БАКАЛАВРСЬКА РОБОТА

БР. ІП - 80.00.00.000 ПЗ

Група ІП-22-3

Саюк Павло

2026

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Саюк Павло Андрійович

(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

БАКАЛАВРСЬКА РОБОТА

Застосування ШІ для автоматичного балансування ігрових механік
(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____Саюк П.А._____
(підпис, ініціали та прізвище здобувача)

Науковий керівник _____Ваврик Тетяна Олександрівна, асистент
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

_доц._____Бандура В.В.__
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доцент. В.В. Бандура
“ ____ ” _____2026 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Саюк Павло Андрійович

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) "Застосування ШІ для автоматичного балансування ігрових механік "

керівник проекту (роботи) Ваврик Тетяна Олександрівна, асистент

2. Строк подання студентом проекту (роботи) 10 червня 2026 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Аналіз вимог до програмного забезпечення
2. Аналіз вимог і постановка задачі
3. Проектування системи
4. Реалізація проекту
5. Тестування та впровадження

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

1 Інтелектуальний агент (рис. 1.1, ст. 12)
2 Інтелектуальна машина з обмеженими ресурсами (рис. 1.2, ст. 13)
3 Спрощена діаграма дизайну гри Pong (рис. 1.9, ст.20)
4 Приклад дерева поведінки (рис. 2.3, ст. 49)
5 Дерево поведінки, що використовується в DiskiSimulation. (рис 3.14, ст. 78)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

2. Дата видачі завдання **2026 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Визначення та обгрунтування теми роботи	17.01.2026	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	21.02.2026	виконано
3	Побудова моделі або алгоритму власного рішення	01.03.2026	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	21.04.2026	виконано
5	Оформлення пояснювальної записки бакалаврської роботи	02.05.2026	виконано

Студент _____Саюк П.А.
(підпис)

Керівник роботи _____ Ваврик Т.О
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 85 сторінки, 39 рисунків, список використаних джерел із 40 найменувань.

Метою роботи є дослідження методів та моделей застосування штучного інтелекту для автоматичного балансування ігрових механік та підвищення якості ігрового процесу.

Об'єкт дослідження: ігрові системи з адаптивною поведінкою.

Предмет дослідження: методи та алгоритми штучного інтелекту, що застосовуються для автоматичного балансування складності та ігрових механік у відеоіграх.

Результати дослідження: проведено аналіз підходів до адаптивного балансування ігор, досліджено методи машинного навчання та інтелектуальні алгоритми прийняття рішень, розроблено архітектуру системи автоматичного балансування та виконано оцінку її ефективності.

У першому розділі розглянуто теоретичні основи застосування ШІ в іграх, концепції адаптивного ШІ та методи балансування ігрових механік.

У другому розділі досліджено методи машинного навчання та інтелектуальні алгоритми, що використовуються для автоматичного балансування, а також розглянуто архітектуру відповідних систем.

У третьому розділі представлено реалізацію системи автоматичного балансування, описано алгоритми оптимізації ігрових параметрів у реальному часі та проведено оцінку ефективності запропонованого підходу.

Висновок: застосування штучного інтелекту для автоматичного балансування ігрових механік дозволяє підвищити адаптивність ігрового процесу, покращити користувацький досвід та забезпечити оптимальний рівень складності гри.

КЛЮЧОВІ СЛОВА: ШТУЧНИЙ ІНТЕЛЕКТ, ІГРОВІ МЕХАНІКИ, АДАПТИВНИЙ ШІ, МАШИННЕ НАВЧАННЯ, БАЛАНСУВАННЯ ГРИ, ІГРОВИЙ ПРОЦЕС, ОПТИМІЗАЦІЯ, GAME AI.

ANNOTATION

Bachelor's thesis of 85 pages, 39 figures, and a reference list containing 40 sources.

The aim of the work is to study methods and models of artificial intelligence for automatic balancing of game mechanics and improving gameplay quality.

Research object: game systems with adaptive behavior.

Research subject: artificial intelligence methods and algorithms used for automatic balancing of game difficulty and mechanics in video games.

Research results: approaches to adaptive game balancing were analyzed, machine learning methods and intelligent decision-making algorithms were studied, the architecture of an automatic balancing system was developed, and its effectiveness was evaluated.

The first section describes the theoretical foundations of artificial intelligence in games, concepts of adaptive artificial intelligence, and methods for balancing game mechanics.

The second section analyzes machine learning methods and intelligent algorithms used for automatic balancing, as well as the architecture of such systems.

The third section presents the implementation of an automatic balancing system, describes real-time game parameter optimization algorithms, and evaluates the effectiveness of the proposed approach.

Conclusion: the use of artificial intelligence for automatic balancing of game mechanics improves gameplay adaptability, enhances user experience, and ensures an optimal level of game difficulty.

KEY WORDS: ARTIFICIAL INTELLIGENCE, GAME MECHANICS, ADAPTIVE AI, MACHINE LEARNING, GAME BALANCING, GAMEPLAY, OPTIMIZATION, GAME AI.

ЗМІСТ

ВСТУП

У сучасній індустрії відеоігор стрімкий розвиток технологій та зростання вимог користувачів до якості ігрового досвіду зумовлюють необхідність створення інтелектуальних систем, здатних адаптувати ігровий процес у реальному часі. Одним із ключових аспектів успішної гри є збалансованість її механік, що забезпечує оптимальний рівень складності, утримання уваги гравця та підвищення рівня задоволеності від гри. Традиційні підходи до балансування ігрових механік, які базуються на ручному налаштуванні параметрів, є трудомісткими, суб'єктивними та недостатньо ефективними в умовах динамічного ігрового середовища.

Актуальність теми зумовлена необхідністю застосування сучасних методів штучного інтелекту для автоматизації процесу балансування ігрових механік. Використання адаптивного ШІ та алгоритмів машинного навчання дозволяє аналізувати поведінку гравця, прогнозувати його дії та динамічно змінювати параметри гри з метою забезпечення оптимального ігрового досвіду. Існуючі підходи часто обмежуються статичними моделями або простими евристиками, що не враховують індивідуальні особливості гравців і не забезпечують достатнього рівня гнучкості.

Метою роботи є дослідження та розробка підходів до застосування штучного інтелекту для автоматичного балансування ігрових механік з метою підвищення адаптивності та ефективності ігрових систем.

Для досягнення поставленої мети необхідно вирішити такі завдання: провести аналіз предметної області та існуючих підходів до балансування ігор; дослідити теоретичні основи застосування штучного інтелекту в ігрових системах; розглянути методи машинного навчання та інтелектуальні алгоритми, що можуть бути використані для адаптивного балансування; спроєктувати архітектуру системи автоматичного балансування; реалізувати прототип системи з використанням сучасних технологій; провести оцінку ефективності запропонованого підходу та сформулювати висновки.

Об’єктом дослідження є ігрові системи з адаптивною поведінкою, що забезпечують динамічну зміну параметрів ігрового процесу.

Предметом дослідження є методи, моделі та алгоритми штучного інтелекту, що застосовуються для автоматичного балансування ігрових механік, зокрема методи машинного навчання, алгоритми прийняття рішень та підходи до оптимізації параметрів гри.

Методи дослідження включають аналіз наукових джерел для вивчення сучасних підходів до балансування ігор, порівняльний аналіз алгоритмів штучного інтелекту, моделювання ігрових процесів, експериментальні методи для реалізації системи автоматичного балансування та оцінки її ефективності.

У роботі передбачається розробка системи автоматичного балансування ігрових механік, яка здатна адаптувати параметри гри в реальному часі залежно від поведінки користувача. Особлива увага приділяється використанню алгоритмів машинного навчання та інтелектуального аналізу даних для підвищення точності прийняття рішень. Очікується, що впровадження запропонованого підходу дозволить підвищити якість ігрового процесу, покращити взаємодію користувача з грою та забезпечити оптимальний рівень складності.

Бакалаврська робота містить 85 сторінки, 39 рисунків, 3 розділи, список використаних джерел із 40 найменувань.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ЗАСТОСУВАННЯ ШІ В ІГРОВИХ СИСТЕМАХ

1.1 Основи штучного інтелекту в іграх та ігрових агентів

Щоб обговорити агентів у контексті штучного інтелекту, спочатку розглянемо загальне визначення агента . Словник Визначення агента говорить: «Людина, яка діє від імені іншої особи чи групи». Зі словникового значення цього терміна можна зробити висновок, що основне значення агента полягає в тому, щоб діяти від імені однієї або кількох сутностей. Поняття агента, який діє як посередник, або навіть принципал, є дуже актуальним для ШІ.

Також важливо охарактеризувати *інтелект* у цьому контексті. Не існує універсального визначення інтелекту [1]. Одне збірне визначення інтелекту говорить: «Здатність використовувати пам'ять, знання, досвід, розуміння, міркування, уяву та судження для вирішення проблем та адаптації до нових ситуацій». З цього визначення можна зробити висновок, що інтелект - це не окрема кількісно вимірювана характеристика, а результат поєднання різних характеристик. Більше того, поєднання цих характеристик дозволяє власнику досягати цілей та адаптуватися до нових ситуацій.

Людський розум має бути найрепрезентативнішим прикладом інтелекту, хоча слід також розглядати й інші види нелюдського інтелекту. Вузьке розгляд лише людського розуму призведе до обмеження інтелекту лише людським інтелектом. Однак визначення інтелекту також не повинно бути настільки широким, щоб воно включало машини чи комп'ютери, які демонструють відносно низький рівень інтелекту.

Отже, якщо людський розум має використовуватися переважно як еталон інтелекту, не слід очікувати, що інтелектуальні агенти досягатимуть цілей заздалегідь визначеними правильними способами (оскільки саме так створюються традиційні комп'ютери, які демонструють низький рівень інтелекту). Натомість, інтелектуальним агентам слід дозволити знаходити свої оптимальні рішення

найкращим можливим *способом*. Такий спосіб міркування про інтелектуальних агентів відповідає ідеї раціональності.

Коротко кажучи, раціонального агента можна уявити як такого, що демонструє найбільш задовільну або «правильну» поведінку, враховуючи його знання. Правильність дій агента можна виміряти тим, наскільки ближче вони наближають агента до досягнення його цілей [2]. У цій роботі аналогічно визначено інтелектуального агента. З цього моменту і надалі інтелектуальний агент – це той, хто діє найбільш раціональним чином, враховуючи його знання та мету(цілі).

Цей розділ відхиляється від спроби відповісти на широке питання інтелекту , а натомість зосереджується на раціональності як керівному принципі інтелектуальних агентів. Таким чином, інтелектуальний агент – у контексті штучного інтелекту – це артефакт, який може сприймати своє оточення, оновлювати свою базу знань та діяти на оточення. Рисунок 1.1 ілюструє цю концепцію.

