligence and modeling the behavior of characters.

A purpose of the study is to improve the gaming experience and improve the quality of the final product, by increasing the level of immersion in the game, and creating a more unpredictable and human-like gaming artificial intelligence.

As a result of the work, modern approaches to the creation of artificial game intelligence were studied. The basic principles and technologies used in gaming AI were researched and described, the prospects of this field were assessed and the analysis of the main directions of development was carried out.

This work consists of six sections. Each section is devoted to: analysis of the subject area, mathematical models and methods used in the master's thesis, modeling and design of planning and optimization of routes, analysis of results, labor protection, methodological part of the master's thesis. Total volume of work - n pages.

The overall scope of the work is 110 pages. Thesis contains 2 applications, 22 figures, 3 tables and 106 sources in it.

Key words: artificial intelligence, games, behavior trees, fuzzy logic.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

ДП – дерево поведінки;

ШІ – штучний інтелект;

NPC – неігровий персонаж;

FSM – скінченний автомат;

Пояснювальна записка

до магістерської кваліфікаційної роботи

на тему:

«ІНТЕЛЕКТУАЛЬНА СИСТЕМА МОДЕЛЮВАННЯ ПОВЕДІНКИ ПЕРСОНАЖІВ В КОМП'ЮТЕРНИХ ІГРАХ»

Спеціальність 122 «Комп'ютерні науки»

122 – МКР – 601.21610418

Студент _____ А. Д. Под'ячев
«__» _____ 2022 р.

Консультант _____ О. П. Гожий
доктор технічних наук, професор
«__» _____ 2022 р.

Миколаїв – 2022

ВСТУП

У 1950 році Клод Шеннон опублікував свою основоположну роботу про комп'ютерні шахи [20]. Це був початок комп’ютерної ери; ENIAC було всього кілька років, і далекоглядні Шеннон і Алан Тьюрінг побачили величезний потенціал цієї технології. Більшість комп’ютерів в ту епоху використовувалися для військових цілей, як правило, для балістичних розрахунків для ракет. На відміну, ігри здавалися природним додатком для комп’ютерів, яким можуть користуватися звичайні люди. Бажання використовувати комп’ютери для додатків, які привернуть увагу громадськості, мотивувало Артура Семюела розпочати свій 25-річний прцес будівництва сильну програму для гри в шашки [21, 22]. З'явилася перша робоча програма для шашок у 1952 р. [23], а невдовзі після цього з'явилися шахові програми [24].

Перші зусилля Шеннона, Семюела, Тьюринга, Аллана Ньюелла, Герберта Саймона та інших викликали значний інтерес до дослідження продуктивності комп’ютера в іграх. Розробка ігрових програм стала головною сферою досліджень у новітній галузі штучного інтелекту (ШІ). У той час мало хто усвідомлював складність створення програм, які демонстрували «інтелект» на рівні людини, і перші дні розвитку ШІ переслідували надмірно оптимістичні прогнози [25].

Ігровий штучний інтелект (ігровий ШІ) є галуззю розробка відеоігор, що стосується розширення можливостей ігри з створенням ілюзії інтелекту. Ігровий ШІ позичає багато методів із ширшої області ШІ, від простих кінцевих автоматів до найсучасніших еволюційних алгоритмів. Серед цих методів нечітка логіка є одним із інструментів повинен бути присутнім в арсеналі хорошого розробника ігрового ШІ через простоту її формулювання в поєднанні з її можливостями.

Сфера ігрового штучного інтелекту охоплює всі технології та методи введення інтелекту в відео ігри. До ігрового ШІ відноситься багато різних аспектів відеоігор: керування анімацією, керування, пошук шляху, планування, процедурна генерація, тактичне та стратегічне мислення та навчання [1,2]. Всі ці аспекти мають спільну основу, вони ставлять проблеми, ефективне вирішення

яких вимагає алгоритмів ШІ. Мета ігрового ШІ – оживити неігрових персонажів (NPC), присутніх в іграх, наприклад, ворожі війська в стратегії або купців у мирному селищі, контрольованому штучним інтелектом.

Ігровий ШІ – це лише одна з галузей більш широкого поля штучного інтелекту. Академічні дослідження штучного інтелекту завжди відчували труднощі з визначенням самого штучного інтелекту, хоча зазвичай його можна визначити, як когнітивні процеси, схожі до когнітивних процесів людини або на рівні когнітивних процесів людини, і в здатність цих процесів до навчання [3]. За словами Алана Тюрінга, якого вважають батьком штучного інтелекту, агент розумний, якщо його поведінку неможливо розрізнити від поведінки людини [4]. Ігровий ШІ додає ще один елемент до задачі визначення ШІ, оскільки вона не стосується реального існування інтелекту, але лише з ілюзією його [5]. Щоб гра була успішною, нам насправді не потрібні високоінтелектуальні, схожі на людину, можливо, непереможні суперники, потрібен переконливий супротивник, який є агентом ШІ з яким приємно грати, та який симулює поведінку реальної людини чи істоти.

Дана магістерська робота прошла апробацію на всеукраїнській науково-практичній конференції молодих вчених, аспірнатів і студентів. Вона складається з шести розділів. Кожен розділ відповідно присвячений: аналізу предметної області, математичним моделям і методам, використаним у магістерській роботі, моделюванню і проектуванню рекомендаційної системи музичного контенту, аналізу отриманих результатів, охороні праці, методичній частині магістерської роботи. . Загальний обсяг роботи – 110 сторінки. Магістерська кваліфікаційна робота містить 2 додатки, 22 рисунків, 3 таблицю і посилання на 106 літературних джерел.

1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ. ПОСТАНОВКА ЗАДАЧІ

1.1 Опис предметної сфери

Інтелект — одне з тих цікавих понять, щодо яких кожен має свою думку, але мало хто може дати визначення — і коли вони це роблять, їхні визначення, як правило, розходяться одне з одним. І, що цікаво, думки консенсусу змінюються з часом: розглянемо, наприклад, ряд показників людського інтелекту, як-от арифметичні навички, обсяг пам'яті, гра в шахи, доведення теорем — усі вони зазвичай використовувалися в минулому, але оскільки зараз машини перевершують людей у ці завдання вони вийшли з ладу. Ми направляємо зацікавленого читача до всебічного розгляду цього питання.

Сучасна література зі штучного інтелекту містить безліч тестів, багато з яких, на жаль, дуже вузькі, застосовні лише для невеликого класу завдань. Це не означає, що вони не можуть бути корисними для просування галузі, але ретроспективно часто стає зрозумілим, наскільки незначний внесок у виконання вузького завдання в загальній області. Наприклад, дослідники стверджували, що серйозний прогрес у такій складній грі, як шахи, обов'язково принесе багато уявлень, а прийоми, використані в розв'язанні, будуть корисні для вирішення реальних проблем – ну, ні.

Ігри та штучний інтелект мають тісні й давні зв'язки один з одним. Сам Тьюринг, який запропонував перший тест на машинний інтелект, також винайшов алгоритм MiniMax для ідеальних інформаційних ігор для двох гравців і вважав шахи важливою областю для майбутніх досліджень інформатики [26]. Імовірно, перший у світі алгоритм навчання з підкріпленням (попередник сучасного навчання тимчасових різниць) був винайдений Семюелом у контексті створення автоматичної програми для гри в шашки, таким чином започаткувавши основний напрямок сучасних досліджень ШІ.

Зовсім недавно кілька відомих дослідників ШІ запропонували ігри як хороший еталон для ШІ. Принаймні частина аргументів полягає в тому, що технічний розвиток сучасних комп'ютерних ігор тепер безсумнівно обігнав

спеціально створені тести та симулятори роботів, а комерційна індустрія ігор надає величезну кількість високоякісних, добре налаштованих проблем і середовищ для досліджень ШІ. як побічний ефект комерційного тиску на кращі ігри.

Існує не одне визначення того, що таке гра, але багато. Насправді, Вітгенштейн використовував поняття гри в ряді думок-експериментів, покликаних показати, що неможливо правильно визначити будь-яке поняття з точки зору достатніх і необхідних умов; натомість поняття неявно визначаються тими речами, на які вони посилаються, і які пов'язані між собою через сімейну схожість. Навчитися використовувати поняття — це навчитися грати в мовну гру, частиною якої є поняття, ще один приклад гри [4].

Неможливість визначення природного поняття не означає, що ви не повинні намагатися; робоче визначення ігор було б чудовим, навіть якщо ми визнаємо, що воно не є всеохопним. Ігровий дизайнер Сід Меєр визначає гру як «серію значущих рішень». Інші, такі як Сален і Циммерман, підкреслюють конфлікти як центральне місце в іграх: гра — це «система, в якій гравці беруть участь у штучному конфлікті, визначеному правилами, що призводить до результату, який можна виміряти». Juul дає більш офіційне визначення: Гра — це заснована на правилах формальна система зі змінним і кількісно виміряним результатом, де різним результатам призначаються різні значення, гравець докладає зусиль, щоб вплинути на результат, гравець відчуває прихильність до результату, а наслідки діяльності є факультативними та обговорюваними.

Наведені вище визначення не позбавлені критики. Наприклад, ігровий дизайнер Раф Костер зауважує, що жодна з них не містить слова «весело». Оскільки веселощі, здається, є центральними в іграх, він присвячує цілу книгу тому, щоб зрозуміти, що робить ігри веселими.

Для наших цілей більш релевантною критикою наведених вище визначень є те, що вони стосуються «гравця», який може «докласти зусиль» і «брати участь» у «значущих виборах». Очевидно, що включення такого гравця у визначення гри,

яка використовується для тестування штучного інтелекту, поставить питання про штучний інтелект в цілому. Тому було вирішено адаптувати визначення Джула відповідно до наших цілей: Гра — це заснована на правилах формальна система зі змінним і кількісно виміряним результатом, де різним результатам призначаються різні значення, і агент може впливати на результат, виконуючи дії під час гри, засновані на повному або частковому знаннях стану гри.

1.2 Огляд та аналіз наявних аналогів та публікацій

1.2.1 Адаптивний ігровий ШІ з динамічним сценарієм

Спронк та ін. розглянули онлайн-навчання в комерційних комп'ютерних іграх. Цей підхі дозволяє супротивникам, керованим комп'ютером, адаптуватися до способу гри. Таким чином, цей підхід забезпечує механізм для боротьби зі слабкими місцями в ігровому ШІ та для реагування на зміни в тактиці гравця. Стверджуэться, що онлайн-навчання ігрового ШІ має відповідати чотирьом обчислювальним і чотирьом функціональним вимогам. Вимоги до обчислень – швидкість, ефективність, надійність та ефективність. Функціональними вимогами є чіткість, різноманітність, послідовність і масштабованість. У роботі досліджується нова техніка онлайн-навчання для ігрового ШІ, яка називається «динамічним сценарієм», яка використовує адаптивну базу правил для створення ігрового ШІ на льоту. Ефективність динамічного написання сценаріїв оцінюється в експериментах, у яких адаптивні агенти протистоять набору вручну розроблених тактик у змодельованій комп'ютерній рольовій грі. Експериментальні результати показують, що динамічне написання сценаріїв успішно наділяє керованих комп'ютером суперників адаптивною продуктивністю. Для подальшого вдосконалення техніки динамічного сценарію досліджується модель, яка дозволяє масштабувати рівень складності гри ШІ до рівня навичок людини. Завдяки вдосконаленню динамічні сценарії відповідають усім обчислювальним і функціональним вимогам. Застосовність динамічного сценарію в найсучасніших комерційних іграх демонструється шляхом реалізації техніки в

грі Neverwinter Nights. Автори роблять висновок, що динамічні сценарії можна успішно застосувати для онлайн-адаптації ігрового ШІ в комерційних комп'ютерних іграх.

Динамічний сценарій відповідає вимогам швидкості, ефективності, надійності, чіткості та різноманітності. Таким чином, можна зробити висновок, що динамічні сценарії відповідають усім восьми вимогам, і, таким чином, можуть застосовуватися в реальних комерційних іграх для реалізації адаптивної гри ШІ. Висновок підтверджується хорошими результатами, досягнутими завдяки динамічному написанню сценаріїв у найсучаснішій CRPG Neverwinter Nights. Springer Mach Learn. У майбутній роботі пропонується дослідити ефективність і розважальну цінність динамічного сценарію в іграх, у яких грають реальні люди. Хоча таке дослідження вимагає багато суб'єктів і ретельного експериментального дизайну, досвід гри людей-гравців важливий, щоб переконати розробників ігор використовувати динамічні сценарії у своїх іграх. Нарешті, очікується, що, навіть, недовіра розробників і видавців до адаптивних методів швидко не зникне. Тому відзначається безризикова стадія розробки гри, де адаптивні методи можуть підтвердити свою цінність, а саме тестування гри. Під час ігрового тестування можна використовувати динамічні сценарії для розробки нових тактик для роботи з тактиками, які успішно використовують тестувальники, а також можуть виявляти можливі експлойти та проблеми з балансуванням гри в створеному вручну ігровому штучному інтелекті. Природно, якщо тестувальникам подобається грати проти адаптивної гри ШІ, розробники можуть розглянути можливість її включення у випущену гру [17].

1.2.2 Масштабування складності ігрового ШІ

Спронк та ін. запропонували модель «Масштабування труднощів» — це автоматична адаптація гри, щоб адаптувати виклик, який гра ставить перед людиною. Загалом, гра, рівень виклику якої відповідає навичкам гравця-людини (тобто «рівна гра»), вважається більш цікавою, ніж гра, яка є або занадто легкою,

або занадто важкою. На практиці, коли масштабування складності реалізується в грі, воно адаптує лише кілька параметрів. Навіть найсучасніші ігри не застосовують його до ігрового ШІ, тобто до поведінки керованих комп'ютером супротивників у грі. У попередній роботі була продемонстрована нова техніка онлайн-навчання під назвою «динамічне написання сценаріїв», яка здатна автоматично оптимізувати ігровий ШІ під час гри. У цій статті досліджується, якою мірою можна використовувати динамічні сценарії для адаптації ігрового ШІ, щоб отримати рівномірну гру. Автори досліджують три покращення масштабування складності в техніці динамічного написання сценаріїв, а саме покарання з високою придатністю, відсікання ваги та видалення зверху. Експериментальні результати показують, що видалення зверху особливо успішно для створення рівної гри. Можна зробити висновок, що динамічне написання сценаріїв із використанням верхнього вибракування може підвищити розважальну цінність ігор, масштабуючи рівень складності ігрового штучного інтелекту до ігрових навичок гравця-людини.

У цій статті запропонували три різні вдосконалення техніки динамічного написання сценаріїв, які дозволяють масштабувати рівень складності ігрового ШІ, а саме покарання з високою придатністю, відсікання ваги та відсіювання зверху. З наших експериментів ми робимо висновок, що вдосконалення масштабування складності для динамічного написання сценаріїв може бути використане, щоб дозволити процесу навчання масштабувати рівень складності тактики, яку використовують керовані комп'ютером супротивники, щоб відповідати ігровим навичкам гравця-людини (тобто, щоб примусити рівну гру). З трьох протестованих покращень масштабування труднощів, штрафування за високу придатність було, загалом, невдалим, але два інших показали хороші результати. Найкращі результати дала верхня вибракування. Також було виявлено, що як зменшення ваги, так і вибракування зверху, окрім забезпечення рівномірної гри, можна використовувати для встановлення іншого співвідношення виграш-програш, налаштувавши один параметр. Можна зробимо висновок, що динамічне

написання сценаріїв із використанням верхнього видалення може підвищити розважальну цінність ігор, масштабуючи рівень складності ігрового штучного інтелекту до ігрових навичок гравця-людини. У майбутній роботі автори мають намір застосовувати динамічні сценарії, включаючи масштабування складності, в інших типах ігор, ніж RPG. Також планується дослідити, чи можна використовувати офлайнові методи машинного навчання, які можуть бути дуже ефективними при розробці тактик [17], щоб «винайти» абсолютно нові правила для бази правил динамічного написання сценаріїв. Нарешті, автори прагнуть дослідити ефективність і розважальну цінність динамічного сценарію в іграх, у яких грають реальні люди. Хоча таке дослідження вимагає багато суб'єктів і ретельного експериментального дизайну, досвід гри людей-гравців важливий, щоб переконати розробників ігор використовувати динамічні сценарії у своїх іграх [18].

1.2.2 Масштабування складності ігрового ШІ

Класичні підходи до ігрового ШІ вимагають або високої якості знань у предметній області, або тривалого часу для створення ефективної поведінки ШІ. Ці дві характеристики перешкоджають досягненню мети створення складного ШІ. У цій статті Г Шасло та ін запропонували Monte-Carlo Tree Search як нову, уніфіковану основу для ігрового ШІ. Для прогнозування найбільш перспективних ігрових дій використовуються рандомізовані дослідження простору пошуку. Автори демонструють, що Monte-Carlo Tree Search можна ефективно застосувати до класичних настільних ігор, сучасних настільних ігор і відеоігор [19].

1.3 Постановка задачі

Задачею цієї магістерської роботи є дослідження та удосконалення математичних моделей та методів ігрового ШІ, та подальше створення інтелектуальної системи моделювання поведінки персонажів в комп'ютерних іграх.

Ігровий ШІ, як і всі інші аспекти відеоігри, існує для досягнення вищої мети відеоігри: бути цікавою та розважальною, іншими словами, суспільство часто вважає девізом ігрового ШІ «Якщо гравець не бачить цього, навіщо це робити?» [1]. Проте контекст нині не такий простий, почасти завдяки успішним продуктам, які висунули передовий ШІ, частково завдяки поширені онлайн-ігор для кількох гравців, які змінили звички гравців, гравці тепер більш вимогливі до ШІ, і все більше і більше розробників ігор переходять від використання сценарних і передбачуваних агентів до використання переважно людиноподібних агентів, здатних до навчання і, отже, за класичним визначенням ШІ, розумні. Ще одна особливість ігрового ШІ полягає в тому, що ігри, здебільшого, обмежені виконанням у режимі реального часу, тому залишається мало місця для повільних автономних методів. Цей факт робить ігровий ШІ подібним до ШІ, що використовується в робототехніці та системах керування [6], і це є причиною того, що можна використовувати, в тій чи іншій формі, ті ж самі прийоми, що використовуються у цих галузях.

Нарешті, ми повинні згадати про те, що відеоігри є частиною індустрії розваг, індустрії, добре відомої за її швидке просування, раптові повороти тенденцій і жорсткі графіки розвитку. Це залишає мало місця для дуже інноваційних, але не ретельно перевірених методів штучного інтелекту, які можуть приховуватись неприємні сюрпризи та розвиток обхідних шляхів. Через ці відмінності ігрове поле штучного інтелекту просувається в іншому але паралельному напрямку по відношенню до академічного ШІ, запозичення ідей і методів з останнього, зберігаючи практичність підходів на увазі [1, 7].

Висновки до розділу 1

У першому розділі дипломної роботи були проведені аналіз предметної сфери, огляд та аналіз наявних аналогів, та розроблені вимоги до дипломної роботи.

Були досліджені та описані основні принципи та технології, що використовуються у ігровому ШІ, оцінена перспективність цієї галузі та проведено аналіз основних напрямків розвитку. Спираючись на результати аналізу можна зробити висновок, що головними проблемами ігрового ШІ є високий рівень передбачуваності поведінки агента, та швидкість обчислень.

Аналіз аналогів та публікацій показав наявність декількох варіантів вирішення проблеми передбачуваності поведінки, які можуть бути використані під час процесу гри, не критично впливаючи на швидкість обчислень.