Запропоновано три характеристики, якими повинен володіти інтелектуальний агент:

1. Реактивність
2. Проактивність
3. Соціальні здібності

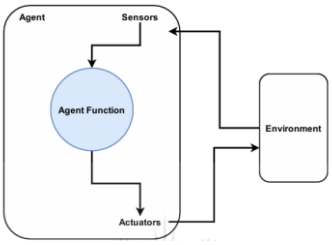

Рисунок 1.1 - Інтелектуальний агент

Інтелектуальний агент повинен мати здатність своєчасно *реагувати* на зміни у своєму середовищі, діючи відповідно до змін, що відбулися, та мети агента. Цілеспрямована поведінка вимагає, щоб інтелектуальний агент динамічно діяв у своєму середовищі для досягнення своїх цілей. Нарешті, соціальна

здатність стосується здатності інтелектуального агента взаємодіяти з іншими агентами. Соціальна здатність також означає, що агент повинен мати можливість вести переговори та співпрацювати з іншими агентами.

Ментальні концепції переконання, бажання та наміру (BDI) також мають значний вплив на проектування інтелектуальних агентів. Братман запропонував об'єднати переконання , бажання та намір в єдину теорію BDI, і ця пропозиція була спричинена безліччю досліджень, заснованих на парадигмі BDI. Багато архітектур BDI дотримуються архітектури інтелектуальної машини з обмеженими ресурсами, зображеної на рисунку 1.2. BRF розшифровується як Belief Revision Function (Функція перегляду переконань).

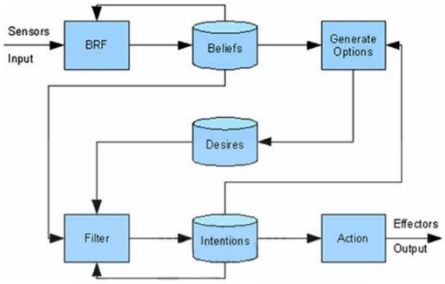

Рисунок 1.2 - Інтелектуальна машина з обмеженими ресурсами

Компонент переконання стосується інформації, яку агент має про середовище, та правил логічного висновку, які дозволяють агенту міркувати про своє оточення. Крім того, переконання агента не обов'язково мають бути завжди істинними (звідси термін «переконання»). Компонент бажання стосується цілей агента. Цілі агента – це стани, яких агент хотів би досягти.

Останній компонент – намір – стосується дій, які агент готовий вжити для досягнення своїх цілей. Наміри можуть мати різні рівні абстракції. Наприклад, наміром може бути досягнення певного пункту призначення, що включає дії нижчого рівня, такі як планування поїздки та визначення найкращого маршруту (вони також можуть призвести до інших дій нижчого рівня).

Класи інтелектуальних агентів

У попередньому розділі ми розглянули різні типи середовищ завдань, у

яких можуть бути знайдені інтелектуальні агенти. Обговорювані властивості не приділяли уваги фізичним властивостям середовищ, а радше вимірам, корисним для оцінки та порівняння середовищ завдань [3]. У цьому підрозділі розглядаються різні класи інтелектуальних агентів.

Центральним компонентом інтелектуального агента є функція агента. Функція агента – це компонент агента, який перетворює сприйняте агентом в його наступні дії. Таким чином, агент вважається комбінацією двох основних компонентів: архітектури агента та самої програми агента. Існує чотири фундаментальні класи агентів:

1. Прості рефлекторні агенти
2. Рефлексні агенти на основі моделей
3. Агенти, орієнтовані на цілі
4. Агенти на основі комунальних послуг

Прості рефлекторні агенти

Цей тип агента є найпростішим видом інтелектуального агента. Він діє лише на поточне сприйняття, не зберігає історію сприйняття та не враховує та не розмірковує над наслідками своїх дій. Простий рефлекторний агент працює за так званими **правилами «умова-дія».** Тому, якщо спостережувана умова не має пов'язаної дії в наборі правил програми агента, агент не діє. Будь ласка, зверніться до рисунка 1.3 для ілюстрації простого рефлекторного агента.

Хоча прості рефлексні агенти мають відносно просту внутрішню структуру, вони мають значно широку область застосування. Наприклад, прості рефлексні агенти використовувалися як частина MAS у системі виявлення вторгнень [4]. Прості рефлексні агенти також використовувалися в децентралізованому управлінні енергією та взаємодії з інтелектуальними мережами. Варто також зазначити, що прості рефлексні агенти здебільшого підходять для використання в повністю спостережуваних середовищах. Розміщення простого рефлексного агента в частково спостережуваному середовищі може легко призвести до того, що такий агент потрапить у небажані ситуації, такі як нескінченні цикли.

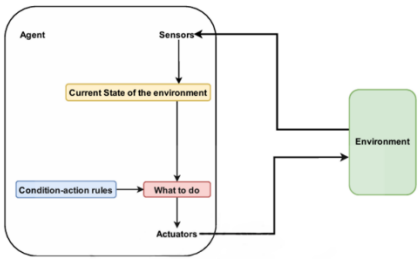

Рисунок 1.3 - Агент простого рефлексу.

Модельно-орієнтовані рефлексивні агенти

На рисунку 1.4 зображено рефлексний агент на основі моделі. Цей тип агента все ще використовує правила умова-дія, як і простий рефлексний агент. Однак, простий рефлексний агент на основі моделі також підтримує внутрішнє представлення свого середовища. Внутрішня модель допомагає рефлексному агенту на основі моделі контролювати поточний стан середовища, що робить його застосовним до частково спостережуваних середовищ.

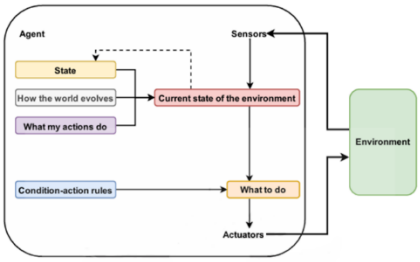

Рисунок 1.4 - Рефлексивний агент на основі моделі

Модельно-орієнтований рефлексивний агент може зберігати історію сприйняттів та звертатися до них, коли потрібно прийняти рішення, навіть коли не всі аспекти середовища є спостережуваними. Таким чином, збереження історії сприйняттів також допомагає модельно-орієнтованому рефлексивному агенту передбачати, як зміниться світ, коли будуть виконані певні дії.

Агенти, орієнтовані на цілі

Хоча рефлексивні агенти на основі моделей можуть бути використані для вирішення проблем у частково спостережуваних середовищах, їх можна покращити, встановивши цілі. Наявність цілі, ймовірно, призведе до того, що агент

виконуватиме цілеспрямовані дії та, таким чином, поводитиметься розумніше [6]. Тому цілеспрямований агент враховує як поточний стан середовища, так і свою ціль, щоб вибрати дію. Рисунок 1.5 ілюструє внутрішню структуру цілеспрямованого агента.

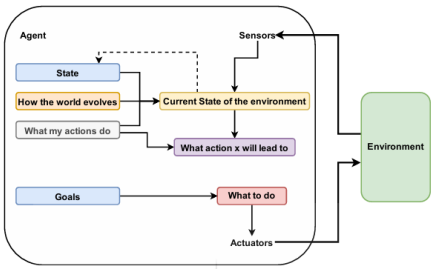

Рисунок 1.5 - Агент, що орієнтований на ціль.

Оскільки агент, орієнтований на ціль, виконує дії, які, ймовірно, призведуть до досягнення ним цієї мети, зміна поведінки агента часто вимагає оновлення його мети, а не оновлення набору правил умов-дій (як це було б вимогою для рефлексних агентів).

Агенти на основі утиліт

Короткий огляд агентів, орієнтованих на цілі, показав, що їх можна розглядати як більш інтелектуальних, ніж рефлексивних агентів, орієнтованих на моделі [5]. Хоча агенти, орієнтовані на цілі, вибирають дії, пов'язані з цілями, вони не можуть вибрати найоптимальніші дії. Агенти, орієнтовані на корисність, враховують оптимальність дій, перш ніж вибрати їх. Таким чином, агенти, орієнтовані на корисність, мають потенціал вибирати найкращі шляхи до цільових станів.Агент, що базується на корисності, використовує *функцію корисності,* яка є моделлю для вимірювання продуктивності агента, таким чином гарантуючи, що агент досягає своїх цілей «найкращим» можливим способом, або, більш формально, таким чином, щоб максимізувати корисність агента. Включення корисності в агента дозволяє створити механізм для вирішення конфліктуючих цілей: функція корисності просто вибирає ціль, враховуючи компроміс між цілями та вибираючи найбажанішу ціль. Будь ласка, зверніться до рисунка 1.6 для зображення внутрішньої архітектури агента, що базується на корисності.

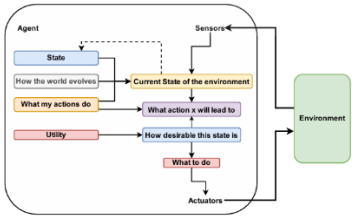

Рисунок 1.6 - Агент на основі корисності.

Навчальні агенти

Архітектури агентів, що обговорювалися досі, різняться за складністю та рівнем раціональності. Однак, на сучасному рівні досліджень у галузі штучного інтелекту, значна кількість досліджень спрямована на те, як агенти можуть покращити свої початкові знання та бути кращими. У цьому розділі розглядається агент, що навчається [7]. Агент, що навчається, не сильно відрізняється від чотирьох агентів, що обговорювалися раніше, оскільки компонент навчання може бути вбудований і в інших агентів.

На рисунку зображено критичні компоненти агента, що навчається. Стандарт продуктивності впливає на швидкість навчання агента, безпосередньо впливаючи на критика, який, у свою чергу, забезпечує зворотний зв'язок з елементом навчання. Елемент навчання перетворює зворотний зв'язок від критика та знання від елемента продуктивності на цілі навчання, які передаються генератору проблем.

Від штучного інтелекту до ігрового штучного інтелекту

Потужний штучний інтелект

Штучний інтелект – це обширна галузь, і для неї немає безпосереднього визначення, яке б стисло охоплювало всі аспекти, пов'язані з цією галуззю. Загальновизнано, що не існує загальноприйнятого визначення ШІ. Визначення терміна *«штучний» не вимагає значних зусиль,* але проблема полягає у визначенні *інтелекту*, оскільки існує багато різних і конкуруючих теорій щодо інтелекту та мети ШІ загалом.

Агент зі штучним інтелектом повинен бути здатним вирішувати проблеми якомога інтелектуальніше, навіть якщо це означає, що людина-спостерігач

не може швидко вказати на кроки, зроблені агентом для досягнення його рішень. Відповідно, агенту має бути дозволено поводитися так, щоб це перевершувало людське розуміння, за умови, що він досягає своїх цілей найкращим можливим способом. Теоретично це називається сильним ШІ.

Існують програми, в яких бажаний сильний ШІ. Однак, розглянемо гру з нульовою сумою, в якій гравець-людина намагається перемогти суперника, керованого інтелектуальним агентом. Завдання агента полягає в тому, щоб максимізувати свою корисність і, таким чином, мінімізувати корисність гравця-людини. Якщо агент вивчить усі слабкі сторони гравця-людини – і зможе скористатися ними – гра може досягти стану, в якому гравець-людина не зможе перемогти інтелектуального агента. Гравець-людина зрештою втратить інтерес до гри. Такий фактор може призвести до низької *реіграбельності* гри .

В останні роки визначення сильного ШІ змінилося та стало більш конкретним. Сильний ШІ все частіше асоціюється із загальним штучним інтелектом (ЗШІ), також відомим як ШІ, що має інтелект людського рівня.

Слабкий штучний інтелект

З іншого боку, повне вилучення ШІ з ігор також може призвести до низької реиграбельності, оскільки, як тільки гравці-люди майже завжди зможуть передбачити наступний хід свого опонента, вони, найімовірніше, втратять інтерес. Тому включення ШІ в ігри корисне лише тоді, коли воно застосовується таким чином, щоб покращити ігровий досвід.

Таким чином, хоча ШІ займається інтелектуальним вирішенням проблем, ігровий штучний інтелект (Ігровий ШІ) більше зосереджений на створенні *ілюзії* інтелекту. В іграх метою не завжди є створення NPC, які володіють інтелектом людського рівня [8]. Ігровий ШІ повинен поводитися так, щоб це було прийнятно для гравців-людей. Мета слабкого ШІ в контексті ігор полягає не в тому, щоб перевершити людські здібності, а в тому, щоб *здаватися* розумним для людського спостерігача.

Так само, як і сильний ШІ, визначення слабкого ШІ за останні роки стало більш конкретним. Зі зростанням асоціації сильного ШІ із загальним штучним

інтелектом (ЗШІ), слабкий ШІ став терміном, що використовується для опису ШІ, що спеціалізується на певних завданнях. У наступному підрозділі досліджується історія ігрового штучного інтелекту.

Рисунок 1.8 - Знімок екрана гри SpaceWar!

Комп'ютерні ігри – це відносно нова галузь. Однією з перших комп'ютерних ігор, яку, як відомо, можна було запускати на комп'ютері, була *SpaceWar!*. Її розробив Стівен Рассел для роботи на комп'ютері DEC PDP-1 у Массачусетському технологічному інституті (MIT) у 1962 році. *SpaceWar!* була грою для двох гравців, що означає, що не було NPC, якому гравець-людина міг би кинути виклик. Знімок екрана SpaceWar! дивіться на рисунку 1.8.

SpaceWar! та *Pong* — це лише дві з відомих ігор, що існували в цей період (1962–1972). Деякі з ігор включають *Adventure, Tic-Tac-Toe* та Marvin Minsky's *Tri-Pos* (Three Position Display). Ігри, обговорювані в цьому розділі, мають спільну властивість: вони не були розроблені з використанням підходів штучного інтелекту, спрямованих на те, щоб ускладнити перемогу над NPC, наприклад. Тому в них не було справжнього штучного інтелекту.

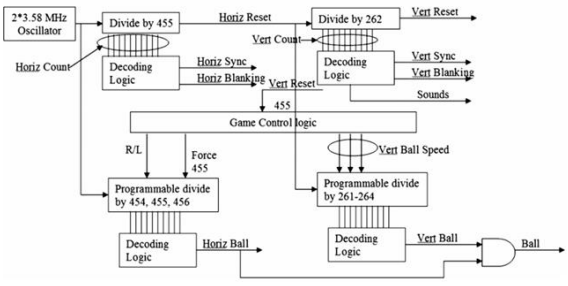

Рисунок 1.9 - Спрощена діаграма дизайну гри Pong

У наступному розділі розглядаються ігри, розроблені з використанням алгоритмів на основі шаблонів. Натхненням для використання підходів на основі шаблонів було створення враження розумніших NPC. Як показано в наступному розділі, підходи на основі шаблонів активно використовувалися в період з 1974 по 1984 рік.

1974 рік став відомим появою ігор з NPC. Такі ігри поставили перед розробниками ігор новий виклик: зробити NPC інтелектуальними, щоб ігри залишалися цікавими для гравців [9]. У першій з цих ігор використовувалися методи на основі шаблонів. Двома яскравими прикладами релізів 1974 року є *Qwak* та *Pursuit,* де завданням гравця було стріляти по цілях, що рухалися ігровим світом. Цей період був першим випадком в іграх, коли вперше було помічено використання штучного інтелекту.

У 1975 році Дейв Наттінг розробив Gunfight, першу комп'ютерну гру, в якій використовувався мікропроцесор. Щоб зробити гру менш передбачуваною та складнішою, гравця під час гри було додано об'єкти, які могли випадковим чином заважати йому. За ним пішли інші ігри, що використовували підхід, заснований на шаблонах, такі як *Space Invaders* , у якій складність могла поступово зростати завдяки різним рівням гри, та *Galaxian*.

У 1980 році компанія Namco випустила найпопулярнішу аркадну гру в історії - *Pac-Man* (див. рис. 1.10). Випуск *Pac-Man* призвів до появи NPC з різними особистостями, тобто вони мали різні моделі руху і, отже, по-різному переслідували гравця. У 1983 році шахова комп'ютерна програма, що працювала на мікрокомп'ютері, вперше в історії перемогла шахіста-людину майстер-рівня).

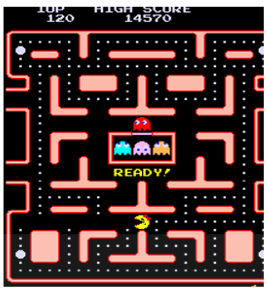

Рисунок 1.10 - Знімок екрана пані Пак-мен.

Скінченні автомати

У 1990 році з'явилися стратегічні ігри в реальному часі (RTS). *Herzog Zwei* , розроблена Technosoft, вважається першою грою в реальному часі. Компонент RTS таких ігор був реалізований переважно за допомогою кінцевих автоматів (FSM) (див. рисунок 1.11). У FSM кола представляють стани, а стрілки, що з'єднують кола, представляють переходи між станами. Як видно на рисунку 1.11, перехід викликаний умовою (наприклад, дією, що виконується над системою).

З 1996 року в індустрії комп'ютерних ігор з'явилися ігри, розроблені з використанням методів, спрямованих на те, щоб зробити NPC більш розумними та менш передбачуваними. Хоча автомати-автомати (FSM) забезпечували більшу гнучкість, ніж прості шаблони, вони все ще відображали скінченну поведінку, оскільки в автоматі-автомата існує скінченний набір можливих станів (що означає скінченний набір можливих моделей поведінки або дій). Крім того, функції переходу не змінюються [10].

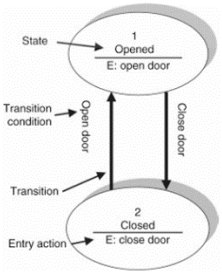

Рисунок 1.11 - Скінченний автомат

Наприклад, перехідна функція 5, застосована до умови t_c зі стану s_a, завжди призведе до стану s_b. Через це обмеження в розробці ігор застосовувалися різні методи для введення невизначеності. Щоб зрозуміти ці методи, корисно розділити типи ігрового ШІ на дві категорії: *детермінований* та *недетермінований* ігровий ШІ.

Детермінований ігровий ШІ

Детерміністичний ігровий ШІ (DGAI) – це найпоширеніший підхід у розробці ігор під час створення NPC. Детерміновані методи швидко впроваджують

та прості для розуміння, що полегшує їх підтримку, налагодження та вдосконалення. FSM – це приклад детерміністичного підходу до ігрового ШІ.

Недоліком DGAI є те, що гравцеві-людині його легко перехитрити. Щоб уникнути цього, розробники ігор часто писали значно великі скрипти, маючи на меті передбачити всі можливі сценарії в грі. Однак цього виявилося недостатньо. Ця слабкість DGAI зрештою призвела до появи недетермінованого ігрового штучного інтелекту (NDGAI).

Недетермінована гра зі штучним інтелектом

NDGAI мав на меті створити непередбачуваних та навчальних ігрових агентів [11]. Відповідно, розробники NDGAI не намагалися відстежувати всі можливі стани гри, а натомість зосередилися на розробці ігрових агентів зі здатністю адаптуватися до неочікуваних змін. Більше того, NDGAI мав потенціал для демонстрації емерджентної поведінки.

Головною проблемою розробки NDGAI була складність налагодження, оскільки програміст не міг розробити тест, який би перевіряв усі можливі дії, які агент міг би виконати, враховуючи послідовність станів. Це обмеження робило створення особливо складним.

NDGAI для комерційного використання, оскільки було складно повністю передбачити, як NPC поводяться протягом життя гри.

Гра «Чорно-біле» (див. рис. 1.12), розроблена Lionhead Studios, була випущена у березні 2001 року. Центральною темою гри було те, що гравець мав діяти як бог і навчати істоту виконувати завдання в регіонах, де гравець не міг діяти сам. Гравець міг надавати істоті зворотний зв'язок після її дії, вдаряючи її або погладжуючи, щоб сигналізувати про корисність дії істоти. Таким чином, істота існувала для того, щоб допомогти гравцеві реалізувати план.

У грі було два типи інтелектуальних агентів. Першим були жителі громади. Причиною створення цих агентів було ускладнення для гравця спроби завоювати прихильність селян. Більше того, всім селом керував інтелектуальний агент для досягнення певної злагодженості. Таким чином, кожен житель села був простодушною сутністю, яка виконувала лише дії, спрямовані на благо села.

Другим типом інтелектуального агента була істота, яку гравець використовував для виконання свого плану. Істота була створена для вираження людської поведінки та надання цінності. Для розвитку інтелекту істоти використовувалися три основні методи. Переконання істоти щодо дискретних об'єктів кодувалися як символічні пари атрибут-значення, де для кожного завдання кожен атрибут мав значення, що описувало його силу (або вплив).

Ця техніка, у поєднанні зі штучним інтелектом на основі правил, надавала істоті необхідну інформацію про навколишні об'єкти. Переконання істоти щодо загальних типів об'єктів були представлені у вигляді дерев рішень. Бажання істоти були представлені нейронними мережами. Поведінку NPC (істоти) у грі Black & White можна класифікувати за допомогою ШІ як шаблон проектування «Стажер», оскільки в такій грі гравець виконує дії, які навчають ШІ виконувати основні обов'язки.

Сучасний ігровий штучний інтелект (2002 – дотепер)

Спираючись на часову шкалу, коротко обговорюються деякі ігри, реалізовані з 2002 року до теперішнього часу. Це обговорення аж ніяк не є вичерпним, оскільки щороку випускається відносно велика кількість ігор. Натомість, в обговоренні розглядаються різні методи, що застосовувалися до ігрового ШІ в період з 2002 року до теперішнього часу.

Галактична гонка озброєнь (2010)

Гра «Галактичні перегони озброєнь» (випущена Evolutionary Group у 2010 році) була створена для демонстрації створення контенту на ігровій онлайн-платформі. Гра являє собою космічний шутер (див. рис. 1.12), де гравець може використовувати зброю різних систем частинок для стрільби по ворогах. Цікава ідея щодо ігрового штучного інтелекту, застосована до «Галактичних перегонів озброєнь», полягає в онлайн-еволюції зброї, яку використовує гравець.