2 МАТЕМАТИЧНІ МОДЕЛІ, МЕТОДИ, ІФОРМАЦІЙНІ ТЕХНОЛОГІЇ ДЛЯ ВИРІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

2.1 Нечітка логіка

Нечітка логіка — це надмножина традиційної логіки, яка була розширена, щоб обробити концепцію значень часткової істини між булевою дихотомією істини та хибності. Нечітка логіка зазвичай приймає форму нечіткої системи міркувань, а її компонентами є нечіткі змінні, нечіткі правила та механізм нечіткого висновку. Використовуючи теорію нечітких множин, змінні фазифікують шляхом вибору набору функцій належності для їх можливого діапазону значень. Функції належності відображають чітке значення змінної як лінгвістичну мітку із ступенем істинності, зазвичай в діапазоні [0,1]. Наприклад, дальність дії зброї в грі можна розділити на рукопашну, дальну або позадистанційн. Змінні можна вважати вхідними, наприклад вимірювання від датчиків (навіть віртуальних датчиків, як в іграх), або їх можна вважати вихідними. Нечіткі правила можна створити за допомогою визначених нечітких змінних та їх наборів. Ці правила зазвичай мають форму if-then, з граматикою, схожою на булеву логіку. Правила визначають значення вихідних змінних з урахуванням поточного значення вхідних змінних. Операторам, таким як І, АБО та НЕ, також надається значення в нечіткій логіці. Після створення правил використовується механізм нечіткого висновку, щоб зробити висновки з поточних значень змінних і правил, які керують системою. Таким чином, вихідним змінним надається нечітке значення. Зрештою, можна виконати процедуру дефазфікації, перетворюючи вихідний результат нечіткого двигуна до чіткого значення.

Теорія нечітких множин була введена в 1965 році Лотфі А. Заде в академічній галузі штучного інтелекту [27], але його ідеї не заслужили прихильників, поки японські дослідники не продемонстрували практичне використання нечітких систем керування. Нині багато контролерів використовують нечітку логіку, від посудомийних машин до електричних вимикачів до зависаючих вертольотів. Нечітка логіка з великим успіхом

використовується в області систем управління через її схожість з людськими міркуваннями, що дозволяє фахівцям у галузі, не обов'язково програмістам, брати участь у процесі проектування. Нечітка логіка також використовується в поєднанні з іншими методами ШІ, такими як еволюційні алгоритми або нейронні мережі, у навчанні та класифікації.

Як і багато інших академічних методів штучного інтелекту, нечітка логіка була перевірена у відеоіграх у пошуках простого дизайну в поєднанні з інтелектуальними агентами. Нечітка логіка задовольняє ці вимоги, оскільки її досить просто обґрунтувати через її мовну природу, що призводить до швидкої та зручної методології проектування [28].

Нечітка логіка була офіційно введена в розробку ігор у 1996 році в журналі Game Developer Magazine Ларрі О'Браєном [29] і з тих пір досліджувалася та вдосконалювалася іншими авторами. Нечітка логіка перелічена як одна з корисних технік для розробки ігрового ШІ у кількох джерелах. У книгах з розробки ігрового ШІ присвячено цілі розділи [30, 31, 32] з великою кількістю прикладів, тоді як кілька статей можна знайти як вступ до застосування нечіткої логіки до ігор, одним із таких прикладів є [33]. Зарозінський припускає, що нечітка логіка знаходить свій шлях майже в кожній грі [34].

Нечітка логіка може бути корисною для ігор ШІ в кількох аспектах. Серед інших застосувань, її можна використовувати для прийняття рішень НІП, таких як вибір предметів або зброї, для контролю переміщення одиниць, подібно до того, що відбувається з системами управління, для надання можливості ШІ противнику оцінити загрози та для класифікації, наприклад, за рейтингом, гравців і НІП з точки зору здоров'я або сили, використовуючи нечіткі змінні.

Нечітка логіка приносить багато переваг розробнику ШІ гри. Основа нечіткої логіки проста, і для неї немає жодних передумов, окрім базової булевої логіки, яку напевно освоїть будь-який розробник ШІ, що робить нечітку логіку хорошим кандидатом для впровадження розширеного ШІ в гру з відносно невеликими зусиллями. Як бонус, функції нечіткої належності можна визначити

за допомогою тих самих кривих[35], які зазвичай використовуються для налаштування більш простих способів поведінки у відеоіграх.

Через лінгвістичну природу нечіткої логіки формулювання її правил може бути виконано експертами в цій галузі, а нечітка система може бути використана для емуляції міркування експерта [33]. Це велика перевага в порівнянні з іншими методами, які вимагають знання самого методу для налаштування. У сфері розробки відеоігор ця властивість стає в нагоді, оскільки нечіткі правила можуть створюватися розробниками ігор без необхідності допомоги програміста, подібно до того, що вже відбувається в ігровій індустрії, коли використовують мови сценаріїв для розробки ігрових послідовностей. Проте має бути подано попередження, оскільки для створення складних ворогів у грі експерт має відповідати за визначення бази правил, але перед випуском гри це може бути неможливо. У зв'язку з нещодавнім зростанням вмісту, створеного користувачами, цей крок можна зробити навіть на бета-фазі гри, давши завдання самому гравцеві.

Традиційне прийняття рішень, коли використовується для моделювання поведінки агента, може призвести до неприродних раптових перемикань від одного рішення до іншого або від одного стану до іншого в FSM. Більш поступових змін можна досягти за допомогою нечіткої логіки. Наприклад, автомобіль, керований ШІ без нечіткої логіки, може гальмувати або прискорюватися, але з нечіткою логікою він також може вирішувати, наскільки гальмувати і наскільки прискорюватися, досягаючи більш природної поведінки. Припускається, що такий поступовий вихід може бути надмірним для більшості сучасних ігор [30], але, тим не менш, це є перевагою нечіткої логіки. Ця поступова зміна поведінки також відбувається в деяких іграх, не вдаючись до нечіткої логіки, хоча нечітка логіка дає кращу основу для роботи з такими безперервними значеннями.

Нечіткі методи дозволяють будувати взаємозв'язок введення-виведення на основі знань експерта, без необхідності складної математичної моделі, яку може

бути стомлююче або неможливо отримати [28]. Завдяки свободі у формі функцій належності, цей зв'язок введення-виведення може бути розроблений як нелінійний, ефективно дозволяючи нелінійній поведінці з'являтися в грі і таким чином підвищуючи непередбачуваність NPC.

Ще одна перевага при розробці набору правил для нечіткої системи полягає в непослідовному підході традиційної нечіткої логіки. Оскільки правила можна розв'язувати в будь-якому порядку, набагато легше видалити або додати нові правила до нечіткої системи, ніж до типового вкладеного блоку if-then-else.

Нечітка логіка може бути використана для моделювання складної поведінки з низькою вартістю обчислень [36]. Через низькі ресурси, доступні розробникам ігор ШІ, і обмеження в реальному часі, яких вони часто повинні дотримуватися, ця перевага дуже важлива і є однією з причин, чому нечітка логіка так користується перевагою в ігровій індустрії.

Однак нечіткі системи мають свої власні проблеми. Завдяки своїй природі, що ґрунтується на знаннях, нечітка логіка вимагає правильного визначення вхідних і вихідних змінних, а також їх взаємозв'язків. Якщо не вдасться знайти фахівця в даній галузі, буде важко придумати правила і може знадобитися багато настроювання, що призведе до збільшення часу розробки.

Ще одним недоліком є те, що розробка нечіткої системи для гри з багатьма керованими комп'ютером агентів, якщо її не ретельно розробити, може призвести до перевірки сотень правил на кожному кроці часу, повністю позбавляючи переваг низької обчислювальної вартості програми. Хоча найпростіша версія нечіткої системи з'являється в більшості ігор, через цей недолік були запропоновані вдосконалення, що відповідають природі ігрового ШІ [37]. Як рішення ми згадуємо вихідні дані з одним станом, щоб уникнути непотрібних обчислень, ієрархічну поведінку для одночасного вирішення груп правил і паралельні та незалежні рівні поведінки з різними частотами оцінки.

Один конкретний випадок недоліку багатьох правил у нечіткій логіці називається комбінаторним вибухом [38]. У типовій відеоігрі може бути багато

вхідних змінних для поведінки агента, кожна з яких має ряд нечітких наборів. Якби ми прагнули до повноти при створенні правил нечіткої системи, ми б мали експоненційне зростання правил щодо нечітких змінних і множин. Як приклад, нечітка система з 5 вхідними змінними, кожна з яких має 5 наборів членства, потребує 3125 правил. Це зробить нечітку систему занадто важкою для обчислень у реальному часі. Розробники ігрового ШІ пропонують застосувати метод Комбса [39], який дозволяє лінійне зростання правил щодо кількості змінних і наборів. Зауважте, однак, що отримана система буде наближеною до вихідної, що дасть дещо інші результати для тих самих вхідних значень. Крім того, існуючу базу нечітких правил неможливо легко трансформувати в нотацію Комбса, але замість цього правила потрібно продумувати з нуля.

Нечітка логіка часто з'являється в іграх під виглядом нечітких автоматів стану (FuSMs), методики, про яку повідомляють декілька авторів [40, 30, 41, 42]. Впровадження FuSM дозволило програмістам ігор розширити свій метод вибору, кінцеві автомати, з перевагами нечіткої логіки. Відомо, що FSM вибухають у складності, оскільки додається більше станів, і нерідко трапляється, що багато станів потрібні для складних символів. Згідно з [41], FuSMs дозволяють створювати тонку, менш передбачувану поведінку з меншою кількістю станів і, таким чином, меншою складністю, ніж базові FSM. Результатом є те, що розробники можуть використовувати добре відому структуру кінцевого автомата з її простотою реалізації та її структурною потужністю в поєднанні з непередбачуваністю та динамізмом, які визначають нечітку логіку.

Існуючі FSM можна легко трансформувати в FuSM, а це означає, що час розробки не страждає від трансформації, перевага, з якою розробники ігор дуже раді стикатися. Існує два основні методи перетворення FSM в FuSM. Перший метод використовує переходи нечітких станів, переходи, викликані нечіткими міркуваннями, і є простими у реалізації. Другий метод використовує нечіткі стани, що означає, що агент може одночасно перебувати в різних станах з різним ступенем приналежності. У цьому останньому випадку нечіткі переходи

змінюють ступінь належності для кожного стану агента. Цей метод може бути корисним для моделювання емоційного стану агента, який може перебувати в стані гніву, смутку чи щастя різного ступеня [43]. Останній дуже простий метод додавання нечіткої логіки до FSM, згаданий у [30], полягає в додаванні нового стану, який використовує нечітку логіку, щоб вирішити, до якого іншого стану перейти, не змінюючи решту FSM.

Переглядаючи літературу, ми можемо знайти кілька згадок про використання нечіткої логіки у відеоіграх. Через їх різний контекст ми можемо відокремити приклади, зібрані з галузі, від тих, що містяться в академічній літературі.

Враховуючи той факт, що про використання нечіткої логіки в іграх написано багато літератури, дивно, що мало згадується про реальні комерційні ігри, які спеціально її реалізують. Особливо важко знайти згадку про це в останніх іграх. Ми вважаємо, що нечітка логіка, через просту форму, яка з'являється у ШІ у відеоіграх, і через високий інтерес до більш екзотичних прийомів, на жаль, більше не вважається гідною фанфар при застосуванні. Тим не менш, можна знайти кілька менш свіжих прикладів. Повний список ігор, що використовують цікаві методи ШІ, можна знайти в [44].

Unreal є одним із найвідоміших шутерів від першої особи в історії відеоігор, і, як повідомляється, використовує FuSM для контролю поведінки ворожих інопланетян. Гра отримала похвалу під час випуску за її правдоподібно ШІ [45, 45].

BattleCruiser: 3000AD — це гра в космічній стратегії зі суперечливою історією розробки, яка мала використовувати нечітку логіку разом із нейронними мережами для керування неграровими персонажами в грі [42].

S.W.A.T. 2, це тактична гра в реальному часі, яка, як повідомляється, широко використовує нечітку логіку, щоб дозволити персонажам, які не грають, вести себе спонтанно на основі їх визначених особистостей і здібностей [46, 42].

Civilization: Call to Power, покрокова стратегічна гра, яка є допоміжним продуктом дуже відомої франшизи, використовує FuSM для встановлення пріоритетів для ШІ стратегічного рівня, що дозволяє визначити риси особистості для різних лідерів цивілізації [46].

Close Combat9 та його продовження Close Combat 210 використовують FuSM, який зважує сотні змінних за допомогою багатьох формул, щоб визначити ймовірність виконання певної дії [42, 47].

Enemy Nations містить ворогів, керованих штучним інтелектом, які використовують системи кінцевих і нечітких станів і базу даних цілей і завдань [44].

Найпопулярніша в усьому світі гра The Sims використовує FuSM, щоб визначити, з якими об'єктами може взаємодіяти персонаж Сіма, на основі їхніх властивостей і рис особистості Сіма, у поєднанні з розумним двигуном ландшафту, створеним відомим дизайнером Віллом Райтом [47].

The Chronicles of Jaruu Tenk використовує технології A-Life поряд із інтенсивним використанням каскадних нечітких автоматів для забезпечення поведінки своїх створінь [44].

З наведених прикладів ми бачимо, що нечітка логіка в іграх успішно використовується в індустрії, хоча розробники, як правило, не йдуть далі простих механізмів висновку або FuSM.

Натомість ми знаходимо іншу ситуацію в дослідницькій галузі, де можна знайти різноманітні застосування нечіткої логіки та систем, хоча і з меншою частотою, ніж більш популярні методи еволюційних алгоритмів і нейронних мереж. Дослідження штучного інтелекту в іграх також часто використовується разом з іншими передовими методами.

Часто через неможливість отримати вихідний код комерційних ігор дослідники намагаються застосувати свої методики до ігор, створених спеціально для дослідження, або до клонів відомих ігор.

Як і в промисловості, нечітка логіка використовувалася в дослідженнях для моделювання поведінки керованих комп'ютером агентів і НІП. Нечітка логіка використовується для проектування поведінки ворожих привид у клоні Pac-Man, хоча для досягнення розумної поведінки необхідна важка настройка [48]. Fuzzy Q-навчання, запозичене з сфер робототехніки, використовується в клоні Ms. Pac-Man [49]. Лі та ін. [36] згадують нечіткий контроль як практичний метод генерування витонченої поведінки та використовують його в структурі Belief-Desire-Intention (BDI) як частину прийняття рішень для гри-клонів BattleCity. Хо та ін. використовувати контекстно-залежний нечіткий контролер для маневрування автомобіля в змодельованих середовищах TORCS з емпірично обраними нечіткими наборами [50]. Перес та ін. використовувати еволюційні алгоритми для розробки параметрів тієї ж нечіткої системи з початкового припущення [51]. Кардемон та ін. найняти експерта, який відповідає за розробку правил подібного нечіткого контролера [52]. На сьогоднішній день налаштований нечіткий контролер все ще перевершує інші контролери.

Високий інтерес представляють застосування нечіткої логіки поряд або замість штучного інтелекту оригінальних комерційних ігор. Оскільки ШІ застосовується до добре відомих ігор, легше знайти досвідчених гравців, щоб перевірити його. Крім того, потужний ШІ, застосований для комерційного успіху, швидше за все, вплине на галузь. Для цього потрібно, щоб ігровий ШІ був розширюваним, функція, яка є лише в кількох іграх. Код Quake III Arena, добре відомого шутера, доступний у відкритому доступі, і тому гра була об'єктом багатьох досліджень. Для гри були випущені боти, керовані нечіткою логікою [53], з вибором зброї та предметів, контрольованим шляхом прийняття нечітких рішень. Ці контролери беруться за основу для порівнянь, коли використовуються більш досконалі методики [54, 55]. Pinto моделює нечіткі датчики як вхідні дані для розширених мереж поведінки в Unreal Tournament, іншому відомому шутері від першої особи [56], тоді як Acampora використовує синхронізовані автомати та

нечіткі контролери для моделювання емоцій для ботів у матчі Unreal Tournament 2004 [57].

Додаткове використання нечіткої логіки в іграх стосується класифікації зворотного зв'язку гравця та навчання від гравця. У цих контекстах нечітка логіка зазвичай вибирається для моделювання гравця (або віртуального суперника) через її силу як метод моделювання емоцій. Ель-Наср повідомляє, що нечітка логіка надала кращий засіб моделювання емоцій завдяки своїй якісній та кількісній виразності [43]. Він застосовує нечітку логіку, щоб визначити емоції для віртуального вихованця, а потім вводить навчання з підкріпленням, щоб адаптувати поведінку вихованця до конкретного гравця. LevillШIn та ін. використовуйте нечіткі дерева рішень, щоб класифікувати емоційний відгук гравців під час гри[58].

Вонг та ін. Використовуйте нечіткий контроль для управління складністю сцени, використовуючи нечітку класифікацію для визначення рівня деталізації для 3D-моделей [28]. Ohsone та ін. використовувати нечіткі дерева рішень, щоб визначити найкращу відповідь свого віртуального опонента [59]. Також досліджувалося вивчення наборів членства або, що частіше, нечітких правил. Завдяки своїй природі, нечіткі системи є хорошими кандидатами для навчання і, таким чином, поєднуються з різними методами для його забезпечення. Наприклад, нечітка логіка використовується для моделювання правил, які потім розвиваються з використанням ігрового процесу гравця як вхідних даних для еволюційного алгоритму [60]. В іншій роботі автори витягують дані з ітераційного виконання ігор, а потім вивчають нечіткі правила для класифікації дій гравців [61].

2.2 Нейронні мережі

До появи нейронних мереж розробники використовували декілька методів розробки, щоб створювати інтерактивні агенти штучного інтелекту. Один з використовуваних методів – скінченний автомат. Скінченні автомати - один з

найбільш ранніх методів опису поведінки, але вони все ще широко використовуються сьогодні [62]. Припустимо, було розроблено скінченний автомат для крамаря в магазині. Власник магазину може байдикувати, розмовляючи з покупцем, або збираєтеся принести товар для покупця. У нас буде кілька переходів між цими станами. Наприклад, в режимі очікування, коли покупець підходить до продавця, він починає розмову, і як тільки покупець запрошує товар, він його приносе.

Використання скінченних автоматів дуже корисно для програмування систем, коли бажана поведінка може бути розбита на набір станів і переходів. Однак, оскільки скінченний автомат є детермінованим, поведінку неігрового персонажу, керованого скінченним автоматом може стати передбачуваною після декількох зустрічей з одним і тим же агентом. Також поведінку агенту задано і вона не може змінитися для нових умов.

Ще один інструмент, який розробники можуть використовувати для ШІ, - це система, заснована на правилах. В системі є визначений набір правил, які диктують правильні дії для агенту. Розглянемо, наприклад, ворога в бойовій грі. Припустимо, розробник вирішить, що противник не діятиме велику частину часу, крім випадків, коли гравець знаходиться в межах видимості, тоді ворог готуватиметься до бою. Таким чином, розробник визначив набір правил, які диктують ворожий персонаж. Цей метод простий, оскільки його легко реалізувати; проте часто стає передбачуваним, якщо гравець стикається з аналогічним неігровим персонажем безліч разів. Крім того, як і скінченні автомати, системи на основі правил не можуть пристосуватися до нових ситуацій, оскільки вони не можуть змінити свої правила самостійно.

Щоб додати трохи випадковості в поведінку ігрових персонажів, розробники можуть використовувати генетичний алгоритм, щоб зробити агентів менш передбачуваними. У генетичному алгоритму, розробник імітує процес, за допомогою якого природний відбір визначає, які гени необхідно передати [63]. При цьому розробник визначає набір параметрів, такі як точність, баланс,

спритність, зростання, володіння зброєю, а потім він генерує початкову сукупність випадково модульованих параметрів. Як тільки у нього з'явиться популяції цих геномів, він оцінює корисність або придатність для кожного потенційного персонажу і рекомбінує особливості геномів з більш високим рейтингом придатності. Цей підхід дозволяє персонажам бути менш передбачуваними, оскільки у кожного з них є свої особисті параметри. Крім того, оскільки генетичні алгоритми рекомбінують геноми для налаштування параметрів, ці системи можуть «навчитися» пристосовуватися до навколишнього середовища, щоб згодом працювати краще. Однак цей метод вимагає великої обчислювальної потужності, щоб визначити параметри для персонажів.

Подібно генетичним алгоритмам, нейронні мережі імітують природні процеси, щоб додати рівень непередбачуваності і реалістичності штучного інтелекту. У людському мозку існують мільярди нейронів, які отримують сигнали від оточуючих нейронів і передавати сигнали іншим нейронам [63]. Зв'язок між цими нейронами називають синапсами.