Рисунок 1.12 - Знімок екрана фільму «Галактична гонка озброєнь»

Для успішного досягнення еволюції зброї на основі систем частинок, у грі використовувалися інтерактивні еволюційні обчислення (IEC), де функція придатності була замінена людським судженням. IEC, у контексті Галактичної гонки озброєнь, було досягнуто шляхом використання нейроеволюції доповнюючих топологій (NEAT) для еволюції мереж, що створюють композиційні шаблони (CPPN). Отриманий алгоритм був названий Гастінгсом, нейроеволюцією доповнюючих топологій генерації контенту (cgNEAT).

Основна ідея розвитку зброї на основі системи частинок полягала в тому, що система ігрового штучного інтелекту постійно генеруватиме нові типи зброї протягом гри на основі уподобань гравця [12]. Використання гравця як функції фізичної підготовки також означало, що система ігрового штучного інтелекту мала більше шансів створювати зброю, яка сподобається гравцеві.

Класифікують підхід, який використовувався для розробки «Галактичної перегони озброєнь», як шаблон проектування ігрового ШІ під назвою «ШІ можна редагувати». Ідея такого шаблону проектування полягає в тому, що гравець постійно змінює елементи системи ігрового ШІ, які є центральними для ігрового процесу. У контексті «Галактичної перегони озброєнь» гравець постійно змінював колекцію зброї системи частинок; і це стало можливим завдяки IEC.

Чужий: Ізоляція (2014)

Розроблена Creative Assembly та випущена в жовтні 2014 року, Alien: Isolation - це гра жахів на виживання від першої особи, в якій NPC - це інопланетянин, який виконує роль лиходія (див. рис. 1.13). Гра заснована на фільмі «Чужий» 1979 року. Головним завданням NPC у грі було постійно дратувати гравця-

людину. Крім того, основною метою NPC було не перехитрити гравця (і зрештою перемогти його), а створити для гравця досвід постійної провокації.

Агент NPC контролювався двома окремими системами: ШІ- директором (також званим «макро-ШІ») та ШІ-інопланетянина (також званим «мікро-ШІ»). ШІ-директор відповідав за відстеження індикатора загрози (міри тиску, що чиниться NPC на гравця в будь-який момент) [13]. Тиск у цьому контексті стосується того, наскільки близько NPC знаходиться до гравця.

Поведінка NPC закодована в дереві поведінки, яке містить понад 100 вузлів. 30 верхніх вузлів дерева відповідають за вибір поведінки, якої дотримуватиметься NPC. Решта вузлів – це підповедінки, що виконуються на основі шаблону поведінки, обраного на вершині дерева.

Рисунок 1.13 - Знімок екрана фільму «Чужий: Ізоляція »

Не всі вузли в дереві поведінки NPC доступні. Деякі вузли спочатку «заблоковані» або до них немає доступу. У міру проходження гри та виконання певних умов недоступні підповедінки стають доступними, створюючи ілюзію того, що NPC навчається на досвіді.

Отримуючи завдання від штучного інтелекту директора, NPC використовує алгоритм пошуку шляху на основі евристики для навігації до певних місць. Надані евристики також використовуються для визначення сили кожного з органів чуття NPC (наприклад, точності, з якою NPC виявляє кроки гравця).

Поведінку NPC у грі можна пояснити за допомогою шаблону проектування *«ШІ як лиходій»*. У цьому шаблоні проектування гравець змушений враховувати поточний стан NPC, таким чином послідовно міркуючи про швидкість

його вдосконалення після багаторазової гри. Постійні міркування про NPC призводять до того, що гравець бере участь у навчанні/розвитку ШІ [14].

Шпигунська вечірка

Гра SpyParty була створена та розроблена Крісом Хекером. Гру вперше представили на Конференції розробників ігор (GDC) у 2009 році, а потім вона була комерційно випущена у 2018 році. Знімок екрана гри показано на рисунку 1.14. SpyParty - це асиметрична багатокористувацька гра, де завдання одного гравця-людини полягає в тому, щоб діяти як шпигун і зливатися з іншими NPC у соціальному середовищі, непомітно виконуючи певні обов'язки. Мета шпигуна - не бути виявленим і не застреленим снайпером (іншим гравцем-людиною).

Цікава ідея SpyParty полягає в тому, що гравець, який виконує роль шпигуна, повинен намагатися поводитися так, щоб не відрізняти себе від NPC (які контролюються ігровими агентами). Таке завдання змушує обох гравців (шпигуна та снайпера) глибоко задуматися про ігрових агентів, які контролюють NPC, щоб зрозуміти їх та виявити будь-які приховані моделі поведінки.

Рисунок 1.14 - Знімок екрана SpyParty

Замість того, щоб намагатися створити значно непередбачуванішу поведінку агентів, більшість ігор реалізують ігрових агентів за допомогою алгоритмів, які відтворюють шаблонну поведінку. Те саме стосується і SpyParty, що є важливим, оскільки для того, щоб гра була приємною, гравці-люди повинні мати можливість вивчати моделі поведінки ігрових агентів. Класифікували дизайн SpyParty як шаблон проектування ШІ як рольової моделі, оскільки один з гравців-людей (шпигун) повинен намагатися відображати поведінку ігрових агентів.

У підрозділі обговорювалася еволюція ігрового ШІ. З обговорюваних ігор можна зазначити, що застосування ШІ в іграх було поступовим розвитком. Більше того, для ігор типово мати NPC, а також гравців-людей. Основна увага приділяється ігровим агентам та розглядається їхня структура [15].

1.2 Концепції адаптивного та динамічного ШІ в іграх

Люди мають властивість бути унікальними. Тому, оскільки різні люди грають в ігри, з цього випливає, що кожен гравець взаємодіятиме з грою унікально. Таким чином, різні гравці використовуватимуть унікальні стратегії, щоб виграти гру, а це означає, що персонажі NPC по-різному сприйматимуть кожного гравця. Це явище є основою для остаточної реалізації адаптивного ігрового штучного інтелекту.

Ігровий ШІ, який може адаптуватися залежно від ігрової стратегії гравця, має більше шансів забезпечити індивідуальний та захопливий досвід для гравців. Порівняно з ігровою системою ШІ, яка поводиться однаково, не враховуючи індивідуальність кожного гравця, це додає цінності комп'ютерній грі. Більше того, розважальна цінність комп'ютерної гри прямо пропорційна її якості.

Можна стверджувати, що якщо гравцеві-людині потрібна більша розвага від гри, він також має можливість кинути виклик іншим гравцям-людям в Інтернеті. Однак можуть бути випадки, коли користувач не має підключення до Інтернету або просто має настрій грати самостійно. Крім того, гравці-люди можуть бажати підвищити свою майстерність у відеоіграх з таких причин, як кіберспорт.

Адаптивний ігровий ШІ - це ігровий ШІ, який може успішно адаптуватися до змінних умов. Це підхід, за якого агенти NPC можуть навчатися та адаптуватися до того, як ведеться гра [16]. Таким чином, адаптивний ігровий ШІ надає спосіб пом'якшити недоліки традиційного ігрового ШІ, дозволяючи NPC ефективно реагувати на стратегії гравців-людей.

Ігровий ШІ можна зробити адаптивним за допомогою онлайн-навчання. Spronck, Ponsen, Sprinkhuizen-Kuyper та Postma визначили, що існує два типи

методів онлайн-машинного навчання для ігрового ШІ: (1) методи, керовані людиною, та (2) методи , керовані комп'ютером. У контексті онлайн-машинного навчання, керованого людиною, гравець надає миттєвий зворотний зв'язок щодо дій NPC під час гри, щоб вказати, чи є вони бажаними. Цей метод подібний до шаблону проектування «ШІ як учень» , який було застосовано до чорно-білого.

Комп'ютерно-кероване онлайн-навчання змінює стратегії (або поведінку) NPC, оцінюючи вплив, який NPC мають на гру під час її проведення. Цей тип онлайн-навчання має обмежене застосування в іграх через ризик того, що після випуску гри NPC можуть безповоротно навчитися посередньої поведінки. Тому комп'ютерно-кероване онлайн-навчання зазвичай застосовується таким чином, що NPC можуть вивчити лише невелику кількість усіх параметрів.

Алгоритм онлайн-навчання для ігрового ШІ повинен відповідати чотирьом обчислювальним та чотирьом функціональним вимогам: (1) швидкість , (2) ефективність , (3) стійкість та (4) результативність . Функціональні вимоги - це (1) ясність , (2) різноманітність , (3) узгодженість та (4) масштабованість . Автори також стверджували, що для того, щоб метод онлайн-навчання працював добре, процес навчання повинен базуватися на знаннях, специфічних для предметної області.

Штучна дурість

Розважальна цінність гри не обов'язково залежить від того, наскільки розумні NPC. Це також не означає, що персонажів NPC слід програмувати таким чином, щоб вони були вразливими, оскільки перебільшення цього також зменшить розважальну цінність гри. Тому підхід до проектування NPC, які навмисно допускають свої помилки, є ключовим для адаптивного ігрового штучного інтелекту [17]. Однак це вимагає точного налаштування агентів NPC таким чином, щоб уникнути нерозумного вигляду агентів NPC, водночас гарантуючи, що їх не буде надто складно чи надто легко перемогти з точки зору гравця. Цей метод називається штучною дурістю.

Щоб зробити агента штучно дурним у певному завданні, на нього навмисно накладаються деякі обмеження, щоб він виконував завдання на рівні, з яким

люди можуть співвідносити себе. Також було зазначено, що комп'ютерні програми, які навмисно здійснюють спрощену поведінку, як правило, показують кращі результати в тестах Тюрінга. Прикладом таких програм є ALICE (Artificial Linguistic Internet Computer Entity) та ELIZA.

Тому штучну дурість не обов'язково легко реалізувати, оскільки для уникнення суперінтелекту потрібен суперінтелект. Наприклад, щоб викрити комп'ютер під час тесту Тюрінга, можна просто попросити його розв'язати список складних математичних задач.

Комп'ютер знаходив би рішення таких задач з дуже високою точністю, тоді як людина, ймовірно, мала б з цим труднощі. Тому, щоб обдурити користувача, комп'ютер міг би просто надавати неправильні відповіді на деякі задачі безпідставним чином, що вимагає значного рівня інтелекту.

Тому здатність ефективно впроваджувати штучну дурість є важливим фактором, який слід враховувати в контексті ЗШІ, оскільки це безпосередньо впливає на розважальну цінність гри [18]. Запропонував конкретні дії, які NPC можуть виконувати для посилення штучної дурості в іграх від першої особи (FPS), такі як рух перед пострілом, погане прицілювання, видимість або промах з першого разу під час стрільби.

Моделювання опонента

Моделювання суперника коротко описується як спосіб представлення того, що відомо про суперника під час гри. Таким чином, моделювання суперника просто стосується побудови моделі, яка представляє стратегію суперника в загальній формі. Основна ідея полягає в тому, що, хоча різні стратегії можуть демонструвати незначні відмінності одна від одної, можна вивести відносно невелику кількість загальних стратегій, згідно з якими можна згрупувати або класифікувати більш детальні стратегії. Ці загальні стратегії називаються моделями гравців. Тому моделювання суперника є проблемою класифікації.

Під час гри ігрова статистика, зібрана від гравця, зіставляється з різними моделями гравців. Модель гравця, яка найкраще нагадує гравця, потім використовується для його представлення. Визначивши відповідну модель для гравця,

NPC отримує інформацію, необхідну для розробки контрстратегії проти нього. На рисунку 1.15 показано таксономію моделювання поведінки гравців, яку можна використовувати як підхід до створення моделі (або профілю) гравця.

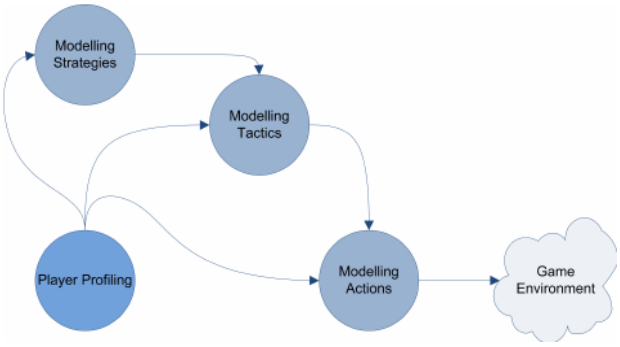

Рисунок 1.15 - Таксономія моделювання поведінки гравців

Здатність успішно класифікувати опонента у відповідну модель гравця та розробити контрстратегію може дозволити NPC бути адаптивними. Крім того, моделювання опонента допомагає використовувати слабкі сторони в стратегіях гравців, оскільки жоден гравець не є ідеальним. Однак головною метою моделювання опонента у відеоіграх є підвищення рівня розваги, а не збільшення сили чи конкурентоспроможності NPC.

Моделювання опонента може застосовуватися у випадках, коли NPC є компаньйоном або фактичним суперником. Як компаньйон, роль NPC полягає в тому, щоб поводитися відповідно до очікувань гравця-людини. У таких іграх, де NPC є компаньйоном гравця-людини, NPC повинен бути ефективним у класифікації бажань/стратегії гравця-людини, щоб позитивно вплинути на розважальну цінність гри.

У випадку, коли NPC застосовує моделювання супротивника, його головною метою є підлаштування навичок гравця для взаємодії з ним на відповідному рівні складності. Для NPC вкрай важливо ефективно адаптуватися до навичок гравця, щоб гра була цікавою.

Підходи до моделювання опонентів у літературі

Запропоновано універсальний та адаптивний підхід до моделювання

суперника для стратегічних ігор у реальному часі. Цей підхід був повністю автоматизованим та вивчав лише важливі та інформативні характеристики гравця. Їхній підхід складався з онлайн- та офлайн-фаз. Метою офлайн-фази була розробка моделей суперника, одночасно визначаючи, який тип ігрової інформації потрібно зібрати про гравця та включити до моделі.

Для досягнення цієї мети підхід передбачав отримання та зберігання зразків ігрового процесу з Інтернету в центральному сховищі даних. Збережені зразки ігрового процесу потім використовувалися для створення моделей суперників офлайн [19]. Моделі суперників зберігалися таким чином, щоб до них був легкий доступ під час гри, коли потрібно було провести класифікацію.

Підхід на основі прецедентів (CBR) використовувався в онлайн-фазі для класифікації гравця відповідно до збережених моделей опонентів. Це допомагало NPC адаптуватися до гравця, оскільки після класифікації гравця за певною моделлю опонента NPC міг працювати над визначенням оптимальної стратегії.

Використано метод опорних векторів (SVM) для моделювання суперника з метою створення адаптивної команди в американському футболі. Запропонована авторами система використовувала послідовність спостережень для успішної класифікації захисної гри, щоб команда могла розумно реагувати атакуючою грою.

Проблема, з якою зіткнулися автори, полягала в необхідності виконувати класифікацію під час гри, використовуючи послідовності спостережень (тобто історію сприйняття). Оскільки послідовність спостережень постійно зростає, доки не буде досягнуто кінця гри, це призвело б до того, що обраний класифікатор мав би обробляти вхідні вектори різної довжини.

Таким чином, було обрано підхід, що полягав у використанні кількох SVM замість однієї, де кожна SVM була навчена обробляти вхідні вектори (послідовності спостережень) певної довжини. Для класифікації гри команди під час гри використовувалася SVM, що відповідає поточному часовому кроку, і вхідний вектор (послідовність спостережень) відповідної довжини (правильна кількість послідовностей) подавався до класифікатора.

Використано теоретико-ігровий підхід до моделювання опонента для великих ігор з недосконалою інформацією. Цей підхід спостерігав частоту дій опонента та формував модель шляхом інтеграції спостережень із рівноважною стратегією, яка була обчислена заздалегідь. Як контрстратегії, так і моделі гравців розробляються в режимі реального часу.

Для досягнення цієї мети автори запропонували алгоритм, відомий як найкраща відповідь на основі відхилення (DBBR). Алгоритм спостерігав за частотою дій опонента та побудував модель для представлення стратегії гравця. Сформована модель опонента потім використовувалася NPC для розробки контрстратегії проти гравця. Для ефективної інтерпретації стратегії гравця алгоритм визначав частоту дій гравця, яка представляла відхилення від рівноважної стратегії (або частоту дій).

Приклади роботи, виконаної з моделювання суперника, показують, що цього можна досягти різними способами. Більше того, проблема, до якої застосовується моделювання суперника, визначає, який підхід буде використано [20]. Однак моделювання суперника не є обов'язковим для кожної гри. Таким чином, у наступному розділі обговорюються обмеження, з якими стикається моделювання суперника у відеоіграх.

Проблеми, з якими стикається моделювання опонентів у відеоіграх

У відеоіграх моделі супротивників мають розроблятися в режимі реального часу, що означає, що існує сильне обмеження на кількість часу, який NPC витрачає на моделювання поведінки гравця. Виконання класифікації в режимі реального часу є складним завданням, оскільки воно має відбуватися одночасно з рендерингом графіки, наприклад.

Лише 20% обчислювальних ресурсів зазвичай виділяється на розробку ігрового штучного інтелекту. Ще однією проблемою є те, що середовище у відеоіграх часто є репрезентативним для реального світу, а це означає, що воно може бути значно складним. Зрештою, середовище у відеоіграх не є повністю спостережуваним.

1.3 Методи балансування ігрових механік та складності гри

Балансування складності гри (GDB) стосується автоматичної адаптації виклику, який гра ставить перед гравцем. Динамічне балансування складності (DDB) повинно відповідати наступним трьом вимогам: (1) гра повинна швидко класифікуватися та адаптуватися до рівня майстерності гравця, (2) швидко відстежувати розвиток, а також провали у продуктивності гравця та (3) бути правдоподібною під час адаптації, щоб гравець не підозрював про будь-яке масштабування складності, яке може відбуватися «під капотом».

Парна гра (гра, рівень складності якої відповідає майстерності гравця) зазвичай цікавіша для гравця, ніж та, яку занадто складно або занадто легко виграти. DDB зазвичай прагне досягти парної гри з гравцем-людиною, а не перемогти його.

Важливо виміряти, наскільки складним завданням є гра для користувача для досягнення DDB. Цей показник можна визначити за допомогою так званої *функції складності*.

Функція виклику є, по суті, евристичною функцією використовується для визначення того, наскільки складною є гра для користувача [21]. Евристики можна отримати за допомогою будь-якої метрики, яка використовується для характеристики продуктивності гравця або впливу на рахунок гри.

У змагальних іграх і – що найважливіше – в іграх, де гравець-людина може застосувати будь-яку стратегію в будь-який момент часу для перемоги, вимірювання майстерності гравця можна ефективно виконати шляхом побудови моделі *гравця* . У наступному розділі обговорюється цей критичний аспект змагальних ігор.

Теорію ігор широко визнають такою, що її запровадили у 1941 році Джон фон Нейман та Оскар Моргенштерн у 1944 році «Теорія ігор та економічна поведінка». Хоча варто зазначити, що у 18 столітті Джеймс Волдегрейв запропонував перше відоме рішення MINIMAX (яке називалося рівновагою стратегії MINIMAX) для гри, в яку грають два суб'єкти.

Більше того, у 19 столітті Огюстен Курно обговорював обмежену версію рівноваги Неша «Дослідження математичних принципів теорії багатства». Таким чином, теорія ігор спочатку була представлена як концепція, заснована на економіці, а не на штучному інтелекті. Ідея теорії ігор полягає в пошуку способів дослідження ігор для виявлення оптимальних стратегій. Крім того, теорія ігор прагне аналізувати людську поведінку так, ніби самі люди є учасниками гри.

Можна змоделювати безліч реальних сценаріїв як ігри. Щоразу, коли існує ситуація, в якій раціональні люди можуть діяти лише за суворими правилами – і за кожну дію, яку можна виконати, є винагорода – таку ситуацію можна розглядати як гру [22]. Переговори є прикладом такої ситуації. Дилема в'язня– типовий приклад, що використовується в інформатиці для демонстрації того, як теорія ігор використовується для дослідження ігор.

1.4 Висновок по розділу

У першому розділі було розглянуто теоретичні основи застосування штучного інтелекту в ігрових системах. Проаналізовано основи штучного інтелекту в іграх та принципи роботи ігрових агентів, що забезпечують адаптивну поведінку ігрового середовища. Досліджено концепції адаптивного та динамічного ШІ, які дозволяють змінювати параметри гри відповідно до стилю та рівня підготовки гравця. Також розглянуто методи балансування ігрових механік і складності гри для забезпечення комфортного та цікавого ігрового процесу. Отримані результати формують теоретичну основу для подальшого дослідження методів автоматичного балансування ігор із використанням ШІ

РОЗДІЛ 2. МЕТОДИ ТА МОДЕЛІ АВТОМАТИЧНОГО БАЛАНСУВАННЯ ІГОР З ВИКОРИСТАННЯМ ШІ

2.1 Методи машинного навчання для адаптивного балансування гри

Агента, який навчається, можна розрізнити, оцінюючи його продуктивність з часом, оскільки він неминуче змінюється зі зміною обставин. Більш конкретно, агент, що навчається, покращує свою поведінку в майбутніх завданнях завдяки спостереженням за своїм оточенням.

Також вкрай важливо встановити фундаментальний мотив спроби змусити агентів навчатися [23]. Зрештою, ми могли б просто жорстко запрограмувати всю поведінку, яку ми хочемо, щоб агент демонстрував за всіх можливих умов. Хоча це може здаватися привабливим варіантом, він має суттєві обмеження:

- Розробники агента не можуть передбачити всі можливі ситуації, в яких він може опинитися .
- Дизайнери агента також не можуть передбачити всі зміни навколишнього середовища , які відбуваються з часом.
- Зрештою, бувають випадки, коли люди можуть знати, як виглядає ідеальне рішення , але вони можуть не знати, як самостійно його запрограмувати.