Було показано, що нейронні мережі дуже добре описують безліч складних взаємодій. Проблема з цією технікою полягає в тому, що часто буває важко визначити, як мережа повинна бути побудована так, щоб вона показувала бажану поведінку. Існують процеси визначення того, як нейронна мережа повинна бути побудована. Таким чином, використовуючи нейронні мережі, ми можемо кодувати систему, яка може демонструвати складну поведінку, а також обробляти нову інформацію за допомогою навчання. Це представляє для нас особливий інтерес, оскільки ми можемо використовувати цю технологію для розробки ігрових персонажів, які можуть вчитися і взаємодіяти більш реалістично. Однак проблема з цією технологією полягає в тому, що використовувати таку систему набагато складніше, ніж системи на основі правил або скінченний автомат і вимагає, як і генетичні алгоритми, більше обчислювальної потужності, щоб робити висновки.

Одним з поширених типів нейронних мереж є нейронна мережа з прямим зв'язком і зворотним поширенням. У мережі такого типу вузли згруповані по шарам, де перший шар вхідний шар, останній шар - вихідний шар, а всі проміжні шари називаються приховані шари [63]. Кожен нейрон в шарі може бути підключений тільки до нейронів в попередньому шарі, і всі вихідні дані нейронів в шарі можуть бути отримані тільки нейронами в наступних шарах.

Модель створена для навчання НІП з використанням глибоких нейронних мереж і навчання з підкріпленням. Модель може навчатися рухам після того, як гравець взаємодіє, використовуючи методи навчання з підкріпленням. Однак складність цієї моделі може привести до затримок в грі через велику часу обробки, що ускладнює процес гри. [64]

Модель побудована з використанням кінематики, нечіткої логіки, об'єднання датчиків, класифікатора ШНМ і алгоритмів кластеризації. Вхід нормалізується, після чого відбувається фазифікація. Використовуючи нечіткі правила, робиться висновок, потім з допомогою дефаззифікації і денормалізації ми отримуємо результат. Складність цієї моделі може привести до затримок в грі через великий час обробки, , що ускладнює процес гри. [65]

Модель побудована з використанням нейронної мережі Deep Convolution, мереж Deep-Q і Asynchronous Advantage Actor Critic (AAAC). Ця модель працює тільки для розрахованої на багатокористувацької бойової арени і не може бути реалізована для ігор розрахованих на одного користувача або багатокористувацьких ігор інших жанрів. [66]

Модель побудована з використанням Deep-Q Networks і використовує ігрові дані гравця. Кількість використовуваної обчислювальної потужності дуже велика для простих ігор, вона буде збільшуватися зі складністю комп'ютерної гри, що призведе до затримок в гри. Це відставання ускладнює процес гри. [67]

Модель побудована з використанням евристичних алгоритмів. Модель використовує евристичні алгоритми для прийняття рішення про те, які кроки зробити. Однак існує висока залежність від правильності евристичних алгоритмів,

оскільки втрата будь-якого необхідного випадку з бази знань призведе до зниження якості ІІ. Також цей підхід обмежений настільними іграми. [68]

2.3 Скінченні автомати

Автомат - це математична модель обчислень. Це абстрактна концепція, згідно з якою автомат може мати різні стани, але в даний момент виконує лише один із них. Існують різні типи автоматів. Найвідомішою, на мою думку, є машина Тьюринга. Це нескінченний автомат, що означає, що він може мати незліченну кількість станів. Машина Тьюринга погано вписується в сучасну розробку інтерфейсу користувача, оскільки в більшості випадків ми маємо кінцеву кількість станів. Ось чому скінчені автомати, такі як Мілі та Мур, мають більше сенсу.

Різниця між ними полягає в тому, що автомат Мура змінює свій стан лише на основі свого попереднього стану. На жаль, у нас є багато зовнішніх факторів, таких як взаємодія користувачів і мережеві процеси. Навідміну від автомату Мілі, який має початковий стан, та переходить до нових станів на основі введення та поточного стану.

Один із найпростіших способів проілюструвати, як працює скінчений автомат, — це подивитися на турнікет. Він має кінцеву кількість станів: заблокований і розблокований. Ось простий графік, який показує нам ці стани з їх можливими входами та переходами (див. рис. 2.1).

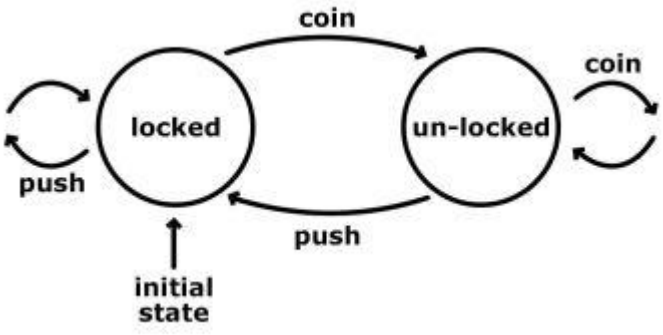

Рис. 2.1. Графік скінченого автомату турнікета

Початковий стан турнікета заблокований. Незалежно від того, скільки разів ми можемо натиснути його, він залишається в такому заблокованому стані. Однак, якщо ми передаємо йому монету, то він переходить у розблокований стан. Інша монета в цей момент нічого не дасть; він все ще буде в розблокованому стані. Поштовх з іншого боку спрацює, і ми зможемо пройти. Ця дія також переводить машину в початковий заблокований стан.

Якби ми хотіли реалізувати єдину функцію, яка керує турнікетом, ми, ймовірно, отримали б два аргументи: поточний стан і дію. Це схоже на добре відому функцію редуктора, де ми отримуємо поточний стан і на основі корисного навантаження дії вирішуємо, яким буде наступний стан. Редуктор — це перехід у контексті кінцевих автоматів. Насправді будь-яку програму, яка має стан, який ми можемо якимось чином змінити, можна назвати автоматом станів. Просто ми впроваджуємо все вручну знову і знову.

Висновки до розділу 2

Нечітка логіка досліджувалася як у дослідженнях, так і в промисловості для використання у відеоіграх і, зокрема, для моделювання поведінки агентів. Сьогодні це добре відома техніка, якою повинен володіти кожен розробник ігор зі штучним інтелектом.

Нечітка логіка приносить багато переваг для моделювання інтелектуальних ігрових агентів. Його головною перевагою є простота формулювання, яка в поєднанні з характером введення-виводу дозволяє розробникам легко інтегрувати його в багато ігор, що є величезною перевагою, якщо врахувати жорсткі графіки розробників ігор.

Навіть у своїй простоті нечітка логіка може бути потужною технікою ШІ, особливо завдяки можливості моделювати нелінійності, досягати складної поведінки за допомогою простих правил, бути кандидатом на навчання та імітувати людські міркування та емоції.

Модель, заснована на нейронних мережах і генетичному алгоритмі, може замінити дерева поведінки при розробці комп'ютерних ігор. Модель немає видимого відставання і є неглибока крива навчання для системи ШІ, оскільки вона починає отримувати хороші результати досить рано, що в основному пов'язано з алгоритмом нейронної мережі, який збільшує швидкість навчання. Обмеження полягає в тому, що для використання цієї моделі в якості мозку для NPC потрібно попереднє навчання, яке може займати багато часу, що робить її не надто практичною в сучасному світі. Якщо цей час навчання якості продукції можна скоротити, то ця модель має серйозний потенціал для комерційного використання.

3 МОДЕЛЮВАННЯ ТА ДОСЛІДЖЕННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

3.1 Методи для створення моделі

Для виконання поставленої задачі було вирішено використати технологію дерев поведінки в поєднанні з методами нечіткої логіки. Саме поєднання цих двох методів може створити одночасно просту та менш передбачувану поведінку неігрових персонажів. Для досягнення поставленої задачі було вирішено використовувати алгоритм Мамдані.

3.1.1 Алгоритм Мамдані

Алгоритм Мамдані — алгоритм нечіткого логічного виводу по базі знань. Нечіткі системи Мамдані спочатку були розроблені, щоб імітувати роботу людей-операторів, відповідальних за контроль певних промислових процесів.

Мета полягала в тому, щоб узагальнити досвід оператора черкз набір (лінгвістичних) правил IF-THEN, які могли б використовуватися машиною для автоматичного керування процесом. Зокрема, використовуючи такий набір правил IF-THEN, нечітка система Мамдані визначає функцію f, яка генерує числові результати y = f(x) з (зазвичай числових) вхідних значень x. Тут ми представляємо скорочений і спрощений виклад методу.

Системи Мамдані складаються з правил IF-THEN у формі «ЯКЩО X є A, ТО Y є B», наприклад «Якщо ТИСК ВИСОКИЙ, ТО ОБ'ЄМ МАЛИЙ». Частина IF «X є A» називається антецедентом правила, а частина THEN «Y є B» називається наслідком правила.

Для простоти викладу методу та прикладів припустимо, що X і Y (ТИСК і ОБ'ЄМ відповідно у прикладі вище) є числовими змінними, визначеними на реальних інтервалах. Приклади, які ми надаємо, можна легко адаптувати до інших просторів введення та виведення, кількох входів або нечітких входів. Таким чином, відтепер передбачається, що змінна X визначена в реальному інтервалі, який ми називаємо інтервалом введення, тоді як змінна Y вважається визначеною

в реальному інтервалі, який ми називаємо вихідним інтервалом. Для позначення конкретних значень змінних X і Y будемо використовувати стрічні літери x і y.

3.1.2 Дерева поведінки

Дерева поведінки (ДП) були винайдені як інструмент для включення модульного ШІ в комп'ютерні ігри, але в останнє десятиліття вони привертають все більше уваги в спільноті робототехніки. Зі зростанням вимог до складності агентного штучного інтелекту ігрові програмісти виявили, що використовувані ними скінченні автомати (FSM) погано масштабуються і їх важко розширити, адаптувати та повторно використовувати.

У ДП логіка переходу станів не розподілена між окремими станами, а організована в ієрархічну деревоподібну структуру, зі станами як листами. Це має значний вплив на модульність, що, у свою чергу, спрощує як синтез, так і аналіз як людьми, так і алгоритмами.

Вперше ДП були задумані програмістами комп'ютерних ігор. Оскільки важливі ідеї були розповсюджені лише в частково задокументованих публікаціях у блозі та презентаціях на конференціях, дещо незрозуміло, хто першим запропонував ключові ідеї, але важливі віхи були визначено в роботі Майкла Матеаса та Ендрю Стерна [69] та Деміан Ісла. Потім ідеї поширювалися та вдосконалювались у суспільстві протягом кількох років, коли перша журнальна стаття про ДП з'явилася в [70]. Перехід від ігрового штучного інтелекту до робототехніки був ще пізніше і незалежно описаний у [71, 72].

ДП — це спрямоване дерево, де ми застосовуємо стандартні значення кореневих, дочірніх, батьківських і листкових вузлів. Листові вузли називаються вузлами виконання, а нелистові вузли називаються вузлами потоку керування. Графічно ДП малюється або з коренем ліворуч, а діти праворуч, або з коренем зверху і дочірніми внизу.

Виконання ДП починається з кореневого вузла, який генерує сигнали, які називаються Ticks з заданою частотою. Ці сигнали уможливлюють виконання

вузла, а потім поширюються до одного або кількох дочірніх елементів позначеного вузла. Вузол виконується тоді і тільки тоді, коли він отримує Ticks.

У класичному формулюванні існують три основні категорії вузлів потоку керування (послідовність, резервний і паралельний) і дві категорії вузлів виконання (дія та умова).

Послідовності використовуються, коли деякі дії або перевірки умов мають виконуватися послідовно, і коли успіх однієї дії необхідний для виконання наступної. Вузол Sequence направляє Ticks до своїх дітей зліва, доки не знайде дочірнього, який повертає FШIlure або Running, а потім повертає FШIlure або Running відповідно до свого власного батька, Він повертає Success тоді і тільки тоді, коли всі його дочірні елементи повертають Success. Зауважте, що коли дитина повертає Running або FШIlure, вузол Sequence не направляє Ticks до наступної дитини (якщо є). У випадку «Біг» дитині дозволяється керувати роботом, тоді як у випадку «Відмови» може бути виконана зовсім інша дія або взагалі не виконуватися, якщо вся ДП повертає «Відмову».

Резервні варіанти використовуються, коли набір дій представляє альтернативні шляхи досягнення подібної мети. Таким чином, резервний вузол направляє Ticks до своїх дочірніх елементів зліва, доки не знайде дочірнього, який повертає або Success, або Running, а потім повертає Success або Running відповідно до свого власного батька. Він повертає FШIlure тоді і тільки тоді, коли всі його діти повертають FШIlure. Зауважте, що коли дитина повертається Running або

Паралельні вузли відзначають усіх дітей одночасно. Потім, якщо M з N дочірніх повертає Success, то паралельний вузол також повертає. Якщо більше ніж N * M повертає FШIlure, що робить успіх неможливим, він повертає FШIlure. Якщо жодна з наведених вище умов не виконується, він повертається.

До ДП кінцевий автомат (FSM) довгий час використовувався як спосіб організації перемикання завдань, і ДП можна розглядати як FSM зі спеціальною структурою. Взаємозв'язок між FSM і ДП було додатково досліджено в [73]. У

цих роботах було показано, як створити ДП, який працює як FSM, відстежуючи поточний стан як зовнішню змінну на дошці, і FSM, який працює як ДП, дозволяючи всім галочкам і статусам повернення відповідати переходи станів. Таким чином, обидві структури еквівалентні з точки зору того, яку загальну поведінку можна створити, майже так само, як алгоритм може бути реалізований у будь-якій загальній мові програмування. Однак з практичної точки зору відмінності часто бувають істотними.

FSM використовують переходи стану, які передають керування агентом у спосіб, дуже схожий на оператор GOTO, який тепер скасований у мовах програмування високого рівня. На відміну від цього, передача керування в ДП нагадує виклики функцій, з піддеревами, які відзначаються і повертають успіх, невдачу або виконання. Крім того, припускаючи n можливих дій (представлених у вигляді станів), n2 можливих переходів FSM швидко перетворюються на єдину велику монолітну структуру, яку дуже складно налагоджувати, оновлювати та розширювати. Для ДП, з іншого боку, кожне піддерево є природним рівнем абстракції, причому всі піддерева мають один і той же інтерфейс вхідних тиків і вихідних статусів повернення.

Крім FSM, було досліджено взаємозв'язок між іншими архітектурами комутації та ДП. Це було показано в [74] як дерева рішень та архітектура субсумпції узагальнені ДП і в [75] як телеореактивний підхід [76] і AndOrTrees узагальнені.

Стандартний резервний вузол в Algorithm2 має статичний порядок пріоритетів усіх своїх дочірніх, і декілька авторів зазначали, що є багато випадків, коли має сенс оновлювати цей порядок на основі стану світу. Один із таких підходів, Utility ДПs, був запропонований у [77]. Тут кожне піддерево ДП може повертати значення корисності, і існують спеціальні резервні утиліти, які сортують своїх дітей на основі цього показника корисності. Тоді листові вузли повинні реалізувати таку оцінку корисності, тоді як інші внутрішні вузли повинні мати можливість агрегувати корисність на основі своїх дочірніх. Це можна

зробити багатьма способами, і запропонований у [77] повідомляє про найвищу дочірню корисність як корисність як резервних, так і послідовностей.

У [78]. Тут система збирає дані про результати успіху/невдачі всіх відпусків. Таким чином, він може оцінити ймовірність успіху кожного аркуша та відповідно змінити порядок дочірніх запасних. Цей результат керується даними, тому його також можна вважати підходом до навчання.

Враховуючи таку статистику щодо ймовірностей успіху та невдачі листків, природно розширити ймовірність успіху та невдачі цілих піддерев, як у [79]. Крім того, якщо дані оцінки часу виконання листових вузлів, з точки зору функцій щільності ймовірності з часом для успіху та невдачі, ці оцінки можна агрегувати вгору і відображати оцінки часу виконання для всіх піддерев. Нарешті, стохастичний аналіз ДП зроблений на один крок далі в [80], де приховані моделі Маркова (НММ) підключаються до ДП, де під час виконання доступні лише шумні спостереження. Враховуючи дані спостережень, показано, як інструменти НММ можуть бути застосовані для оцінки ймовірностей переходу станів ДП, щоб оцінити, які переходи найбільш вірогідні для певних даних і наскільки ймовірним є вихід деяких даних за набору параметрів.

Вузол послідовності в алгоритмі1є реактивним у тому сенсі, що він постійно переоцінює своїх дітей. Це іноді може викликати проблеми при виконанні дій, які не залишають сліду їх успішного завершення. Приклад цього був описаний у [81] з агентом, що йде за маршрутом точки маршруту. Після проходження маршрутної точки агент перемикається на наступну. Однак, якщо умовою успіху є відстань до маршрутної точки, перша дія не поверне Успіх при виході з першої точки маршруту на шляху до другої. Одним із поширених рішень цієї проблеми є створення нового типу вузла, який є вузлом послідовності з пам'яттю[1] від того, яка дитина активна, і пропустити дитину, яка вже повернула Успіх у якийсь більш ранній момент часу. Однак, як зазначено в [81] того самого результату можна досягти, додавши вузол декоратора над кожною дією

[1] іноді називається Послідовність *

маршрутної точки, яка продовжує повертати успіх без відмітки її дочірнього елемента після повернення початкового успіху.

Резонне запитання полягає в тому, в яких частинах простору станів ДП повертає успіх чи невдачу. Ця проблема була розглянута для телереактивних конструкцій у [80] і передано до ДП в [83]. Для того, щоб зробити формальний аналіз простору станів, функціональна модель була запропонована в [83]. Використовуючи інструменти теорії керування, такі як область тяжіння та експоненційна стабільність, розглядалися такі властивості, як надійність (області притягання), безпека (уникнення деяких регіонів) та ефективність (збіжність у верхніх часових межах). Показуючи, як властивості переносяться на послідовність і резервні композиції, аналіз можна зробити для більших ДП.

Аналіз надійності, безпеки та ефективності був продовжений у [84], де було показано, як можна додати піддерево навчання з, можливо, ненадійною продуктивністю, в той час як деякі гарантії надійності, безпеки та ефективності все ще можуть бути надані для загального ДП.

В [85], концепція, дуже схожа на ДП, була введена у вигляді надійних логічних динамічних систем. Доведено збіжність послідовностей таких компонентів разом із стохастичним аналізом часів збіжності при обмеженій кількості переходів зовнішніх станів [86] розширити основне позначення ДП, додавши попередні та постумови до вузлів Action і таким чином видаливши вузли Condition. Ця модифікована модель називається розширеним деревом поведінки (еДП) і призначена для взаємодії з плануванням ієрархічної мережі завдань (HTN). Цей алгоритм планування також відповідає за перебудову дерева, оптимізацію часу виконання та використовуваних ресурсів, але без шкоди для коректності дерева. Автори також констатують відмінності їх моделі від стандартної. Ця структура також використовується в [87], де він поєднується з генератором руху (MG), відповідальним за накладання окремих і незалежних примітивів руху для створення гібридного руху. Ця настройка дозволяє одночасно

активувати різні примітиви за допомогою оператора Parallel, таким чином перетинаючи межі, зазначені також у [88].

Більшість дій ДП вимагають даних про світ, наприклад розташування об'єктів та інших агентів, або кількість заряду, що залишився в акумуляторі [89] запровадити параметризацію в деревах поведінки. Замість використання дошки, тобто сховища даних, доступних для всіх вузлів дерева, як у [90], вони пропонують додати параметри до вузла так, щоб вузол зберігав усі параметри, необхідні для виконання свого піддерева. Це робиться трьома різними способами: його можна жорстко закодувати, взяти з характеристик світу та агентів або задовольнити аргументами параметризованого представлення дій (PAR), які дозволяють повторно використовувати та інкапсуляцію піддерева.

ДП були вперше створені для діалогового ШІ у [69], і з тих пір стали стандартним інструментом для розробки ШІ неігрових персонажів у багатьох ігрових жанрах, від стратегічних ігор до шутерів.