Три вищезазначені обмеження є одними з багатьох причин, чому галузь навчання розвивається останніми роками. Перспектива того, що програмісти зможуть змусити програму виконувати певне завдання без необхідності надавати явні інструкції, є багатообіцяючою щодо майбутнього машинного навчання.

Хоча люди добре справляються з класифікацією зображень знайомих людей, запрограмувати агента на виконання такого ж завдання - справа нетривіальна. Тому було б корисно навчити комп’ютер виконувати таке завдання, не вказуючи йому прямо, як це робити.

Підходи до машинного навчання

Першим досліджуваним підходом є навчання деревом рішень (або

індукція), один з найпростіших та найширше використовуваних підходів до машинного навчання.

Лінійні моделі базуються на лінійних функціях від неперервнозначних вхідних даних і використовуються вже століттями. Найпростішою лінійною моделлю є одновимірна лінійна регресія і вона розглядається в наступному підрозділі.

Одновимірну лінійну регресію (також відому як проста лінійна регресія) можна розглядати як апроксимацію прямої лінії (див. рівняння 2.1). Одновимірна лінійна функція генерує y з одного прикладу x і має вигляд:

$$y = xw_0 + w_1 \tag{2.1}$$

Багатовимірна лінійна регресія. Якщо вибірка даних (або приклад) більше не є одним значенням, а вектором значень, задача перетворюється на багатовимірну лінійну регресію. У цьому випадку ψ є функцією більш ніж однієї змінної. Тому ψ тепер можна розглядати як скалярний добуток прикладного вектора **x** та вагового вектора **y** . Багатовимірна лінійна функція тоді стає такою:

$$y = w_0x_0 + w_1x_1 + \cdots + w_nx_n \tag{2.2}$$

У цьому випадку пошук більше не полягає в пошуку однієї прямої, яка найкраще відповідає нашим даним у двовимірному просторі. Наш алгоритм тепер має знайти гіперплощину, яка найкраще відповідає точкам даних у просторах вищих вимірів (див. рівняння 2.2).

Основна відмінність між багатовимірною та одновимірною лінійною регресією полягає в тому, що при багатовимірній лінійній регресії можливе перенавчання . Оскільки вхідні дані є багатовимірними в багатовимірній лінійній регресії, один нерелевантний вимір може здаватися корисним через випадковий шум або флуктуації.

Лінійні моделі також можна використовувати для завдань класифікації

шляхом знаходження межі прийняття рішення : прямої лінії (або гіперплощини, у випадку багатовимірних даних), яка служить роздільником між двома класами прикладів [24]. Ця роздільна лінія або межа прийняття рішення також називається лінійним роздільником.

Щоб лінійна модель ефективно розділяла дані, що належать до двох класів, ці дані повинні бути лінійно роздільними. Нелінійні класифікатори можна використовувати для підбору даних, які не є лінійно роздільними. Методи опорних векторів (SVM). Мережі,є прикладами нелінійних класифікаторів. У наступному підрозділі штучні нейронні мережі розглядаються детальніше.

Штучні нейронні мережі. ШНМ натхненні структурою кори головного мозку людини, і це очевидно з того, що ШНМ дозволяють комп'ютерам виконувати завдання, які легко виконувати людям, але важко виконувати звичайним комп'ютерам. Людський мозок складається приблизно зі 100 мільярдів взаємопов'язаних нейронів. Нейрон є фундаментальною одиницею центральної нервової системи, і кожен нейрон має від 100 до 10000 зв'язків з іншими нейронами. Нейрон отримує вхідні сигнали від інших нейронів через дендрити, де кожен дендрит, що називається синаптичним контактом. вхідні сигнали фенілу, отримані нейроном, інтегруються, і якщо інтегрований сигнал достатньо сильний, нейрон спрацьовує, генеруючи вихідний сигнал. Згенерований вихідний сигнал надсилається вздовж аксона, який далі розділяється таз'єднується з тисячами дендритів інших нейронів через синаптичні з'єднання.

Розмір сигналу в кожному дендриті залежить від синаптичної сили зв'язку між аксоном і дендритом (синаптичним контактом). Провідна сила кожного синаптичного контакту в нейронній мережі змінюється в міру навчання мозку . Для цього синапси називають фундаментальними одиницями пам'яті мозку .

Застосування концепцій нейронних мереж до обчислень. Комп'ютерна реалізація нейронної мережі називається штучною нейронною мережею , яка складається зі штучних нейронів. Штучний нейрон – це вузол обробки, який приймає кілька вхідних даних (сигнали через «дендрити») та підсумовує їх в один об'єднаний сигнал. Об'єднаний сигнал проходить через функцію активації, вихід якої

використовується для визначення того, чи спрацьовує нейрон, чи ні. Нейрон спрацьовує лише тоді, коли вихід передавальної функції перевищує заздалегідь визначене порогове значення.

Функція активації також може бути неперервною. Наприклад, такі функції, як сигмоподібна, гіперболічний тангенс та арктангенс, використовувалися в літературі. У нещодавніх роботах також використовувалася функція *випрямленої лінійної одиниці* (ReLU) як функція активації.

Вихідний сигнал нейрона на підключений нейрон посилюється вагою зв'язку між двома нейронами. Ця ідея подібна до того, як сила вихідного сигналу аксона на дендрит визначається синаптичною силою. Нейрони в штучній нейронній мережі зазвичай організовані в шари. Перший шар – це вхідний шар, на який надаються вхідні дані. За вхідним шаром йдуть приховані шари [25].

Кількість прихованих шарів у ШНМ вибирається розробником на основі конкретних потреб. Останній шар мережі – це вихідний шар, де кожен вихідний нейрон представляє клас. Щоб ШНМ зробила прогноз, вхідні дані передаються з вхідного шару через приховані шари, доки не досягнуть вихідного шару. Вихідний нейрон, який спрацьовує, вказує на клас, який «передбачила» мережа. Однак ШНМ можна налаштувати таким чином, щоб усі вихідні нейрони спрацьовували з ймовірністю, де сума ймовірностей дорівнює 1 (це називається *активацією SoftMax*).

Як і в інших моделях, метою є мінімізація похибки прогнозування ШНМ, що може бути досягнуто шляхом безперервної оптимізації вагових значень, доки ШНМ не почне поводитися бажаним чином. Оптимізація зазвичай виконується за допомогою *правила навчання перцептрона* або *градієнтного спуску* для мінімізації функції втрат ШНМ. Деякими прикладами ШНМ є радіально-базисні мережі (RBN), згорткові нейронні мережі (CNN) та генеративно-змагальні мережі (GAN).

Непараметричні моделі. Непараметричну модель не можна визначити фіксованим набором параметрів. Наприклад, усі моделі, які ми розглядали досі, намагаються вивчити функцію гіпотези $h(x)$, вектор ваг якої призведе до того, що

вона здебільшого зробить правильний прогноз (у випадку задач класифікації). У цьому розділі розглядаються деякі непараметричні моделі, що існують у літературі.

Навчання деревом рішень. Дерево рішень – це інструмент, який використовується для аналізу доступних варіантів з метою визначення можливих кроків, які необхідно зробити під час процесу. Дерева рішень використовуються в інформатиці, щоб допомогти комп'ютерним програмам приймати рішення за певних умов та визначеного способу їх оцінки, щоб остаточно прийняти рішення.

З огляду на їхню деревоподібну структуру, набір правил умов-дій, закодованих у дереві рішень, можна зручно візуалізувати для зручності читання людиною [26]. Кожна точка прийняття рішення вздовж дерева називається вузлом. Вузол може мати кілька *дочірніх* вузлів, але не може мати більше одного батьківського вузла. Структура даних, в якій вузли можуть мати кілька *батьківських вузлів,* вже не є деревом, а графом.

Найважливішою властивістю дерева рішень є те, що кожне рішення призводить до наступного кроку в процедурі прийняття рішень, доки не буде прийнято остаточне рішення. Рисунок 2.1 ілюструє дерево рішень. Кореневий вузол *(Погода?)* задає питання про погоду. Від кореневого вузла відходять три гілки, кожна з яких представляє можливий стан погоди *(Сонце, Хмара* або *Дощ).* Який дочірній вузол буде оцінюватися *(Час?, Голод?* або *Автобус),* залежить від стану погоди (тобто, якщо сонячно, наступним кроком буде оцінка *Часу?).* Процес продовжується, доки не буде прийнято остаточне рішення (тобто *Пішки* чи Автобус).

Навчання дерев рішень (або індукція) здебільшого використовується у випадках, коли вичерпна інформація не є одразу доступною (або необхідною) для прийняття рішень програмою. Дерева рішень також можна використовувати для виконання завдань класифікації. У цьому випадку вхідні дані, що надаються дереву, спочатку перевіряються за допомогою кореневого вузла. Результат перевірки визначає, вздовж якого ребра вхідні дані будуть відправлені для подальших перевірок [27]. Цей процес продовжується до досягнення кінцевого вузла.

Кінцевий вузол містить клас, який буде остаточно призначено вхідним даним.

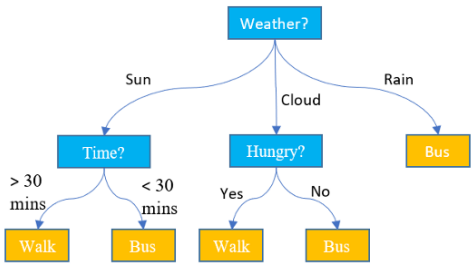

Рисунок 2.1 - Дерево рішень 17 років

Для виконання індукції дерева рішень можна використовувати кілька алгоритмів. Деякі з відомих алгоритмів - це ID3 (Ітеративний дихотомізатор 3), Випадковий ліс та CART (Дерево класифікації та регресії). Однією з найважливіших переваг дерев рішень є те, що вони *інтерпретуються* , і це означає, що можна зручно простежити кроки, які виконало дерево рішень для досягнення рішення (або прогнозу). Це може бути проблемою, пов'язаною з іншими класифікаторами, такими як штучні нейронні мережі (ANN), класифікатори k-найближчих сусідів (KLNN) та методи опорних векторів (SVM). Відсутність інтерпретованості призводить до того, що більшість класифікаторів розглядаються як чорні ящики, і важко виконувати налагодження, коли класифікатор поводиться неочікувано.

Дерева рішень також вимагають мінімальної підготовки даних порівняно з іншими моделями машинного навчання. Вони також часто не потребують нормалізації даних. Більше того, дерева рішень можуть обробляти як категоріальні, так і числові дані та задачі з кількома виходами. Зрештою, вартість використання дерева рішень логарифмічна до кількості точок даних, які були використані для навчання дерева.

Навчання за методом найближчого сусіда належить до сімейства методів навчання на основі екземплярів. Такі методи зберігають весь набір навчальних даних для подальшого використання. Коли моделі надається приклад, вона виконує пошук. Якщо подібний приклад існує в навчальному наборі, модель

повертає відповідний прогноз.