Стратегічні ігри в режимі реального часу вимагають від гравців прийняття тактичних рішень, наприклад, взяти під контроль певну задану територію або здійснити атаку на певні об'єкти противника. У таких іграх існує велика кількість різноманітних юнітів, які можна призначити для виконання конкретних завдань, виходячи з їхніх сильних і слабких сторін. Ці підрозділи також можна об'єднати в загони, взводи і так далі аж до цілої армії, а ШІ використовувалися для прийняття рішень на багатьох таких рівнях абстракції. Попередня стаття про ШІ RTS [91], де використовувалися еволюційні методи для створення ШІ, які могли б перемогти штучний інтелект із ручним кодуванням для гри DEFCON в реальному часі. Інші приклади еволюційних методів створення ШІ включають [92], де був розроблений ШІ для покрокових стратегічних ігор, автори створили ДП на основі повторюваних дій, які виконуються людьми-експертами. Запропонований підхід не потребував жодної моделі дій чи винагород, але вимагав значної кількості тренувальних слідів. Альтернативний підхід із використанням знань, введених вручну, був запропонований у [93]. Однією з найвідоміших і найскладніших ігор

у реальному часі є StarCraft. в [94], StarCraft використовувався для демонстрації здатності мови поведінки (ABL) здійснювати реактивне планування, а агент на рівні людини був запропонований в [95].

Одна з перших статей про ШІ описує, як вони використовуються для створення штучного інтелекту дуже популярної FPS гри Halo 2. У іграх FPS гравець взаємодіє з великою кількістю неігрових персонажів у дуже динамічному бойовому середовищі, і приклад прийнятих рішень і дій можна побачити на малюнку2.

Проблема повторного використання ДП в ситуаціях, коли доступні дії агента з часом збільшуються, досліджувалася в [96]. Запропоновано підхід, у якому дії розподіляються за категоріями, при якому листи ДП роблять запити високого рівня, щоб знайти дію, що підходить для даної ситуації. Потім ця робота продовжена в [97], де над ДП додано вищий шар абстракції. Маючи набір ДП для досягнення різних цілей, для вибору відповідного ДП для використання використовується аргументація на основі випадків.

До FPS ДП було внесено додаткові модифікації, включаючи час, сприйняття ризику та емоції, або шляхом вивчення ДП зі слідів, отриманих під час гри людини (програмування шляхом демонстрації) [98].

Платформні ігри — це ігри, де гравець керує персонажем у 2D-середовищі та переміщується між набором платформ [Б80]. Фреймворк Mario ШІ імітує популярну платформену гру і став широко використовуваним інструментом для перевірки еволюційних підходів до вивчення ДП [99]. Pac-Man технічно не є платформною грою, але має 2D-структуру і використовувалася для тестування пошуку дерева Монте-Карло за допомогою ДП [100], гібридні ДП, що розвиваються, та інструменти, зосереджені на підтримці ручного проектування ДП.

Прагнучи зробити ігрове середовище більш реалістичним, дизайнери іноді дозволяють НІП взаємодіяти з гравцем не лише за допомогою дій, а й за допомогою мовленнєвих діалогів. Це була мотиваційна заявка на концепцію ДП у

[69], і пізніше було досліджено в [93], в [78] розглядаються такі ситуації, як розмова офіціанта під час отримання замовлення, а [82] досліджує сценарій взаємодії людини і робота. Інтерактивні наративи досліджуються в [99], де два віртуальних плюшевих ведмедика просять гравця знайти їм м'яч для гри, перебуваючи в [93], досліджуються типи розмов, такі як переговори покупця та продавця та просте запитання-відповідь.

3.2 Моделювання

В основі системи моделювання поведінки знаходиться дерево поведінки (див. рис. 3.1).

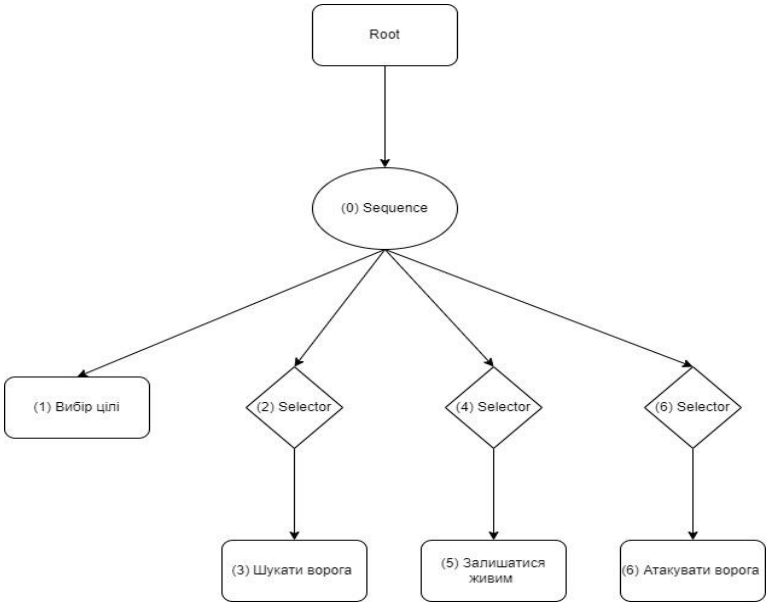

Рис. 3.1. Схема дерева поведінки.

Система передбачає три основні цілі:

1) Шукати ворога;
2) Залишатися живим;
3) Атакувати ворога.

Вибір найбільш вагомої цілі проводиться кожен тік, на основі результатів алгоритму нечіткого виводу Мамдані (див. рис. 3.2). Результатом виконання алгоритму є три кількісних значення трьох лінгвістичних змінних, кожна з яких описує рівень доцільності однієї з цілей, усі значення знаходяться у межах [0, 1].

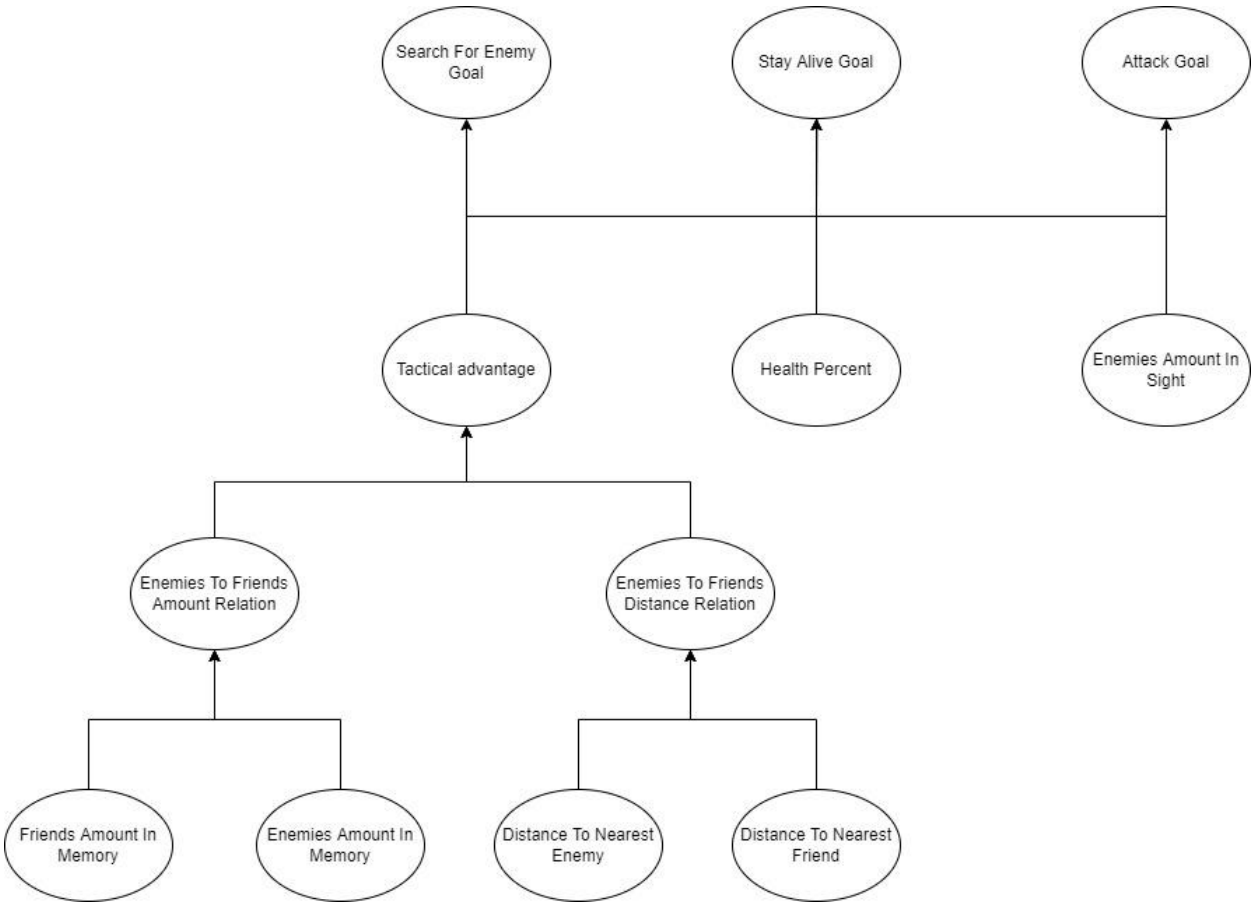

Рис. 3.2. Вплив лінгвістичних змінних у базі правил.

Кожна з лінгвістичних змінних, що використовуються у базі правил має 3 лінгвістичних терми. Більша кількість призведе до збільшення кількості правил, що призведе до зменшення продуктивності. Менша кількість правил призведе гірших розрахунків.

Для приведення змінних до нечіткості використовується трикутна функція залежності.

$$trimf(x,a,b,c) = \begin{cases} 0, & x \le a \\ \frac{x-a}{b-a}, & a \le x \le b \\ \frac{c-x}{c-b}, & b \le x \le c \\ 0, & c \le x \end{cases}, \quad (3.1)$$

У базі правил використовується дванадцять лінгвістичних змінних, з них вхідними є:

1) Кількість друзів у пам'яті (Friends Amount In Memory) (див. рис. 3.3);

2) Кількість ворогів у пам'яті (Enemies Amount In Memory) (див. рис. 3.4);
3) Дистанція до найближчого ворога (Distance To Nearest Enemy) (див. рис. 3.5);
4) Дистанція до найближчого друга (Distance To Nearest Friend) (див. рис. 3.6);
5) Процент здоров'я (Health Percent) (див. рис. 3.7);
6) Кількість ворогів у полі зору (Enemies Amount In Sight) (див. рис. 3.8).

Проміжними лінгвістичними змінними є:

1) Відношення кількість ворогів до друзів (Enemies To Friends Amount Relation) (див. рис. 3.9);
2) Відношення відставні до ворогів і до друзів (Enemies To Friends Distance Relation) (див. рис. 3.10);
3) Тактичні переваги (Tactical advantage) (див. рис. 3.11).

Вихідними лінгвістичними змінними є:

1) Ціль пошуку ворога (Search For Enemy Goal) (див. рис. 3.12);
2) Ціль залишатися живим (Stay Alive Goal) (див. рис. 3.13);
3) Ціль атакувати ворога (Attack Goal) (див. рис. 3.14).

Після вибору найбільш вагомої цілі, дерево поведінки (див. рис. 3.1) обирає один з селекторів та переходить до виконання саме тієї послідовності дій, що відповідає обраному селектору. Після виконання усієї послідовності, дерево починає виконуватися спочатку, саме через це так важливо мінімізувати кількість правил, оскільки активних персонажів може нараховуватися декілька десятків, або навіть сотень. Також потрібно брату у увагу, шо ми не можемо виділити усі ресурси на обчислення лише неігрових персонажів, оскільки під час гри може відбуватися велика кількість обчислень, що не стосуються ігрового ШІ.

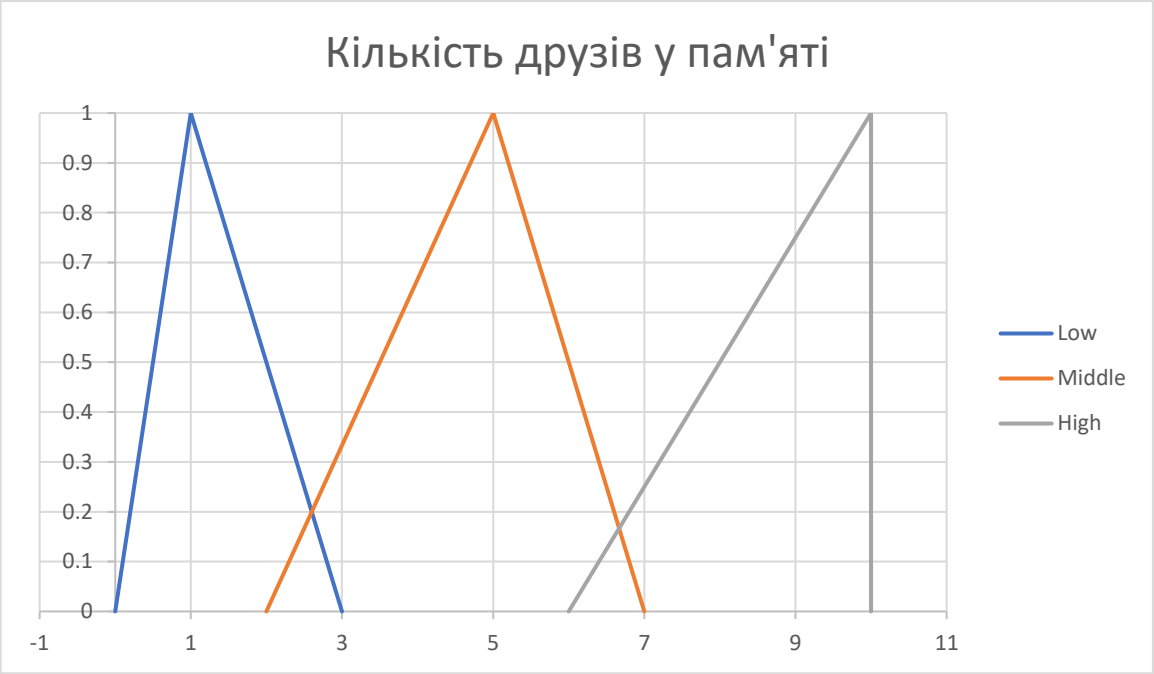

Рис. 3.3. Лінгвістична змінна «Кількість друзів у пам’яті»

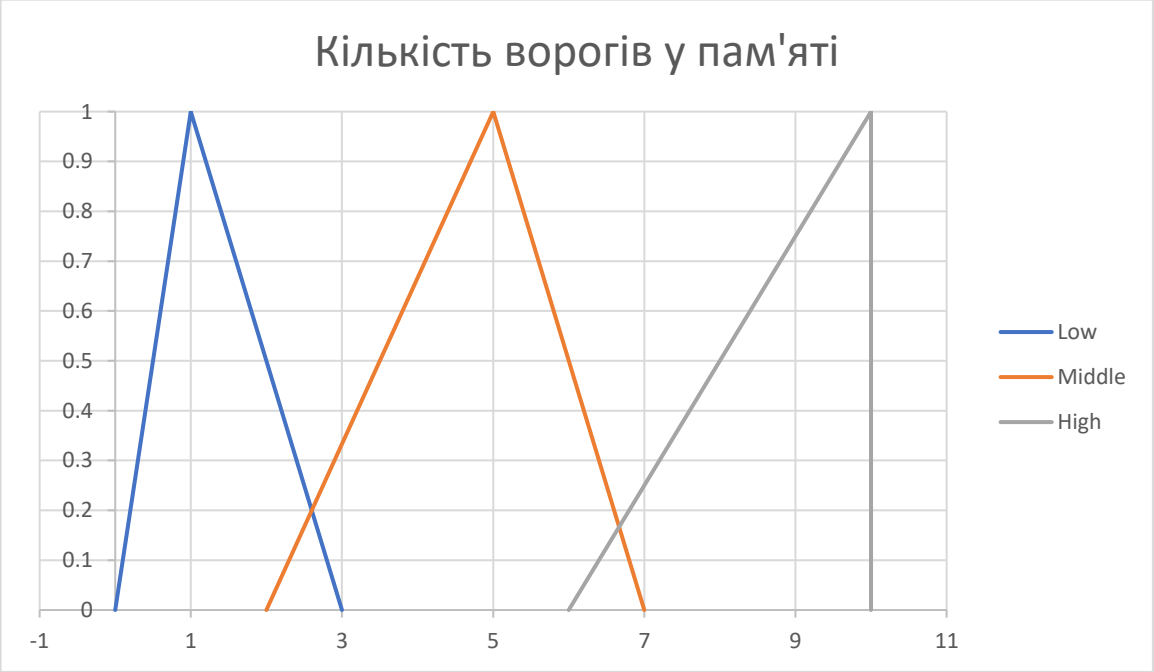

Рис. 3.4. Лінгвістична змінна «Кількість ворогів у пам’яті»

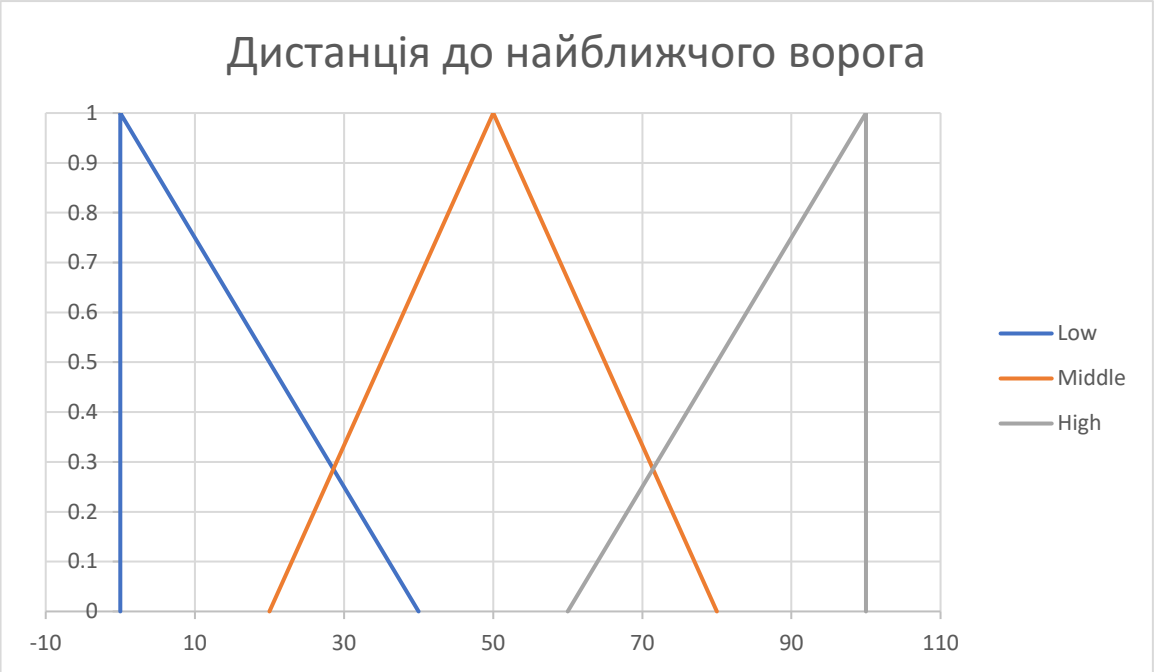

Рис. 3.5. Лінгвістична змінна «Дистанція до найближчого ворога»

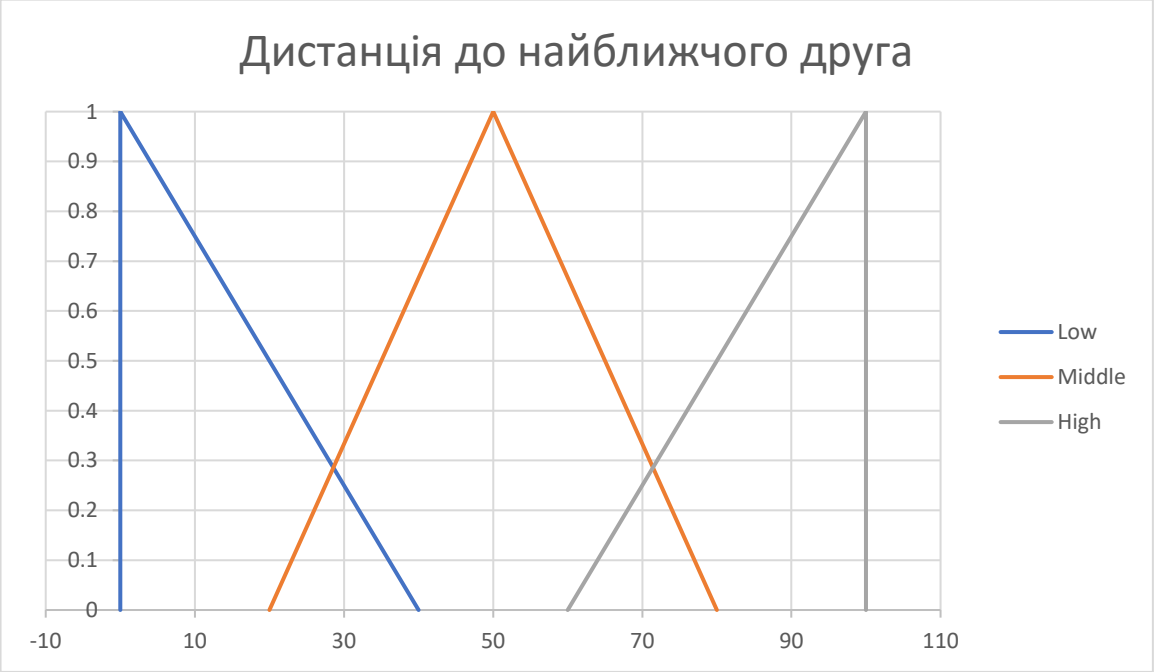

Рис. 3.6. Лінгвістична змінна «Дистанція до найближчого друга»

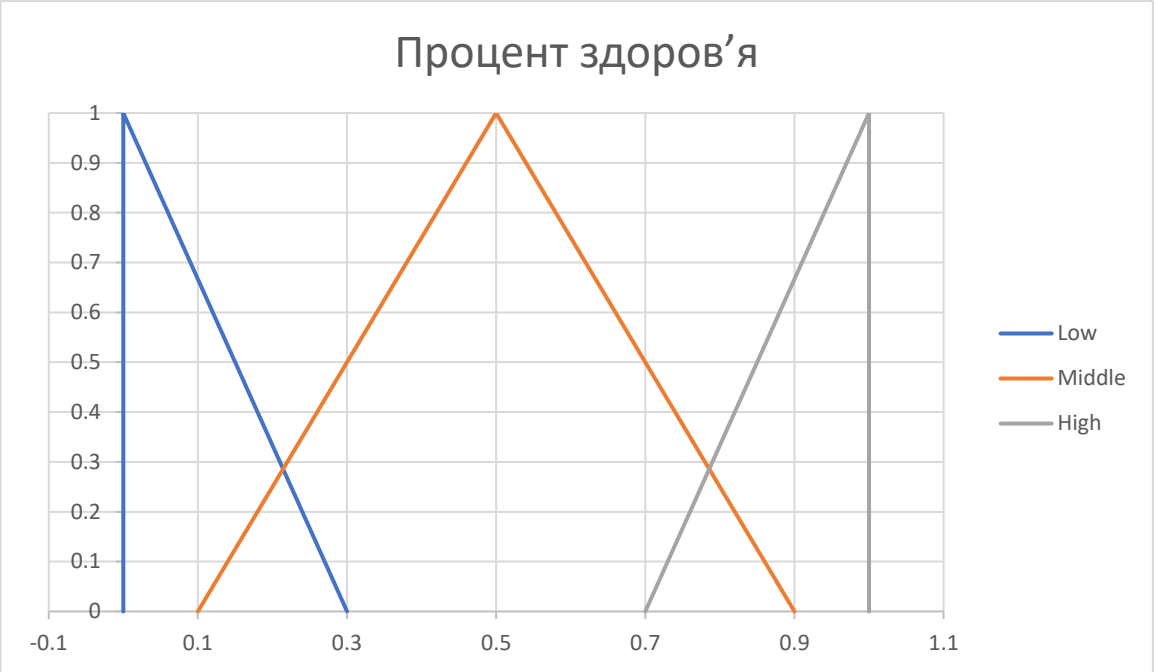

Рис. 3.7. Лінгвістична змінна «Процент здоров'я»

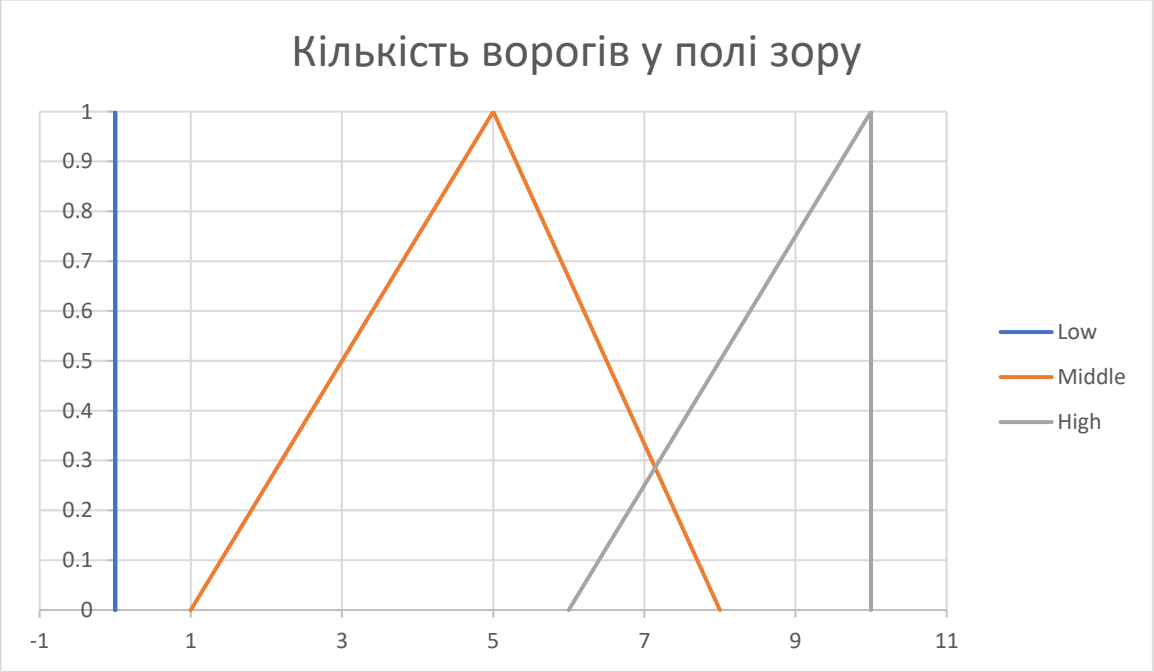

Рис. 3.8. Лінгвістична змінна «Кількість ворогів у полі зору»

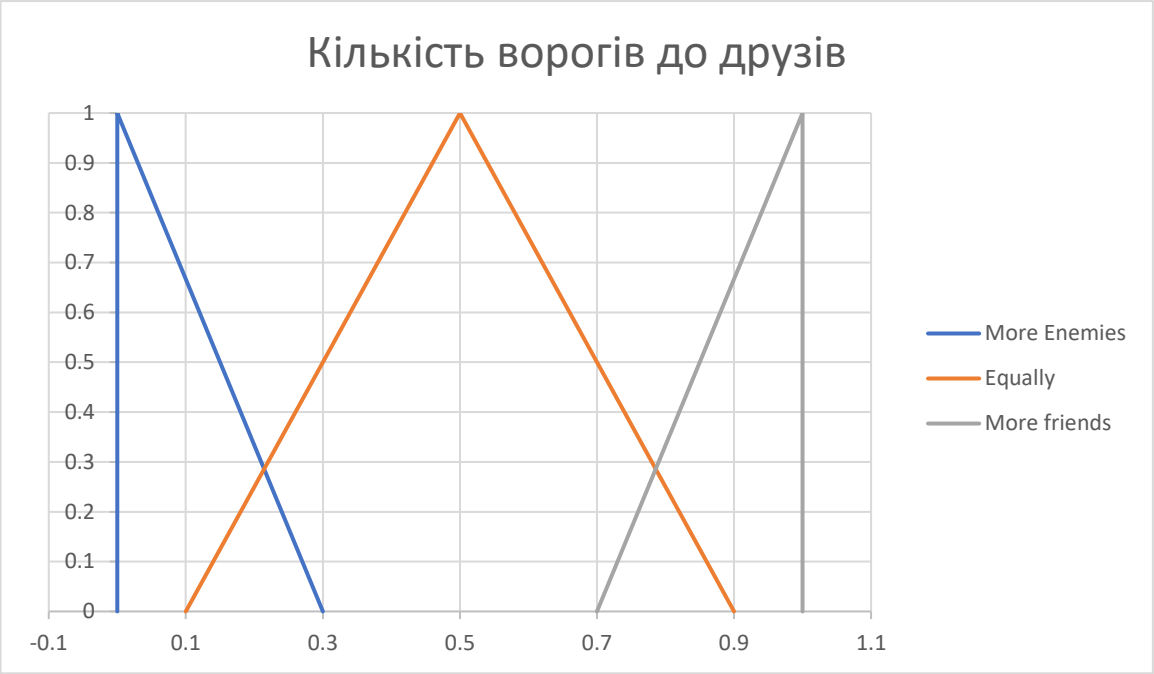

Рис. 3.9. Лінгвістична змінна «Відношення кількість ворогів до друзів»

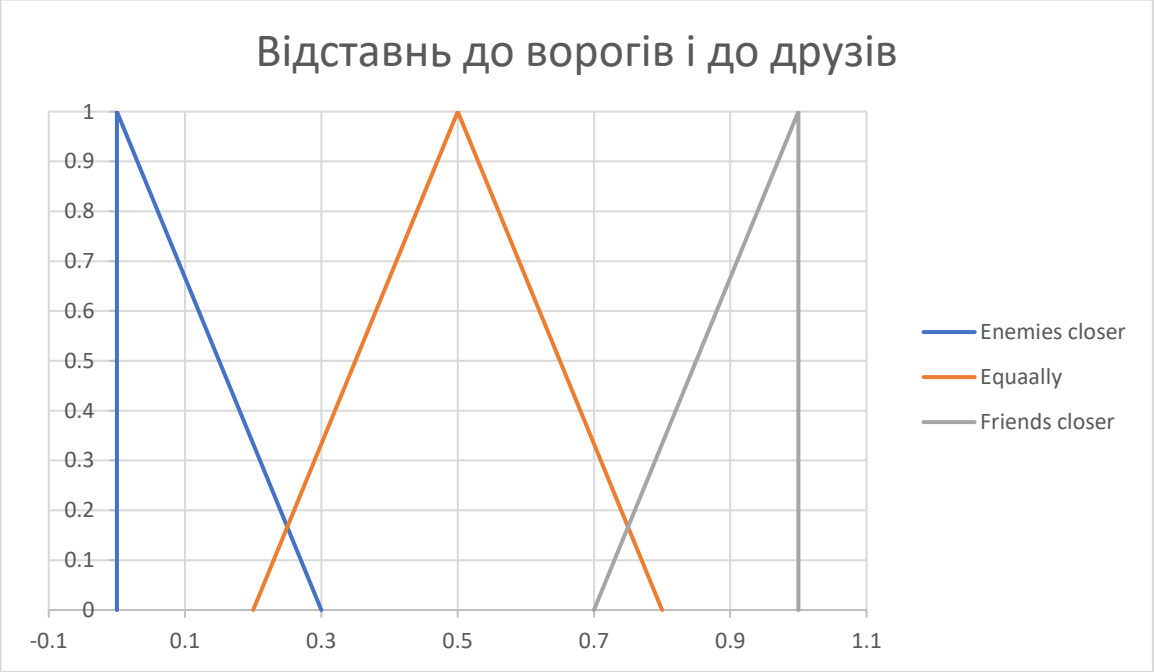

Рис. 3.10. Лінгвістична змінна «Відношення відставні до ворогів і до друзів»

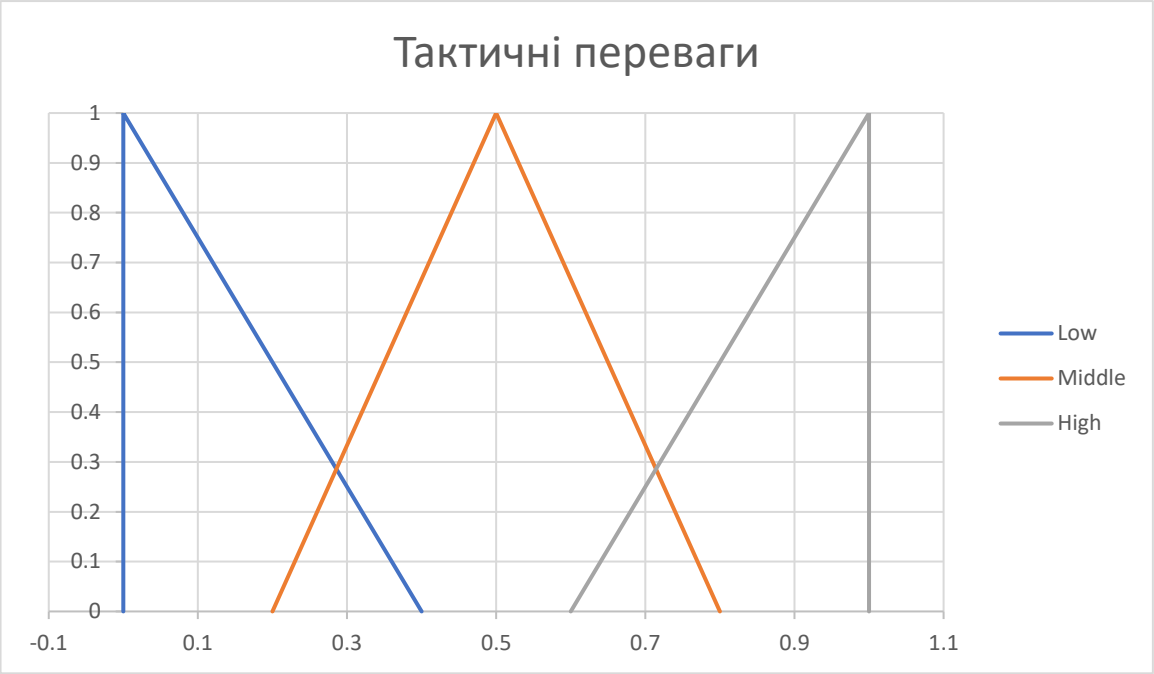

Рис. 3.11. Лінгвістична змінна «Тактичні переваги»

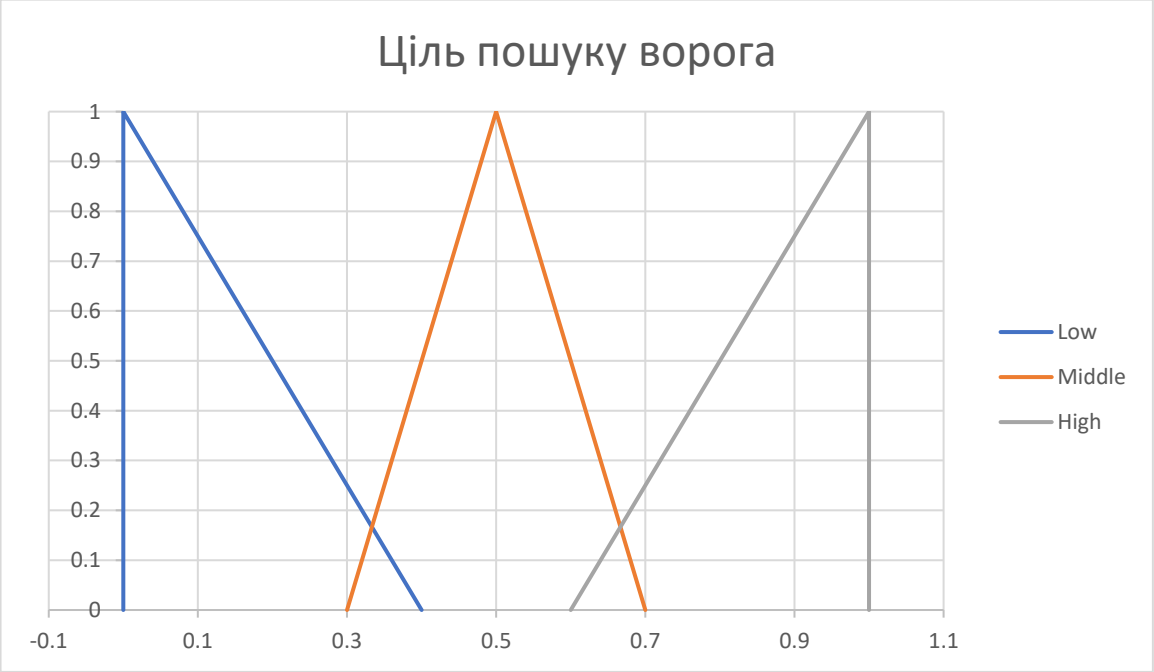

Рис. 3.12. Лінгвістична змінна «Ціль пошуку ворога»

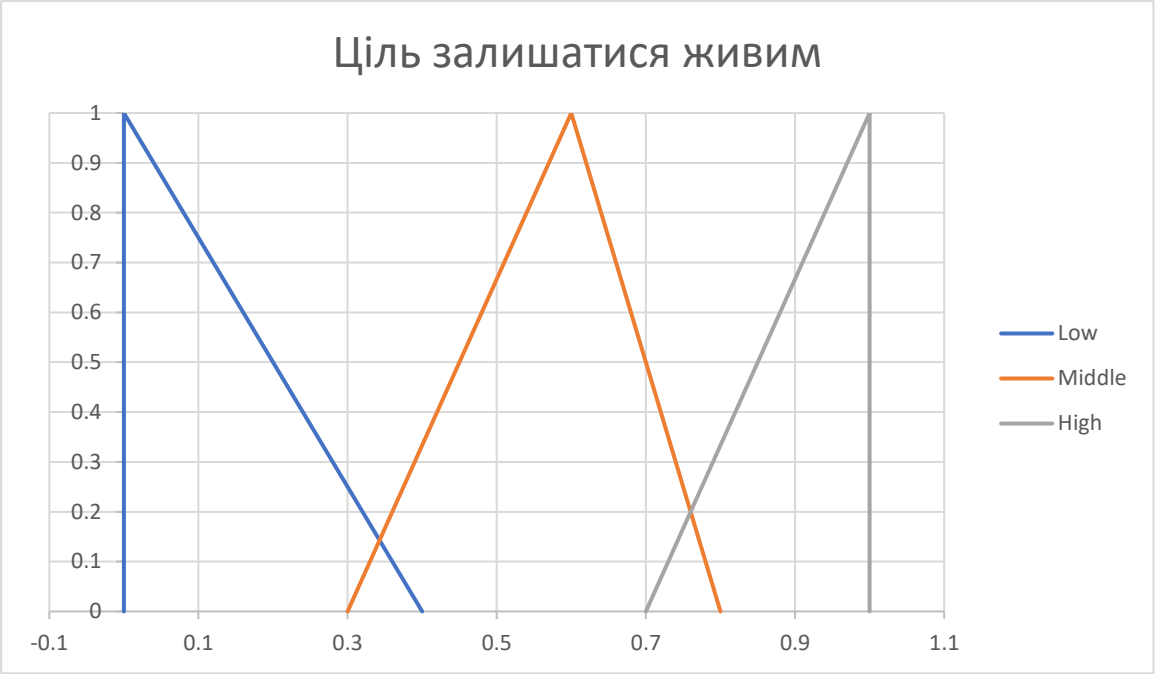

Рис. 3.13. Лінгвістична змінна «Ціль залишатися живим»

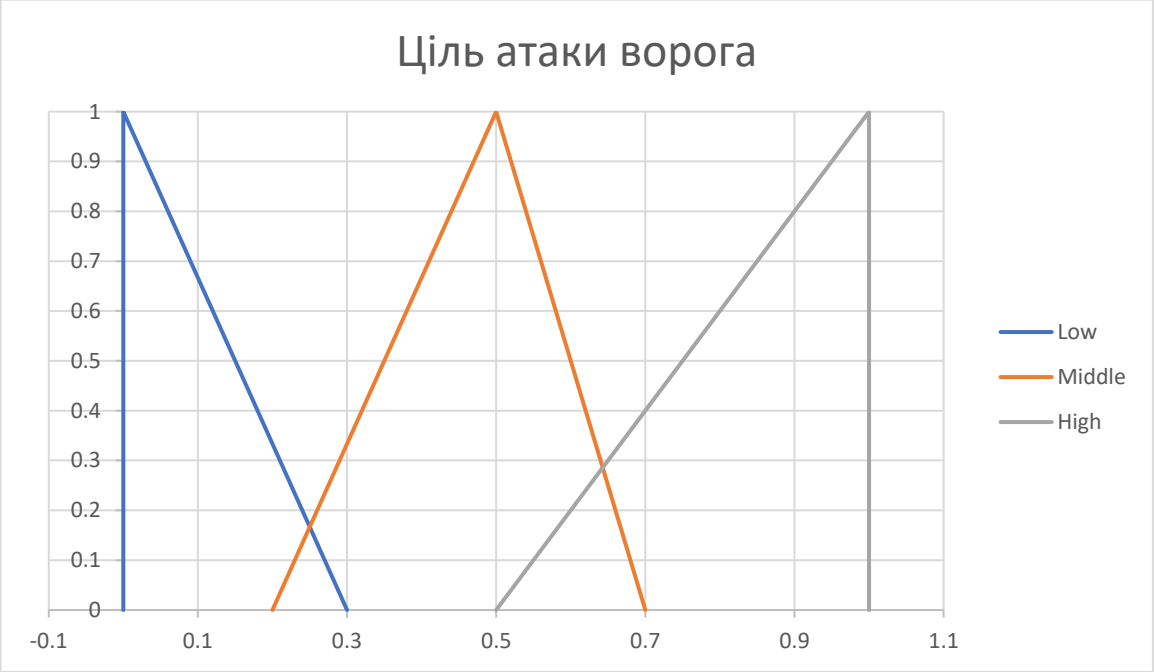

Рис. 3.14. Лінгвістична змінна «Ціль атакувати ворога»

Висновки до розділу 3

У третьому розділі був розглянутий алгоритм нечіткого виведення Мамдані. Алгоритм примітний тим, що він працює за принципом «чорної скриньки». На вхід надходять кількісні значення, виході також вони. На проміжних етапах

використовується апарат нечіткої логіки та теорія нечітких множин. У цьому полягає елегантність використання нечітких систем. Можна маніпулювати звичними числовими даними, але використовувати гнучкі можливості, які надають системи нечіткого виведення.