Найчистішою формою методів на основі екземплярів є пошук у таблиці: створюється таблиця для зберігання набору даних. Коли надається приклад, алгоритм виконує пошук і повертає відповідний прогноз, якщо пошук був успішним [28]. Якщо в таблиці пошуку не знайдено подібного прикладу, повертається випадкове значення. Метод пошуку в таблиці погано узагальнює, оскільки для того, щоб модель зробила правильний прогноз на основі прикладу, подібний приклад повинен існувати в таблиці.

Методи найближчих сусідів базуються на передумові, що подібні приклади зазвичай належать до одного класу. Тому нам потрібно знайти правильний спосіб вимірювання подібності та визначення оптимальної кількості сусідів з найбільшою подібністю до прикладу. Найпоширенішим методом найближчих сусідів *є k-найближчих сусідів* (kNN) (рис. 2.2), де *k* – кількість прикладів з найбільшою подібністю до наданого прикладу. Подібність між двома векторами можна виміряти за допомогою метрики відстані, такої як, наприклад, евклідова відстань.

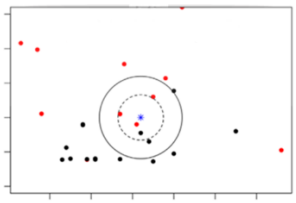

Рисунок 2.2 - Зображення алгоритму k-найближчих сусідів.

Було зазначено, чим непараметричні моделі відрізняються від параметричних. Наприклад, під час використання дерева рішень не потрібно відстежувати конкретні параметри як засіб для виконання прогнозів. Те саме можна сказати і про алгоритм kNN. Єдине значення, яке слід вказати для kNN, це k , яке вказує на кількість сусідів, яких необхідно розглянути, але немає набору параметрів, які модель повинна використовувати для вивчення функції для виконання прогнозів.

Еволюційні моделі

Штучні імунні системи. Можна стверджувати, що штучні імунні системи (ШІС) – або імунологічно натхненні алгоритми – можуть бути використані для створення систем, що навчаються. ШІС базується на біологічній імунній системі (БІС), яка може формувати адаптивні імунні відповіді. Важливим аспектом адаптивної імунної відповіді є те, що вона може *запам'ятовувати* свої зустрічі з антигенами. Як результат, вона може швидко реагувати на антигени, які спостерігалися раніше.

AIS намагається змоделювати цю особливість BIS. Здатність BIS формувати адаптивні імунні відповіді ще не повністю вивчена. Однак, для моделювання деяких теорій про BIS було впроваджено різні алгоритми. Деякі з цих алгоритмів - це алгоритми негативного відбору (NSA), алгоритми клонального відбору (CSA) та штучні імунні мережі (AIN).

Навчання з підкріпленням (НП) – це підхід до машинного навчання, заснований на концепції винагород і покарань. Розглянемо, наприклад, класифікацію: щоб навчити класифікуючого агента за допомогою методів, які ми розглядали досі, потрібно використовувати позначені приклади. Мітки допомагають агенту одразу зрозуміти, чи він на правильному шляху, чи ні.

Крім того, при розгляді еволюційних моделей: функція придатності може бути використана для визначення того, які агенти (індивіди) добре працювали (мали високий рівень придатності). Після цього агенти з бажаними показниками придатності будуть використані для відтворення більшої кількості агентів, які мають схожі риси. Це означає, що агенти не навчаються оптимальним стратегіям, а радше функція придатності використовує принципи еволюції для створення нових агентів, які можуть демонструвати кращі стратегії.

У RL агенту не надаються жодні навчальні приклади [29]. Натомість агент проходить процес навчання, заснований на методі спроб і помилок. Зворотний зв'язок, що надається системою, служить лише для того, щоб вказати агенту, чи добре він виконує свою роботу чи ні, не уточнюючи агенту, як виконувати поставлене завдання. Винагороди (або штрафи), що надаються агенту, також називають підкріпленнями.

Завдання агента полягає в тому, щоб знайти політику (відповідність станів до дій), яка максимізує підкріплення в довгостроковій перспективі. Крім того, агенту, що базується на RL, необхідно провести дослідження можливих дій, щоб визначити середню віддачу від кожної дії. Одночасно агент також повинен використовувати набір дій, які, як він виявив, мають високу середню віддачу. Цей сценарій найкраще описується за допомогою задачі k-руких бандитів.

Існує кілька підходів до навчання на основі дослідження, пов'язаного з реальним життям (RL). Найпоширенішими підходами є динамічне програмування, Q-навчання та навчання на основі часових різниць (TD).

2.2 Інтелектуальні алгоритми прийняття рішень у балансуванні

Навчання деревом рішень. Дерева рішень здебільшого використовуються як ефективні механізми вибору дій в іграх. Тому дерево рішень зазвичай використовується як класифікатор, де умови «класифікуються» у відповідні дії або рішення. Як згадувалося раніше, дерева рішень мають перевагу, оскільки ми можемо зручно перевірити, чому дерево рішень приходить до певного рішення.

Дерево рішень – це, перш за все, функція, яка приймає вектор атрибутів на вхід і надає рішення на вихід. Дерева рішень можна використовувати як для неперервних, так і для дискретних вхідних і вихідних значень. Для прийняття рішення дерево рішень виконує серію тестів [30]. Кожен вузол представляє тест атрибута, а гілки вузла представляють можливі значення для атрибута. Листові вузли дерева рішень представляють значення, які може повернути функція.

Дерева рішень можна використовувати для керування реактивним агентом у реальному часі за допомогою багаторівневого навчання. Алгоритм використовувався для навчання агента навичкам мета-рівня, таким як дриблінг, перегравання суперника та прямі та наскрізні паси.

Дерева рішень також можна використовувати в іграх у реальному часі для імітації поведінки людей під час гри. Наприклад, навчання дерев рішень, разом із міркуваннями на основі прецедентів (CBR) та імітованим відпалом, було

використано для побудови довгострокової стратегії в DEFCON.- багатокористувацька гра в реальному часі. Зокрема, алгоритм ID3 був використаний для створення високорівневого плану для ігрового агента у вигляді дерева рішень. План включав визначення початкового розміщення наземних підрозділів, а також стратегій атаки та оборони флоту та метафлоту.

Глибоке навчання з підкріпленням. Використання глибоких нейронних мереж (DNN) та RL набуло популярності як підхід до розробки ігрових агентів, здатних до навчання. DNN можуть навчатися функціям у багатовимірних просторах. Однак, щоб досягти такого навчання, DNN спочатку необхідно навчити, і саме тут застосовується навчання з підкріпленням (RL). RL використовується для навчання DNN через кілька випадків самостійної гри.

Глибоке RL можна використовувати для програмування агентів, здатних навчатися різним іграм. Глибоке RL було використано, щоб ігровий агент міг навчатися 49 іграм для Atari 2600. Згорткову нейронну мережу (CNN) було використано разом з алгоритмом Q-навчання, створюючи глибоку Q-мережу (DQN), завданням якої було навчання оптимальної функції значення дії Q.

Агент отримував на вхід лише пікселі та поточний рахунок у кожному інтервалі, і йому вдалося вивчити кожну з 49 ігор для Atari 2600, досягнувши вражаючих рівнів. Більшість застосувань підходів глибокого RL були реалізовані у двовимірних ігрових світах [31]. Реалізувано глибокий RL для тривимірних ігрових світів з частково спостережуваними станами та застосували його до гри шутер від першої особи (FPS). Крім того, замість того, щоб зосереджуватися лише на пікселях, що представляють ігровий світ, ця реалізація включала інші функції, такі як наявність або відсутність ворогів.

Щоб врахувати часткову спостережуваність станів, автори використали глибоку рекурентну Q-мережу (DRQN). Використана DRQN складалася з CNN, вихідні дані якої передавалися до мережі довгої короткочасної пам'яті (LSTM). Таким чином, вхідний вектор ознак подавався до CNN, вихідні дані якої передавалися до мережі LSTM.

У роботі було реалізовано асинхронну архітектуру RL, яка могла навчати

глибокі нейронні мережі (DNN), вимагаючи при цьому відносно невеликих ресурсів. Основна ідея цього підходу відрізняється від традиційних методів глибокої Q-мережі (DQN), оскільки вони спираються на *повторення досвіду.* Асинхронне (паралельне) виконання кількох агентів на кількох екземплярах середовища означає, що агенти будуть паралельно перебувати в різних станах.

Підходи до обчислювального інтелекту . Підходи на основі обчислювального інтелекту (OI) також мають перевагу в тому, що агента можна навчати через самостійну гру. Ще однією перевагою підходів на основі Cl є те, що їх можна легко гібридизувати з існуючими методами машинного навчання для використання їхніх можливостей. Нарешті, алгоритми на основі Cl дозволяють нам спостерігати за емерджентною поведінкою еволюційних систем за різних правил взаємодії.

Еволюційні обчислювальні підходи

Експертні помічники (EA) використовувалися для створення оптимальних контролерів штучного інтелекту (AI) для ігрових агентів з метою досягнення оптимальної поведінки. Для досягнення такого завдання можна використовувати генетичні алгоритми (GA). Наприклад, генетичний алгоритм (GA) використовувався для розробки ігрового агента в Infinite Mario Bros, яким керував автоматний автомат (FSM). Правила переходу автоматного автомата були розроблені для того, щоб знайти найбільш підходяще правило переходу для кожного можливого стану.

Генераторні алгоритми (ГА) також використовувалися для налаштування ботів у грі від першої особи від першої особи, щоб зменшити значні витрати часу на побудову експертних систем на основі правил. Головною метою роботи було використання ГА для ефективного налаштування певних параметрів, які визначали, як і коли боти виконуватимуть певні дії. ГА також використовувалися для створення стратегій військових ігор. Зовсім недавно ГА, що базується на вірусній інфекції для ефективного пошуку шляху в грі Tower Defense. ГА також нещодавно використовувався в футболі роботів для уникнення перешкод та планування шляху.

Генетичне програмування (GP) використовується для навчання ігор з метою еволюції алгоритмів (або комп'ютерних програм) на основі графів чи дерев, які застосовуються для керування поведінкою ігрових агентів. У роботі уло проведено дослідження з еволюції дерев поведінки (приклад дерева поведінки наведено на рисунку 2.3 у грі DEFCON. Ігровий агент був наділений базовими здібностями для гри. Аспектом агента, який еволюціонував, була його стратегічна поведінка (як і коли агент повинен виконувати конкретні завдання). Стратегічна поведінка агента була закодована в дереві поведінки. Дерево було ініціалізовано випадковим чином, а потім еволюціонувало.

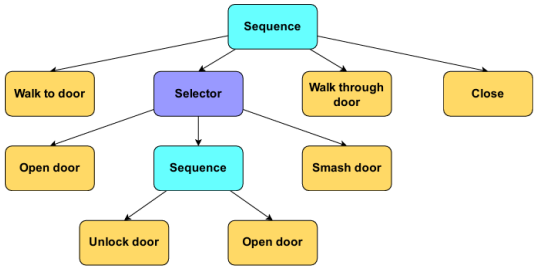

Рисунок 2.3 - Приклад дерева поведінки.