Дерева поведінки надають можливість створювати гнучку модель поведінки, що зручно масштабувати і змінювати. Завдяки своїм якостям дерева поведінки є ідеальним рішенням для моделювання поведінки неігрових персонажів в комп’ютерних іграх. Простота використання та швидкість виконання якісно відрізняють цей підхід, та роблять його найкращим варіантом для використання під час моделювання ігрового ШІ.

Поєднання цих двох методів дозволяють розробити інтелектуальну систему, що буде зручна в використанні, та мати переваги використання нечіткої логіки. Цей метод дозволе швидко, що дуже важливо під час комерційної розробки, створювати швидкі у виконанні інтелектуальні системи наближені до поведінки людей. Однак слід пам’ятати, щоб досягти бажаного ефекту потрібно ретельно налаштувати базу правил, та обрати лінгвістичні змінні, у протилежному випадку система може не надати бажаного ефекту, чи виявитися занадто ресурсозатратним. Останнє неприпустимо, особливо у сфері відеоігор.

Розроблена модель використовую сильні сторони алгоритму Мамдані та дерев поведінки. Система передбачає три основні цілі персонажу: атака, захист життя та пошук ворогів. На підставі таких показників як кількість друзів у пам’яті, кількість ворогів у пам’яті, дистанція до найближчого ворога, дистанція до найближчого друга, процент здоров’я та кількість ворогів у полі зору система робить вибір однієї з трьох стратегій, після чого виконується логіка дерева поведінки.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ

4.1 Інструментальні засоби для створення інтелектуальної системи моделювання поведінки персонажів

У наш час існує багато ігрових рушіїв, але більшість із них створюються для використання у однією студією чи видавництвом. Однак існують декілька ігрових рушіїв доступних усім бажаючим без обмежень, серед них виділяються два найпопулярніші, Unreal Engine та Unity.

4.1.1 Ігровий рушій Unreal Engine 4

Unreal Engine (наразі випущений як Unreal Engine 4) — популярний і широко використовуваний ігровий рушій, розроблений Epic Games. Він використовується в багатьох сучасних іграх ААА рівня, як-от власний шутер Battle Royale Fortnite від Epic або інших хітових іграх, як-от Psyonix «Rocket League».

Він дозволяє розробляти на кількох платформах від ПК до консолей, таких як PS4, Xbox One та Nintendo Switch. Це є однією з причин, чому він настільки широко використовується, завдяки його гнучкості для роботи між цими різними платформами.

Більш досвідчені програмісти можуть використовувати мову C++ для створення власних скриптів, які запускаються в ігровому движку. Більше любителів розробників можуть використовувати його візуальне представлення blueprint, які в основному є готовими блоками коду, які ви можете додати до своїх об'єктів для взаємодії.

Він також має потужний матеріал та інструменти анімації для художників, які дозволяють швидко створювати складні сцени. Налаштування деяких з цих функцій спочатку може здатися складним, але наведено кілька прикладів, де ви можете просто змінити параметри, поки не створите те, що шукаєте.

Unreal Engine в певній формі існує з 1998 року. Вперше він був використаний для гри «Unreal», коли Unreal Engine 2 вийшов у 2002 році, Unreal Engine 3 вийшов у 2006 році, а поточна версія UE 4 була випущена в 2014 році.

Завдяки довгому стажу роботи в ігровій індустрії, він має величезну кількість підписників і безліч сторонніх навчальних посібників та онлайн-форумів.

4.1.2 Blueprint

Система Visual Scripting Blueprint в Unreal Engine — це повна система сценаріїв ігрового процесу, заснована на концепції використання інтерфейсу на основі вузлів для створення елементів ігрового процесу з Unreal Editor. Як і в багатьох поширених мовах сценаріїв, вона використовується для визначення об'єктно-орієнтованих класів або об'єктів у механізмі.

Ця система надзвичайно гнучка та потужна, оскільки надає можливість дизайнерам використовувати практично весь спектр концепцій та інструментів, які зазвичай доступні лише програмістам. Крім того, спеціальна розмітка Blueprint, доступна в реалізації Unreal Engine на C++, дозволяє програмістам створювати базові системи, які можуть бути розширені дизайнерами.

4.2 Реалізація алгоритму Мамдані

Оскільки для UE4 не існує готової реалізації алгоритму Мамдані, було прийнято рішення створити власну реалізацію, використовуючи Blueprint. Для цього була створена окрема бібліотека FuzzyLib (табл. 4.1).

Таблиця 4.1

Опис функцій бібліотеки FuzzyLib

Назва Функції	Опис
Trimf	Для заданої області *x* і параметрів параметрів повертає відповідні значення *y* для трикутної функції належності.
Fuzzification	Метою цієї функції є отримання значень істинності для всіх умов із бази правил.
Aggregation	Функція визначає ступені істинності умов для кожного правила системи нечіткого висновку.
Activation	На цьому етапі відбувається перехід від умов до висновків.
GetActivatedValue	Функція надає активоване значення.
Accumulation	Метою цієї функції є отримання нечіткої множини (або їх об'єднання) для кожної з вихідних змінних.
getMaxValueOfUnionFuzzySet	Функція повертає максимальне значення об'єднання нечітких множин.
Defuzzification	Мета дефазифікації одержати кількісне значення для кожної з вихідних лінгвістичних змінних.
Integral	Ця функція розраховує інтеграл об'єднання нечітких множин.

Для спрощення обробки даних були створенні додаткові структури даних, для роботи з нечіткою логікою (табл. 4.2).

Таблиця 4.2

Додаткові структури даних

Назва Структури	Опис
FuzzySet	Структура що складається з трьох змінних з плаваючою комою. Створена для зберігання нечіткої множини.
FuzzyStatement	Включає в себе FuzzySet та строку. Створена для зберігання терму.
ActivatedFuzzySet	Активована нечітка множина. Вмістить FuzzySet та коефіцієнт істинності.
Rule	Описує правило. Включає два масиви FuzzyStatement. Перший відповідає за умови, другий – за висновки.

Реалізація алгоритму Мамдані передбачує послідовне виконання функцій Fuzzification, Aggregation, Activation, Accumulation та Defuzzification. Перша функція приймає базу правил та вхідні змінні, після цього кожна наступна функція у послідовності приймає результат виконання попередньої. Результатом виконання алгоритму є кількісне значення у проміжку [0, 1].

4.3 Дерево поведінки

В UE4 вже передбачено інструмент створення та розширення дерев поведінки. Створення штучного інтелекту в UE4 для персонажів можна зробити кількома різними способами. Можна використовувати візуальні сценарії Blueprint,

щоб доручити персонажу «зробити щось», наприклад відтворити анімацію, переміститися в певне місце, реагувати на удар тощо. Якщо потрібно, щоб персонажі зі штучним інтелектом самі думали та приймали власні рішення, дерева поведінки можуть допомогти! Нижче наведено приклад дерева поведінки, де персонаж ШІ перемикається між патрулюванням і переслідуванням гравця.

Дерева поведінки створюються візуально, подібно до Blueprint, шляхом додавання та підключення ряду вузлів, які мають певну функціональність, додану до них, до графіка дерева поведінки. У той час як дерево поведінки виконує логіку, окремий ресурс, який називається Blackboard, використовується для зберігання інформації (так звана Blackboard Keys), про яку дерево поведінки має знати, щоб приймати зважені рішення. Типовим робочим процесом буде створення Blackboard, додавання деяких ключів Blackboard, а потім створення дерева поведінки, яке використовує актив Blackboard.

Дерева поведінки в UE4 виконують свою логіку зліва направо та зверху вниз. Числовий порядок роботи можна переглянути у верхньому правому куті вузлів, розміщених на графіку. На зображенні нижче зразок гілки (див. рис. 4.1).

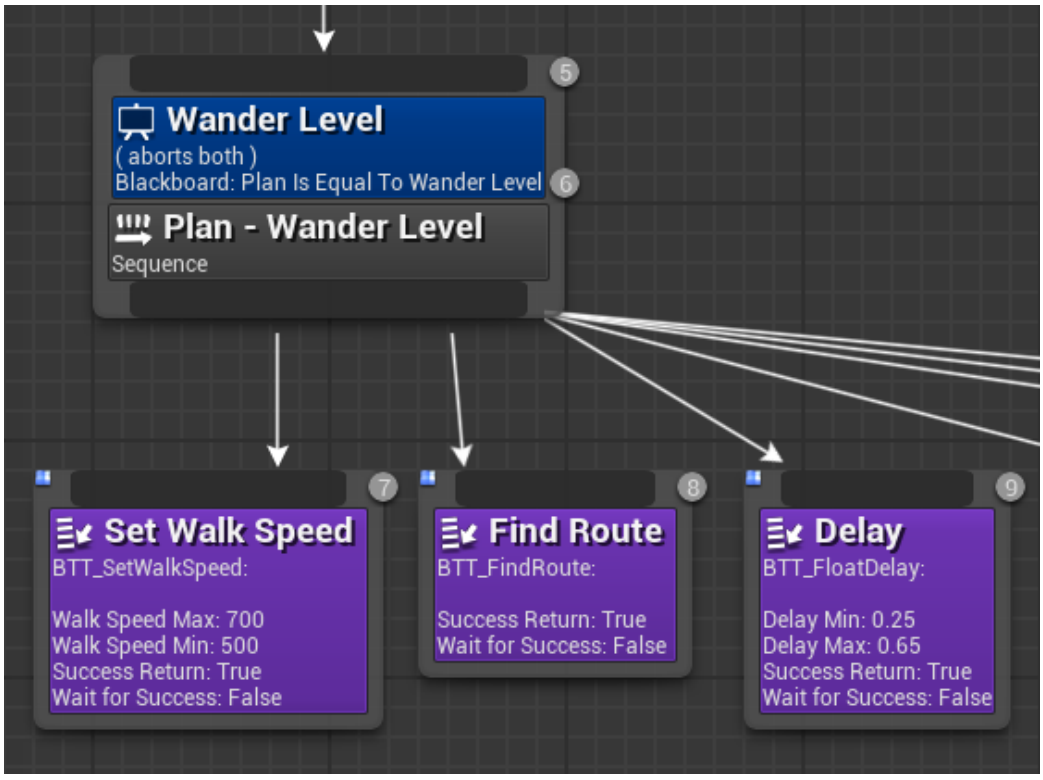

Рис. 4.1. Приклад дерева поведінки у UE4

Розроблене дерево поведінки призначене для керування персонажем – бійцем ближнього бою. Поведінка подібного персонажу складається з трьох основних цілей: пошуку ворога, атаки ворога та спроби залишитися живим рятуючись втечею.

Виконання дерева починається з кореня, після чого переходить вибору логіки поведінки згідно до вибраної цілі (див. рис. 4.2). Деякі цілі передбачають різні стратегії поведінки, вибір стратегії спирається на чіткі данні, оскільки для більшості з них не є доцільним приведення до нечіткості. Наприклад знаходження у стані бою може бути описано лише двома значеннями.

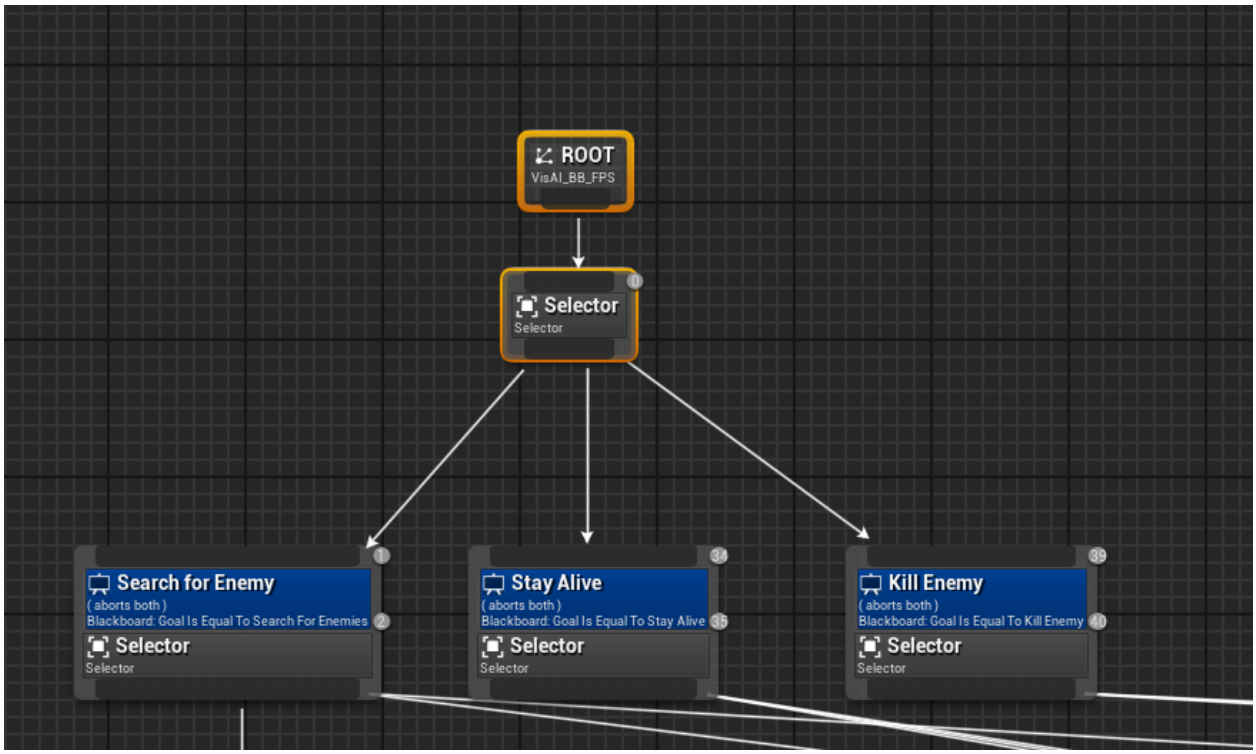

Рис. 4.2. Вибір поведінки згідно до обраної цілі

Ціль пошуку ворога передбачає кілько стратегій поведінки. Перша стратегія, це обхід рівня у пошуку ворога. Ця стратегія використовується, коли агент немає жодної інформації про ворогів. Спершу персонаж знаходить точку патрулювання та переходить до неї (див. рис. 4.3), потім проводить операцію пошуку ворогу.

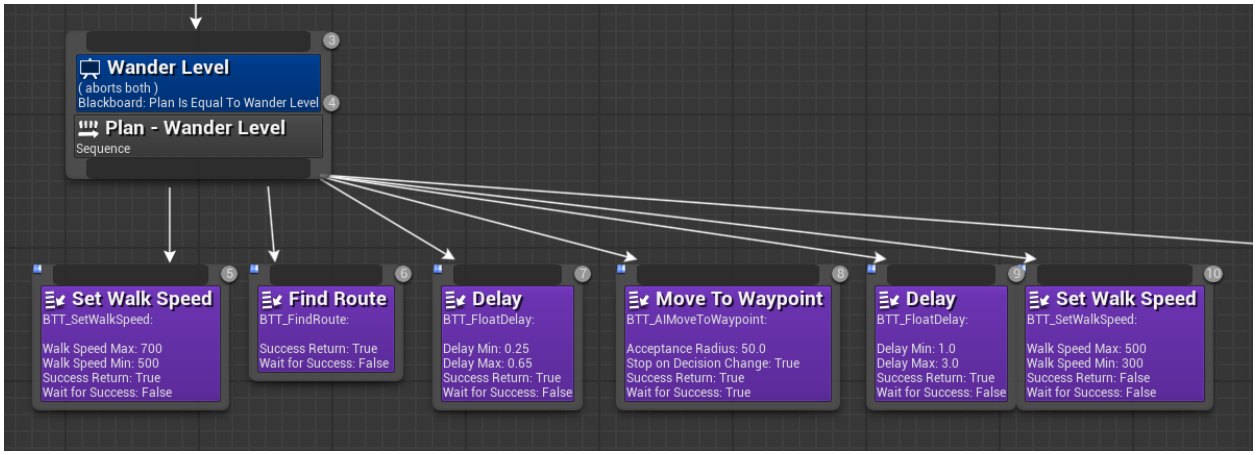

Рис. 4.3. Перехід до точки патрулювання

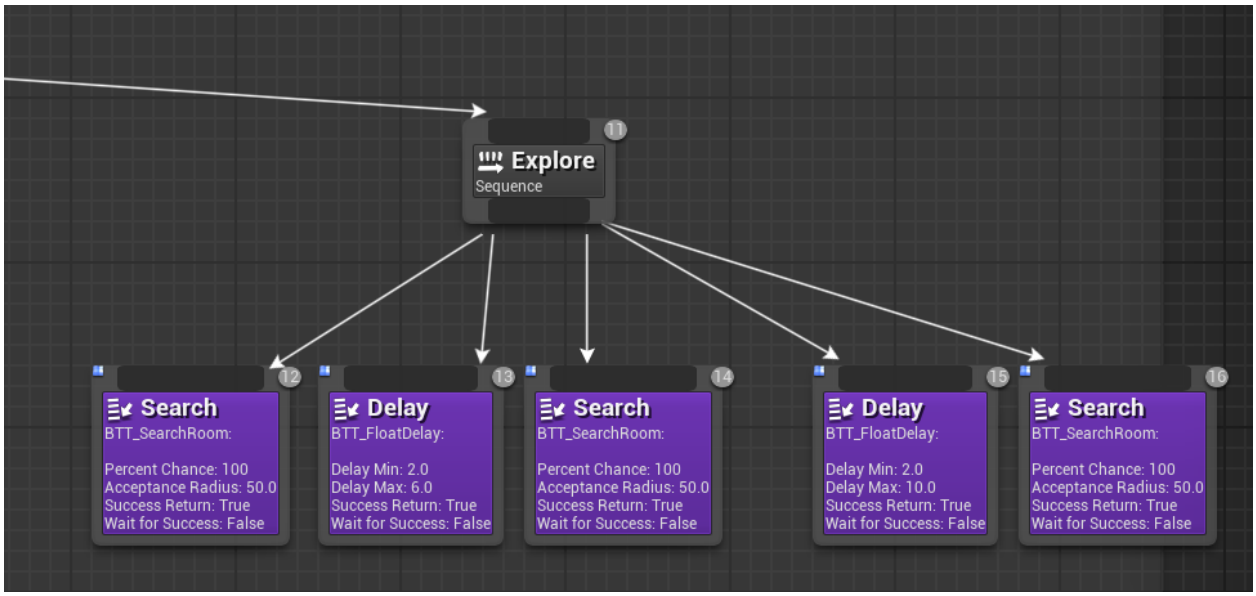

Рис. 4.4. Операція пошуку ворога.

Наступна стратегія пошуку, це пошук останнього ворога. Ця стратегія може застосовуватися, якщо агент знає, що залишився лише один ворог, або бій закінчився (див. рис. 4.5). Персонаж буде по черзі обходити усі кімнати у приміщені у пошуку останнього ворогу, при цьому у вже перевірені кімнати він не повернеться, поки не обійде усі неперевірені.

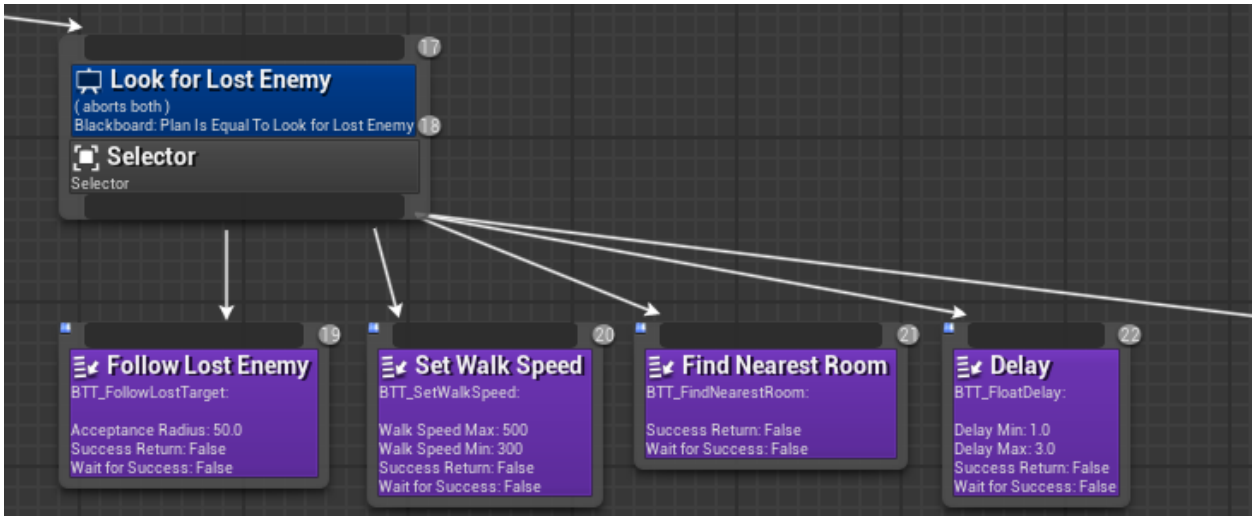

Рис. 4.5. Пошук останнього ворога

Остання стратегія пошуку, це стратегія розслідування звуку. Оскільки агенти можуть орієнтуватися не тільки на зір, але й на слух, є можливість пошуку ворога по звуку. Ця стратегія доцільна, якщо ворог не знаходиться у полі зору, але веде себе гучно.

Ще однією ціллю, є ціль врятувати собі життя. Логіка виконання цією ціль дуже проста, агент намагається вийти з бою, шляхом пошуку найближчого безпечного місця.

Останньою ціллю є атака ворога, ця ціль є пріоритетною, коли агент бачить ворога, і має досить здоров'я, щоб вступати у бій. Ця ціль також має декілька стратегій.

Перша стратегія, це стратегія зближення з ворогом (див. рис. 4.6). Оскільки персонаж спеціалізується на ближньому бою, він повинен проводити атаки з коротких дистанцій.

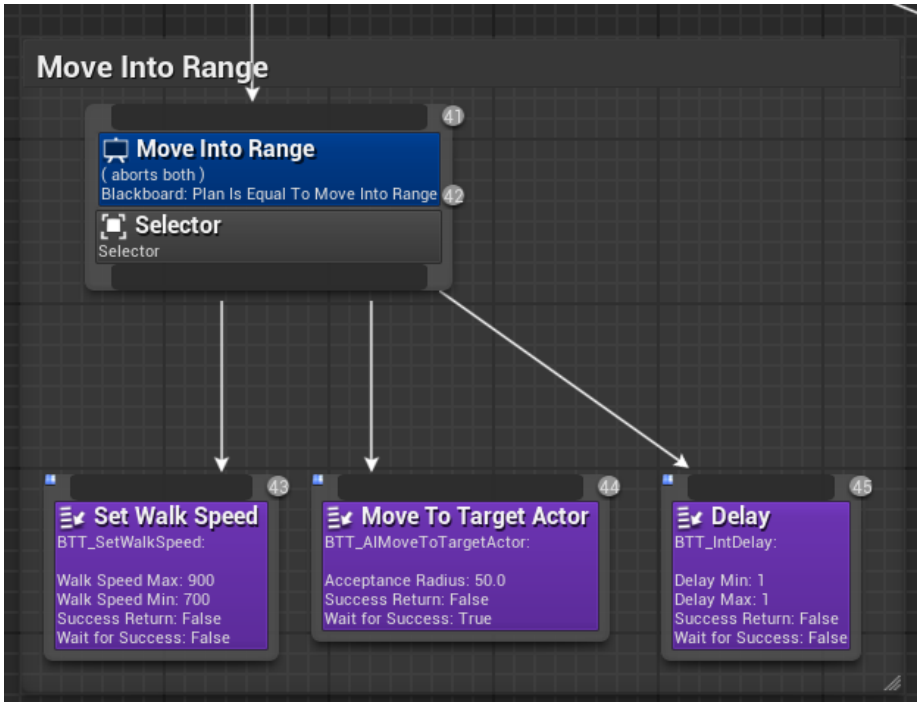

Рис. 4.6. Скорочення дистанції з ворогом

Другою стратегією є атака ворога (див. рис. 4.7), вона використовується, якщо дистанція між ворогом та персонажем дозволяю проводити рукопашні атаки.

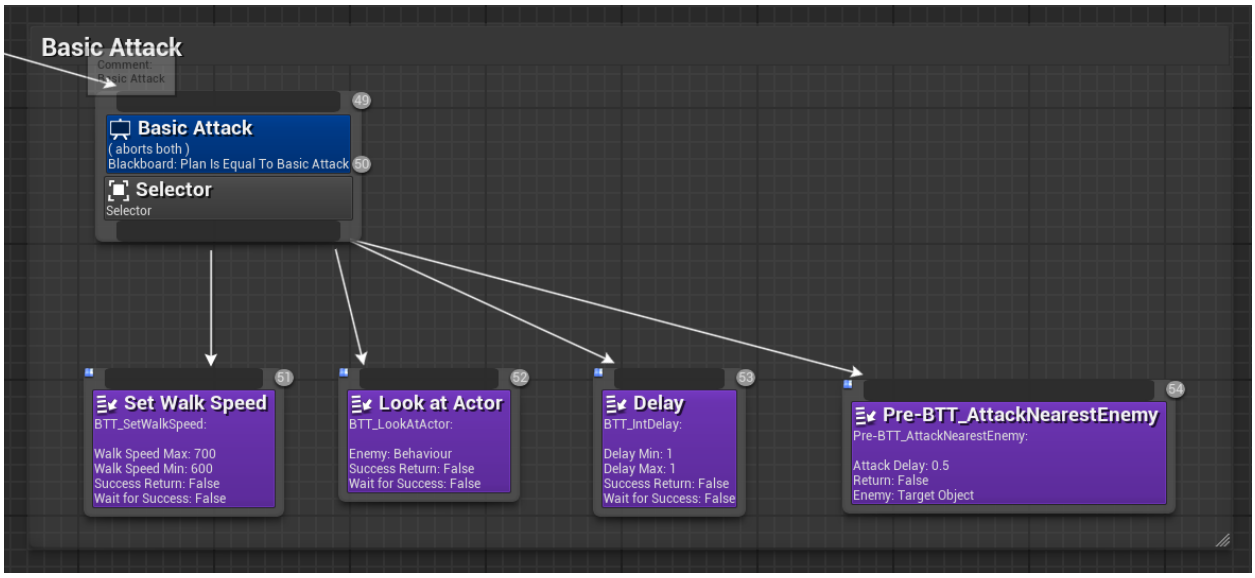

Рис. 4.7. Атака ворога

Для початку зменшується швидкість переміщення персонажу, це робиться для створення відчуття близького контакту. Другим кроком є фокус на

супротивнику, поворот усього корпусу до нього. Завдяки цьому створюється вигляд постійного контролю опонента, та ілюзія постійного переміщення навколо цілі. Наступним кроком є затримка на одну секунду, щоб не давати можливості персонажу наносити постійні удари, не даючи шансу на відповідь. Останнім кроком є безпосередньо удар по опоненту, після чого гілка атаки завершується.

4.4 Персонаж

Для використання у проекті був створений персонаж (див. рис. 4.8). Персонаж має людиноподібні форми та розміри. На зображенні можна побачити декілька сфер та циліндрів. Навколо персонажу можна побачити дві капсули. Менша капсула відповідає за сприйняття доторків, по суті вона симулює дотик. Більша капсула, це фізична оболонка персонажу. Пуста кімната, статичне восвітленнявіф

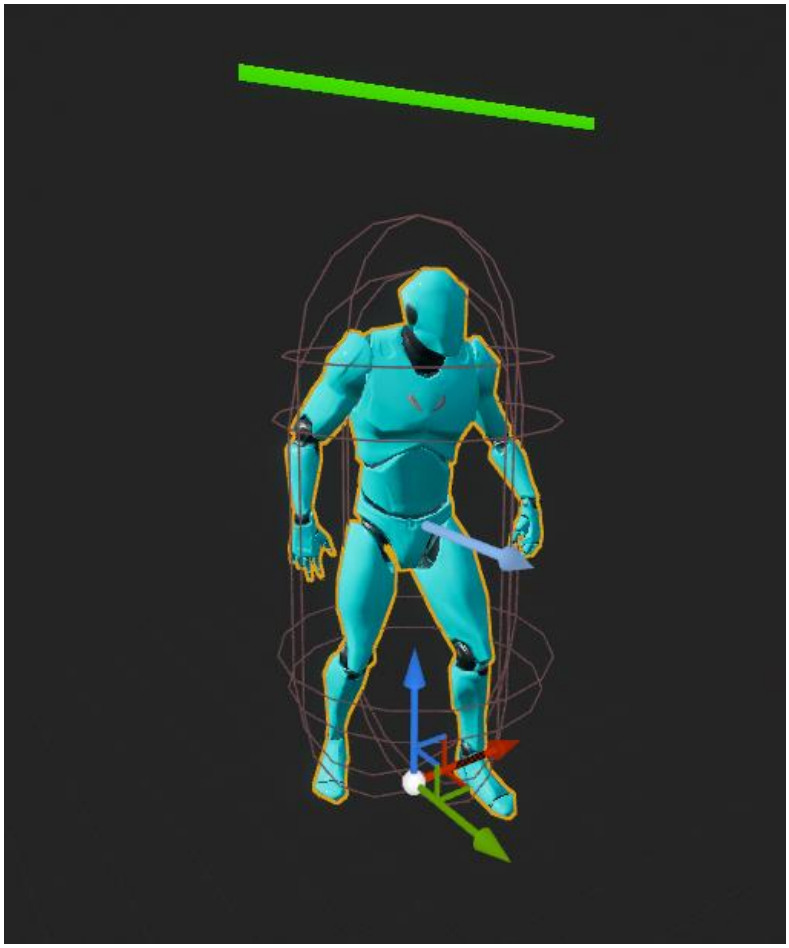

Рис. 4.8. Зображення персонажу

Висновки до розділу 4

Засобами ігрового рушія Unreal Engine 4 була реалізована інтелектуальна система моделювання поведінки персонажу. Система заснована на об’єднанні алгоритму Мамдані та дерев поведінки.

Для використання алгоритму Мамдані була створена функціональна бібліотека, яка реалізує кожен шаг алгоритму. Також були створені окремі структури даних для зручності використання алгоритму.

Для керування поведінкою персонажу на низькому рівні було розроблено дерево рішень. Для кожної з цілей, яка може бути обрана алгоритмом Мамдані, створено декілька стратегій поведінки, вибір стратегій ґрунтується на вхідних параметрах, які не має сенсу приводити до нечіткості, тому алгоритм нечіткого виведення не приймає участі у виборі стратегії.

Був розроблений персонаж, який має людиноподібні форми та розміри. Персонаж був наділений органами чуття, таким як зір, слух та дотик. Цей набор дозволяє персонажу без проблем орієнтуватися у просторі.

ВИСНОВКИ

В результаті виконання магістерської кваліфікаційної роботи було досліджено сучасні підходи до створення ігрового штучного інтелекту. Були досліджені та описані основні принципи та технології, що використовуються у ігровому ШІ, оцінена перспективність цієї галузі та проведено аналіз основних напрямків розвитку. Було з'ясовано, що значна частина методів не є оптимальною, та не дають можливості створювати дійсно гнучкий штучний інтелект. Спираючись на результати аналізу можна зробити висновок, що головними проблемами ігрового ШІ є високий рівень передбачуваності поведінки агента, та швидкість обчислень.

Проведений аналіз і дослідження літературних джерел показав наявність декількох варіантів вирішення проблеми передбачуваності поведінки, які можуть бути використані під час процесу гри, не критично впливаючи на швидкість обчислень.

У другому розділі були розглянуті методи та моделі, які можуть вирішити поставлену задачу. В основному це використання нечіткої логіки або нейронних мереж. Нечітка логіка може бути потужною технікою ШІ, особливо завдяки можливості моделювати нелінійності, досягати складної поведінки за допомогою простих правил, бути кандидатом на навчання та імітувати людські міркування та емоції.

У третьому розділі була запропонована модель, що використовує алгоритм нечіткого логічного виводу Мамдані, для прийняття рішень, та дерева поведінки для реалізації прийнятого рішення. Поєднання цих двох методів дозволяють розробити інтелектуальну систему, що буде зручна в використанні, та мати переваги використання нечіткої логіки.

Четвертий розділ демонструє реалізацію інтелектуальної системи моделювання поведінки персонажу, на основі ігрового рушія Unreal Engine 4. Створен персонаж, наділений органами чуття, що дозволяє йому орієнтуватися у просторі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Ian Millington. Artificial Intelligence for Games (The Morgan Kaufmann Series in Interactive 3D Technology). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

2. Mat Buckland. Programming Game IIII by Example. Jones & Bartlett Publishers, 1 edition, September 2004.

3. Daniel Johnson and Janet Wiles. Computer games with intelligence. In In Procs. 10th IEEE Intl Conf. on Fuzzy Systems, pages 61–68. IEEE, 2001.

4. Alan Turing. Computing machinery and intelligence. Mind LIX, 1950

5. Bob Scott. The illusion of intelligence. IIII Game Programming Wisdom, 2002.

6. Hugo Pinto and Luis Otavio Alvares. Behavior-baed robotic architectures for games. Game Programming Gems 6, 2006.

7. Paul Tozour. Evolution of game IIII. IIII Game Programming Wisdom, 2002.

8. G. Skinner and T. Walmsley, "Artificial Intelligence and Deep Learning in Video Games A Brief Review," 2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS), Singapore, 2019, pp. 404-408.

9. S. Sadeghi Esfahlani, J. Butt and H. Shirvani, "Fusion of Artificial Intelligence in Neuro-Rehabilitation Video Games," in IEEE Access, vol. 7, pp. 102617-102627, 2019.

10. N. A. Barriga, M. Stanescu, F. BesoIIIIn and M. Buro, "Improving RTS Game IIII by Supervised Policy Learning, Tactical Search, and Deep Reinforcement Learning," in IEEE Computational Intelligence Magazine, vol. 14, no. 3, pp. 8- 18, Aug. 2019.R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press

11. R. Tan, J. Zhou, H. Du, S. Shang and L. DIIII, "An modeling processing method for video games based on deep reinforcement learning," 2019 IEEE 8th Joint

International Information Technology and Artificial Intelligence Conference (ITIIIC), Chongqing, China, 2019, pp. 939-942

12. D. Perez, "Tutorial I: Video Game Description Language (VGDL) and the challenge of creating agents for General Video Game Playing (GVGP)," 2015 IEEE Conference on Computational Intelligence and Games (CIG), TIIIInan, 2015, pp. 20-20.

13. Steven Woodcock. Game IIII: the state of the industry. 1999.

14. Steven Woodcock. Game IIII: the state of the industry. 2000.

15. Daniele Loiacono, Julian Togelius, Pier Luca Lanzi, Leonard KinnIIIIrd-heether, Simon M. Lucas, Matt Simmerson, Diego Perez, Robert G. Reynolds, and Yago Saez. The wcci 2008 simulated car racing competition, 2008.

16. E. Hastings, R. Guha, and K. Stanley. Automatic content generation in the galactic arms race video game. In Proceedings of the 5th international conference on Computational Intelligence and Games, CIG'09, 2009.

17. P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper, E. Postma. Adaptive game IIII with dynamic scripting, 2006

18. P. Spronck, I. Sprinkhuizen-Kuyper, E. Postma. Difficulty scaling of game IIII.

19. G. Chaslot, S. Bakkes, I. Szita and P. Spronck. Monte-Carlo Tree Search: A New Framework for Game IIII

20. C. Shannon, Programming a computer for playing chess, Philosophical Magazine 41 (1950) 256–275.

21. A. Samuel, Some studies in machine learning using the game of checkers, IBM J. Res. Develop. 3 (1959) 210–229.

22. A. Samuel, Some studies in machine learning using the game of checkers: Recent progress, IBM J. Res. Develop. 11 (1967) 601–617.

23. C. Strachey, Logical or non-mathematical programmes, in: Proc. Association for Computing Machinery Meeting, Toronto, ON, 1952, pp. 46–49

24. J. Kister, P. Stein, S. Ulam, W. Walden, M. Wells, Experiments in chess, J. ACM 4 (1957) 174–177

25. H. Simon, A. Newell, Heuristic problem solving: The next advance in operations research, Oper. Res. 6 (1958) 10.

26. Turing, A. 1950. Computing machinery and intelligence. *Mind*.

27. Lofti A. Zadeh. Fuzzy sets and systems. System Theory, 1965.

28. Jiljang Wang Gabriyel Wong. A fuzzy-control approach to managing scene complexity. Game Programming Gems 6, 2006.

29. Larry O'Brien. Fuzzy logic in games. Game Developer Magazine, 1996.

30. Ian Millington. Artificial Intelligence for Games (The Morgan Kaufmann Series in Interactive 3D Technology). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

31. Mat Buckland. Programming Game IIII by Example. Jones & Bartlett Publishers, 1 edition, September 2004.

32. David M. Bourg and Glenn Seemann. IIII for Game Developers. O'Reilly Media, 2004.

33. Mason McCuskey. Fuzzy logic for video games. Game Programming Gems 1, 2000.

34. Michael Zarozinski. An open source fuzzy library. IIII Game Programming Wisdom, 2002.

35. Bob Alexander. The beauty of response curves. IIII Game Programming Wisdom, 2002. Bob Alexander. The beauty of response curves. IIII Game Programming Wisdom, 2002.

36. Petr Musilek Yifan Li and Loren Wyard-Scott. Fuzzy logic in agent-based game design. Fuzzy Information, 2004. Processing NAFIPS '04. IEEE Annual Meeting of the, 2004

37. Thor Alexander. An optimized fuzzy logic architecture for decision-making. IIII Game Programming Wisdom, 2002.

38. Michael Zarozinski. Imploding combinatorial explosion in a fuzzy system. Game Programming Gems 2, 2001.

39. William E. Combs. The Fuzzy Systems Handbook 2nd Ed, Academic. 1999.

40. Eric Dybsand. A generic fuzzy state machine in c++. Game Programming Gems 2, 2001.

41. Sbastien Schertenleib. Designing a multilayer, pluggable IIII engine. Game Programming Gems 6, 2006.

42. Penelope Sweetser and Janet Wiles. Current IIII in games: a review. Australian Journal of Intelligent Information Processing Systems, 8(1), 2002.

43. Magy Seif El-Nasr, John Yen, and Thomas R. Ioerger. Flamefuzzy logic adaptive model of emotions. Autonomous Agents and Multi-Agent Systems, 3(3):219–257, September 2000.

44. teven Woodcock. Games making interesting use of artificial intelligence techniques. Game IIII, 1995-2000.

45. Steven Woodcock. Game IIII: the state of the industry. 1999.

46. Daniel Johnson and Janet Wiles. Computer games with intelligence. In In Procs. 10th IEEE Intl Conf. on Fuzzy Systems, pages 61–68. IEEE, 2001.

47. Kathryn E. Merrick and Mary Lou Maher. Motivated Reinforcement Learning: Curious Characters for Multiuser Games. Springer, 2009.

48. Adnan Shaout, Brady W. King, and Luke A. Reisner. Real-time game design of pac-man using fuzzy logic. Int. Arab J. Inf. Technol., 3(4):315–325, 2006.

49. Lori L. DeLooze and Wesley R. Viner. Fuzzy q-learning in a nondeterministic environment: developing an intelligent ms. pac-man agent. In Proceedings of the 5th international conference on Computational Intelligence and Games, CIG'09, pages 162–169, Piscataway, NJ, USA, 2009. IEEE Press.

50. Duc Thang Ho and Jonathan M. Garibaldi. A fuzzy approach for the 2007 cig simulated car racing competition. In Proceedings of the 4th international conference on Computational Intelligence and Games, CIG'08, 2008.

51. Diego Perez, Gustavo Recio, Yago Saez, and Pedro Isasi. Evolving a fuzzy controller for a car racing competition. In Proceedings of the 5th international conference on Computational Intelligence and Games, CIG'09, pages 263–270, Piscataway, NJ, USA, 2009. IEEE Press.

52. L. Cardamone, D. Loiacono, and P. L. Lanzi. Overtaking opponents with blocking strategies using fuzzy logic. In Proceedings of the 6th international conference on Computational Intelligence and Games, CIG'10, 2010.

53. J. M. P. van Waveren. The quake iii arena bot. Master's thesis, University of Technology Delft, 2001.

54. Joost Westra and Frank Dignum. Evolutionary neural networks for non-player characters in quake iii. In Proceedings of the 5th international conference on Computational Intelligence and Games, CIG'09, pages 302– 309, Piscataway, NJ, USA, 2009. IEEE Press.

55. Armand Prieditis and Mukesh Dalal. Applying modelbased decision-making methods to games: Applying the locus IIII engine to quake iii. Game Programming Gems 6, 2006.

56. Hugo Pinto and Luis Otavio Alvares. Costructing a goaloriented robot for unreal tournament using fuzzy sensors, finite-state machines, and extended behavior networks. Game Programming Gems 6, 2006.

57. Giovanni Acampora. Synthesizing bots emotional behaviors through fuzzy cognitive processes. In Proceedings of the 6th international conference on Computational Intelligence and Games, CIG'10, 2010.

58. F. LevillIIIIn, J. Orero, and M. Rifqi. Characterizing players experience from physiological signals using fuzzy decision trees. In Proceedings of the 6th international conference on Computational Intelligence and Games, CIG'10, 2010.

59. K. Ohsone and T. Onisawa. Friendly partner system of poker game with facial expressions. In Proceedings of the 4th international conference on Computational Intelligence and Games, CIG'08, 2008.

60. Philippa Avery and Zbigniew Michalewicz. Adapting to human game play. In Proceedings of the 4th international conference on Computational Intelligence and Games, CIG'08, 2008.

61. Hisao Ishibuchi, Ryoji Sakamoto, and Tomoharu Nakashima. Learning fuzzy rules from iterative execution of games. Fuzzy Sets and Systems, 2003.

62. Bourg, D. M., Seemann, G. (2004). IIII for Game Developers. Beijing: O'Reilly Media

63. Schwab, B. (2009). IIII game engine programming. Boston, MA: Course PTR.

64. G. Skinner and T. Walmsley, "Artificial Intelligence and Deep Learning in Video Games A Brief Review," 2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS), Singapore, 2019, pp. 404-408.

65. S. Sadeghi Esfahlani, J. Butt and H. Shirvani, "Fusion of Artificial Intelligence in Neuro-Rehabilitation Video Games," in IEEE Access, vol. 7, pp. 102617-102627, 2019.

66. N. A. Barriga, M. Stanescu, F. BesoIIIIn and M. Buro, "Improving RTS Game IIII by Supervised Policy Learning, Tactical Search, and Deep Reinforcement Learning," in IEEE Computational Intelligence Magazine, vol. 14, no. 3, pp. 8- 18, Aug. 2019.R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press

67. R. Tan, J. Zhou, H. Du, S. Shang and L. DIIII, "An modeling processing method for video games based on deep reinforcement learning," 2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITIIIIC), Chongqing, China, 2019, pp. 939-942

68. D. Perez, "Tutorial I: Video Game Description Language (VGDL) and the challenge of creating agents for General Video Game Playing (GVGP)," 2015 IEEE Conference on Computational Intelligence and Games (CIG), TIIIInan, 2015, pp. 20-20.

69. Mateas, M. and Stern, A. (2002). A behavior language for story-based believable agents. IEEE Intelligent Systems, 17(4):39–47.

70. Florez-Puga, G., Gomez-Martin, M. A., Gomez-Martin, P. P., Diaz-Agudo, B., and Gonzalez-Calero, P. A. (2009). QueryEnabled Behavior Trees. IEEE Transactions on Computational Intelligence and IIII in Games, 1(4):298–308.

71. Ögren, P. (2012). Increasing Modularity of UAV Control Systems using Computer Game Behavior Trees. In IIIIAA Guidance, Navigation, and Control Conference 2012; Minneapolis, MN; United States; 13 August 2012 through 16 August 2012.

72. Bagnell, J. A., Cavalcanti, F., Cui, L., Galluzzo, T., Hebert, M., Kazemi, M., Klingensmith, M., Libby, J., Liu, T. Y., Pollard, N., PivtorIIIIko, M., Valois, J., and Zhu, R. (2012). An integrated system for autonomous robotics manipulation. In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 2955–2962.

73. Chen, J. and Shi, D. (2018). Development and Composition of Robot Architecture in Dynamic Environment. In Proceedings of the 2018 International Conference on Robotics, Control and Automation Engineering, RCAE 2018, pages 96–101, Beijing, China. ACM.

74. Colledanchise, M. and Ögren, P. (2017). How Behavior Trees Modularize Hybrid Control Systems and Generalize Sequential Behavior Compositions, the Subsumption Architecture, and Decision Trees. IEEE Transactions on Robotics, 33(2):372–389. KTH.

75. Colledanchise, M. and Ögren, P. (2016). How Behavior Trees generalize the Teleo-Reactive paradigm and And-OrTrees. In 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 424–429.

76. Nilsson, N. (1993). Teleo-reactive programs for agent control. Journal of artificial intelligence research, 1:139–158.

77. Merrill, B. (2013). Building Utility Decisions into Your Existing Behavior Tree. In Game IIII Pro, pages 127–136. CRC Press.

78. Hannaford, B., Hu, D., Zhang, D., and Li, Y. (2016). Simulation Results on Selector Adaptation in Behavior Trees. arXiv:1606.09219 [cs].

79. Colledanchise, M., Marzinotto, A., and Ögren, P. (2014). Performance analysis of stochastic behavior trees. In 2014 IEEE International Conference on Robotics and Automation (ICRA), pages 3265–3272.

80. Hannaford, B. (2019). Hidden Markov Models derived from Behavior Trees. arXiv:1907.10029 [cs].

81. Klöckner, A. (2015). Behavior Trees with Stateful Tasks. In Bordeneuve-Guibé, J., Drouin, A., and Roos, C., editors, Advances in Aerospace Guidance, Navigation and Control, pages 509–519. Springer International Publishing, Cham.

82. Nilsson, N. (1993). Teleo-reactive programs for agent control. Journal of artificial intelligence research, 1:139–158.

83. Colledanchise, M. and Ögren, P. (2017). How Behavior Trees Modularize Hybrid Control Systems and Generalize Sequential Behavior Compositions, the Subsumption Architecture, and Decision Trees. IEEE Transactions on Robotics, 33(2):372–389. KTH.

84. Sprague, C. I. and Ögren, P. (2018). Adding Neural Network Controllers to Behavior Trees without Destroying Performance Guarantees. arXiv:1809.10283 [cs].

85. Paxton, C., Ratliff, N., Eppner, C., and Fox, D. (2019). Representing robot task plans as robust logical-dynamical systems. In 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 5588–5595.

86. Rovida, F., Grossmann, B., and Krüger, V. (2017). Extended behavior trees for quick definition of flexible robotic tasks. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 6793–6800.

87. Rovida, F., Wuthier, D., Grossmann, B., Fumagalli, M., and Krüger, V. (2018). Motion Generators Combined with Behavior Trees: A Novel Approach to Skill Modelling. In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 5964–5971.

88. Colledanchise, M. and Natale, L. (2018). Improving the Parallel Execution of Behavior Trees. In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 7103–7110

89. Shoulson, A., Garcia, F. M., Jones, M., Mead, R., and Badler, N. I. (2011). Parameterizing Behavior Trees. In Allbeck, J. M. and Faloutsos, P., editors, Motion in Games, Lecture Notes in Computer Science, pages 144–155. Springer Berlin Heidelberg.

90. Rovida, F., Wuthier, D., Grossmann, B., Fumagalli, M., and Krüger, V. (2018). Motion Generators Combined with Behavior Trees: A Novel Approach to Skill Modelling. In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 5964–5971.

91. Lim, C.-U., Baumgarten, R., and Colton, S. (2010). Evolving Behaviour Trees for the Commercial Game DEFCON. In Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Mattern, F., Mitchell, J. C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M. Y., Weikum, G., Di Chio, C., Cagnoni, S., Cotta, C., Ebner, M., Ekárt, A., Esparcia-Alcazar, A. I., Goh, C.-K., Merelo, J. J., Neri, F., Preuß, M., Togelius, J., and Yannakakis, G. N., editors, Applications of Evolutionary Computation, volume 6024, pages 100–110. Springer Berlin Heidelberg, Berlin, Heidelberg.

92. Oakes, B. J. (2013). Practical and Theoretical Issues of Evolving Behaviour Trees for a Turn-Based Game. Master thesis, McGill University

93. Palma, R., González-Calero, P. A., Gómez-Martín, M. A., and Gómez-Martín, P. P. (2011a). Extending Case-Based Planning with Behavior Trees. In Twenty-Fourth International FLIIIIRS Conference.

94. Weber, B. G., Mawhorter, P., Mateas, M., and Jhala, A. (2010). Reactive planning idioms for multi-scale game IIII. In Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games, pages 115–122, Copenhagen, Denmark. IEEE.

95. Weber, B. G., Mateas, M., and Jhala, A. (2011). Building Human-Level IIII for Real-Time Strategy Games. In 2011 AAIIII Fall Symposium Series

96. Flórez-Puga, G., Gómez-Martín, M., Díaz-Agudo, B., and González-Calero, P. A. (2008). Dynamic Expansion of Behaviour Trees. In Proceedings of the Fourth AAIIII Conference on Artificial Intelligence and Interactive Digital EntertIIIInment, IIIIIDE'08, pages 36–41, Stanford, California. AAIIII Press.

97. Iske, B. and Ruckert, U. (2001). A methodology for behaviour design of autonomous systems. In Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180), volume 1, pages 539–544. IEEE.

98. Buche, C., Even, C., and Soler, J. (2020). Orion: A Generic Model and Tool for Data Mining. In Gavrilova, M. L., Tan, C. K., and Sourin, A., editors, Transactions on Computational Science XXXVI: Special Issue on Cyberworlds and Cybersecurity, Lecture Notes in Computer Science, pages 1–25. Springer, Berlin, Heidelberg.

99. Nicolau, M., Perez-Liebana, D., O'Neill, M., and Brabazon, A. (2017). Evolutionary Behavior Tree Approaches for Navigating Platform Games. IEEE Transactions on Computational Intelligence and IIII in Games, 9(3):227–238.

100. Dagerman, B. (2017). High-Level Decision Making in Adversarial Environments Using Behavior Trees and Monte Carlo Tree Search. Master thesis, KTH Royal Institute of Technology.

101. Гетия И.Г., Леонтьева И.Н., Шумилин В.К. Методические указания по проведению занятия по дисциплине «Безопасность жизнедеятельности » на тему: «Определение интегральной бальной оценки тяжести труда на робочем месте» . – М.: МГАПИ, 2002 . – 22 с.

102. Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу //Охорона праці. – 2001. –№ 12. – С. 12-20.

103. Жидецький В.Ц. Основи охорони праці [текст]: Підручник. – Львів: УАД, 2006. – 336 с.

104. НПАОП 0.00-1.28-10. Правила охорони праці під час експлуатації електронно-обчислювальних машин.

105. /IEC 31010:2009. Менеджмент ризиків. Методи оцінки ризиків.

106. Бикова О.В. Основи цивільного захисту [текст]: навч. посіб. / О. В. Бикова, О. Ч. Болієв, Д. М. Деревинськийк та ін. – К.: МНС України, Ун-тет цивільного захисту України, Ін-тут держ. упр. у сфері цивільного захисту 2008. – 223 с.

ДОДАТОК А

Всеукраїнська науково-практичній конференція молодих вчених, аспірантів і студентів

Міністерство освіти і науки України
Чорноморський національний університет
імені Петра Могили

PETRO MOHYLA BLACK SEA NATIONAL UNIVERSITY
COMPUTER SCIENCE FACULTY
Est. 1996

«Інформаційні технології та інженерія»

Всеукраїнська науково-практична конференція молодих вчених, аспірантів і студентів

ТЕЗИ

9–11 лютого 2022 року

Миколаїв – 2022

Рис. А.1. Головна сторінка конференції

Системний аналіз, моделі і засоби підтримки прийняття рішень

Рис. А.2. Зміст конференції

УДК 681.3

Под'ячев А. Д., Гожий О. П.
Чорноморський національний університет імені Петра Могили,
м. Миколаїв, Україна

ВИКОРИСТАННЯ НЕЧІТКОЇ ЛОГІКИ В ІГРОВОМУ ШТУЧНОМУ ІНТЕЛЕКТІ

Ігровий штучний інтелект привертає увагу науковців та індустрії протягом останнього десятиліття. Сфера ігрового штучного інтелекту (ігровий ШІ) охоплює всі технології та методи введення інтелекту в відео ігри. До ігрового ШІ відноситься багато різних аспектів відеоігор: керування анімацією, керування, пошук шляху, планування, процедурна генерація, тактичне та стратегічне мислення та навчання [1]. Всі ці аспекти мають спільну основу, вони ставлять проблеми, ефективне вирішення яких вимагає алгоритмів ШІ. Мета ігрового ШІ – оживити неігрових персонажів (NPC), присутніх в іграх, наприклад, ворожі війська в стратегії або купців у мирному селищі, контрольованому штучним інтелектом. Ігровий ШІ, як і всі інші аспекти відеоігри, існує для досягнення вищої мети відеоігри: бути цікавою та розважальною, іншими словами, суспільство часто вважає девізом ігрового ШІ «Якщо гравець не бачить цього, навіщо це робити?» [1]. Проте контекст нині не такий простий, почасти завдяки успішним продуктам, які висунули передовий AI, частково завдяки поширені онлайн-ігор для кількох гравців, які змінили звички гравців, гравці тепер більш вимогливі до ШІ, і все більше і більше розробників ігор переходять від використання сценарних і передбачуваних агентів до використання переважно людиноподібних агентів, здатних до навчання і, отже, за класичним визначенням ШІ, розумні.

Академічні дослідження штучного інтелекту завжди відчували труднощі з визначенням самого штучного інтелекту, хоча зазвичай його можна визначити, як когнітивні процеси, схожі до когнітивних

Рис. А.3. Зміст тези «Використання нечіткої логіки в ігровому штучному інтелекті»

процесів людини або на рівні когнітивних процесів людини, і в здатність цих процесів до навчання [3]. За словами Алана Тюрінга, якого вважають батьком штучного інтелекту, агент розумний, якщо його поведінку неможливо розрізнити від поведінки людини.

Головна особливість ігрового ШІ полягає в тому, що ігри, здебільшого, обмежені виконанням у режимі реального часу, тому залишається мало місця для повільних автономних методів. Цей факт робіть ігровий ШІ подібним до ШІ, що використовується в робототехніці та системах керування [2], і це є причиною того, що можна використовувати, в тій чи іншій формі, ті ж самі прийоми, що використовуються у цих галузях.

Ще одна особливість це те, що сучасні ігри постійно удосконалюють та нарощують свої візуальні та аудіо складові, щоб залучити більше споживачів. Таким чином, більшість обчислювальних потужностей зайняті графічними обчисленнями, за якими часто слідувала обробка звуку і, звичайно, ігрова логіка. Це залишило ШІ дуже мало ресурсів для виконання своїх обчислень, що призвело до таких особливостей, як наприклад розповсюдження ворогів з низьким рівнем інтелекту у старих іграх. Порив до кращого, більш реалістичного та приємної графіки сповільнилася, деякі графічна складова деяких сьогоднішніх ігор практично не відрізняються від реальності. Зараз більшість із них підтримують виділені графічні процесори (GPU), які виконують графічні обчислення, залишаючи більше місця в ЦП для інших компонентів з потребуючих великих обсягів обчислень, таких як фізика та ШІ. У цьому контексті ігровий штучний інтелект заробив свою славу як засіб виділитися серед конкурентів. Нині більше часу потужності ЦП надаються ШІ в кожному ігровому кадрі, чого більше ніж достатньо для більшості програм, і деяких командах є програмісти, чиєю єдиною задачею є створенню складного, цікавого та (здавалося б) розумного.

Ігровий ШІ використовує багато методів із ширшої області ШІ, від простих кінцевих автоматів до найсучасніших еволюційних алгоритмів. Серед цих методів нечітка логіка є одним із інструментів повинен бути присутнім в арсеналі хорошого розробника ігрового ШІ через простоту її формулювання в поєднанні з її можливостями. Нечітка логіка приносить багато переваг для моделювання інтелектуальних ігрових агентів. Головна перевага – простота формулювання, що дозволяє розробникам з легкістю інтегрувати його в багато ігор, шо є великою вигодою, якщо врахувати щільні графіки розробників ігор. Навіть у своїй простоті нечітка логіка може бути потужною технікою ШІ, особливо завдяки можливостям моделювати нелінійності, досягати складної поведінки за допомогою простих

Рис. А.4. Продовження змісту тези «Використання нечіткої логіки в ігровому штучному інтелекті»

правил, бути інструментом для моделювання та відображення людських міркувань та емоцій.

Ігровий AI – це лише одна з галузей більш широкого поля штучного інтелекту. Ігровий ШІ додає ще один елемент до задачі визначення ШІ, оскільки вона не стосується реального існування інтелекту, але лише з ілюзією його. Щоб гра була успішною, нам насправді не потрібні високоінтелектуальні, схожі на людину, можливо, непереможні суперники, потрібен переконливий супротивник, який є агентом ШІ з яким приємно грати, та який симулює поведінку реальної людини чи істоти. При програмній реалізації поведінки ігрового персонажу нечітка логіка є однім з основних інструментів «інтелектуалізації» його поведінки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ian Millington. Artificial Intelligence for Games (The Morgan Kaufmann Series in Interactive 3D Technology). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

2. Daniel Johnson and Janet Wiles. Computer games with intelligence. In In Procs. 10th IEEE Intl Conf. on Fuzzy Systems, pages 61–68. IEEE, 2001.

3. Hugo Pinto and Luis Otavio Alvares. Behavior-based robotic architectures for games. Game Programming Gems 6, 2006.

Рис. А.5. Кінець змісту тези «Використання нечіткої логіки в ігровому штучному інтелекті»