GP також використовувався для оптимізації координації команди софт-ботів шляхом еволюції для домену RoboCup Soccer Server. Одне дерево контролювало команду. Метою було використовувати GP для еволюції дерева, доки це не призведе до того, що команда продемонструє прийнятну
координацію під час гри у футбол.

Пакет попередньої обробки, реалізує шаблон проектування «Фасад» [25], що забезпечує єдину та зрозумілу точку входу, через яку можна отримати доступ до всієї наданої функціональності.

Крім того, обробка всіх методів реалізована за допомогою шаблону проектування *пулу об'єктів* [25], який дозволяє мати один екземпляр одного класу, який повинен створювати, містити та обробляти всі екземпляри іншого класу. У цьому випадку **InstanceHandler** є інтерфейсом для такого класу, а **MethodHandler** – фактичною реалізацією. **Method** – це клас, який має

оброблятися, і він представляє один член набору відфільтрованих нормалізованих методів. **Інтерфейси InstanceHandler** та Instance – це єдині два файли, які повинні імпортуватися будь-яким клієнтським класом, оскільки вони забезпечують абстракцію всього, що обробляється цим пакетом. Нейроеволюція стосується застосування методів еволюційних обчислень до еволюції нейронних мереж. Існують різні підходи, такі як еволюція лише ваг мережі або також її топології. Нейроеволюція доповнюючих топологій (NEAT) – це добре відомий підхід до нейроеволюції, який застосовувався в різних іграх. Цей підхід не передбачає жодних виведених або теоретичних знань про оптимальну топологію нейронної мережі або типи зв'язків (тобто рекурентні або прості зв'язки).

NEAT базується на таких трьох ідеях:

- Еволюційний пошук починається з якомога простих топологій. Складні структури виникають в результаті еволюційного процесу, і їхнє виживання залежить від їхньої пристосованості.
- *Історичне маркування* використовується для рекомбінації мереж з різними топологіями. За допомогою цього механізму подібність між мережами визначається за допомогою інноваційних номерів, що запобігає дороговартісному топологічному обстеженню.
- Топологічні інновації також «захищені» видоутворенням24 . *Таким* чином, конкуренція за виживання обмежується лише нішами, де мережі мають подібні топології .

У роботі NEAT було використано для розробки контролера для The Open Racing Car Simulator (TORCS). Еволюція була спрямована на досягнення двох цілей з точки зору навичок: (1) швидка та надійна їзда на різних типах трас та (2) обгін суперників, уникаючи зіткнень. Для досягнення цих навичок NEAT було використано для розвитку кожної навички окремо та їх рекомбінації на подальшому етапі. Кожна нейронна мережа представляла окрему особину (кандидат рішення) у популяції, а PSO використовувався для калібрування ваг кожної нейронної мережі, доки вона не почала працювати оптимально. Зафіксовані експериментальні результати показали, що цей метод працює краще, ніж еволюційні

підходи для навчання нейронних мереж.

PSO також використовувався як частина гібридного алгоритму, який також використовував EA та ANN. Алгоритм використовувався для оптимізації оцінки позицій ради директорів у *Capture Go.* Була розроблена популяція функцій оцінки у вигляді ANN, щоб знайти хороших кандидатів, які були додатково оптимізовані за допомогою PSO для кращої продуктивності. Оскільки EA не потребують початкових знань предметної області для пошуку прийнятних рішень, алгоритм зміг виявити корисні рішення з ANN.

Оптимізація колонії мурах (ACO) – це сімейство паралельних алгоритмів оптимізації, заснованих на штучних мурахах. Ідея ACO полягає в імітації реальних мурах у пошуку шляху вирішення заданої проблеми з мінімальними витратами. ACO використовувався, щоб агент міг навчитися грі в *Tempic.* Підхід полягав у використанні ACO для вивчення оптимального вектора ваг для функції гіпотези агента. Вивчення оптимального вектора ваг здійснювалося шляхом пошуку оптимального шляху ваг у графі ваг. Згідно із записаними експериментальними результатами, цей метод перевершив більшість традиційних методів RL.

2.3 Архітектура систем автоматичного балансування ігрових механік

Роль агента-симбіонта сприйняття в AdaptiveSGA полягає в систематичному зберіганні сприйняттів, важливих для досягнення балансування складності протягом усієї гри. Дані, пов'язані з грою, надаються агенту-симбіонту сприйняття як сприйняття. Завдання агента полягає в тому, щоб перевірити сприйняття та перетворити його на вектор ознак. Після виконання цього перетворення агент-симбіонт сприйняття додає вектор ознак до спільної пам'яті [32].

Таким чином, спільна пам'ять містить список векторів ознак, які також можна називати історією сприйняття. Історію сприйняття, що зберігається в спільній пам'яті, потім можна використовувати для аналізу або класифікації навичок гравця. Агент, відповідальний за класифікацію навичок гравця, є агентом-симбіонтом класифікації.

Як, коли і де зберігаються сприйняття під час виконання або гри, є важливим врахування досягнення ефективного та плавного регулювання складності. Як згадувалося в описі проблеми в розділі 1, досягнення плавного регулювання складності є критично важливим, щоб гравець не відчував, що відбувається елемент шахрайства [33].

Як обговорювалося, комунікаційна мембрана є єдиним компонентом AdaptiveSGA, який може надсилати повідомлення симбіонтним агентам. Ці повідомлення мають форму сприйняття. Отримавши сприйняття, кожен симбіонтний агент діятиме на середовище симбіонта за допомогою своїх виконавчих механізмів. Дії, що виконуються симбіонтними агентами, відрізняються залежно від того, який тип симбіонтного агента діє.

Наприклад, агент-симбіонт сприйняття впливає на середовище симбіонта, оновлюючи історію сприйняття у спільній пам'яті. Агент-симбіонт класифікації виконує дві ролі. Перша - інформувати комунікаційну мембрану про те, як історія сприйняття має зберігатися у спільній пам'яті, вказуючи дві речі: (1) кількість ознак, які повинен містити кожен вектор, та (2) назви ознак, які повинні міститися у векторі.

Таким чином, комунікаційна мембрана повинна передавати цю інформацію агенту-симбіонту сприйняття. Метою повідомлення цих вимог є формування контракту між агентом класифікації та симбіонтом сприйняття щодо формату даних, що зберігаються у спільній пам'яті. Повідомлення вимог допомагає уникнути ситуації, коли агент-симбіонт класифікації отримує несумісні вектори ознак зі спільної пам'яті. Друга роль агента-симбіонта класифікації полягає в класифікації або прогнозуванні майстерності персонажа суперника, щоб вибрати рівень складності. Обраний рівень складності повідомляється комунікаційній мембрані. Для цього класифікаційний агент. Симбіонтний агент може бути реалізований за допомогою різних підходів.

NPC зазвичай розроблені для впливу на навколишнє середовище на основі визначених цілей. Це можна побачити, наприклад, у грі Pac-Man. Гравець-людина керує персонажем Pac-Man, щоб з'їдати точки в лабіринті, уникаючи

зіткнень з привидами (NPC суперника). Мета кожного привида - переслідувати Pac -Man, перш ніж усі точки будуть з'їдені. Однак, коли між Pac-Man та привидом немає прямої видимості, метою кожного привида є блукання лабіринтом .

У лабіринті також є таблетки, які, коли їх ковтає Пекмен, змушують привидів ставати синіми. Коли привиди сині, вони «зникають» на кілька секунд, якщо стикаються з Пекменом [34]. Таким чином, коли Пекмен ковтає таблетку, метою кожного привида стає ухилення від Пекмена.

Отже, існують три цілі, між якими можуть перемикатися NPC-привиди: *блукати, переслідувати* або *ухилятися.* Хоча ці цілі й надають напрямок для кожного NPC, їх потрібно детальніше розбити. Наприклад, все одно має бути наданий алгоритм для блукання лабіринтом, щоб NPC знали, що робити під час кожного оновлення гри. Таким чином, частину прийняття рішень (визначення цілей на основі умов) та частину виконання (перетворення цілей на окремі дії) можна розділити.

Роль симбіонтного агента, що приймає рішення, полягає у встановленні механізму прийняття рішень NPC. Тому симбіонтний агент, що приймає рішення, повинен постійно обробляти дані, пов'язані з грою, щоб забезпечити актуальність цілей NPC (див. рис. 2.4).

Рисунок 2.4 - Загальний огляд ролі симбіонтного агента, що приймає рішення.

Після оновлення цілі NPC, агент-симбіонт, що приймає рішення, повинен повідомити про ціль комунікаційній мембрані. Завдання комунікаційної мембрани полягає в передачі оновленої цілі агенту-симбіонту, що виконує її, щоб дії NPC у світі гри відображали його оновлені цілі.

Роль агента-симбіонта виконання полягає в тому, щоб перетворювати цілі

агента-симбіонта, що приймає рішення, на дії, які має виконати NPC. Рівень абстрактності цілей залежить від гри та рішення програміста. Наприклад, у грі PacMan метою може бути переслідування персонажа PacMan, що є абстрактним, оскільки включає послідовність дій, які NPC повинен виконати для її досягнення.

З іншого боку, цілі можуть бути створені як менш абстрактні. Прикладом цього є мета передачі м'яча товаришу по команді у футбольній грі. У цьому випадку єдиним завданням агента-симбіонта виконання буде пошук найкращого гравця, якому потрібно передати м'яч [35]. Таким чином, гра, до якої застосовується AdaptiveSGA, та розсуд програміста визначатимуть кінцевий дизайн пулу можливих цілей та дій. Завдання агента-симбіонта виконання полягає у визначенні для кожного оновлення гри наступної дії NPC на основі його поточної мети.

Комунікаційна мембрана - єдиний компонент середовища симбіонта, який взаємодіє з агентом-симбіонтом виконання. Під час кожного оновлення гри комунікаційна мембрана надсилає агенту-симбіонту виконання сприйняття, яке містить дані, пов'язані з грою, та поточну мету NPC. Якщо поточна мета відрізняється від тієї, що була в попередньому оновленні, агент-симбіонт виконання оновлює цю інформацію внутрішньо. На рисунку 2.5 показано загальний вигляд інформації, яка надходить до агента-симбіонта виконання, та результат, що виробляється.

Агент-симбіонт виконання обробляє сприйняття, що надається комунікаційною мембраною, для вилучення різних фрагментів інформації про стан гри. Поточна мета NPC використовується для визначення його наступної негайної дії. Інші фрагменти Інформація, що міститься у сприйнятті, допомагає агенту-симбіонту виконання обчислити найціннішу дію з урахуванням поточної мети.

Рисунок 2.5 - Роль, яку виконує агент-симбіонт виконання.

Отже, кінцева роль агента-симбіонта виконання полягає не лише в обчисленні наступної дії NPC, але й у забезпеченні обчислення найбажанішої дії. Агент-симбіонт виконання повинен бути розроблений як агент, який завжди прагне знайти найоптимальнішу дію для виконання NPC, враховуючи час, відведений для обчислення такої дії.

2.4 Висновок по розділу

У другому розділі було досліджено методи та моделі автоматичного балансування ігор із використанням штучного інтелекту. Розглянуто методи машинного навчання для адаптивного балансування гри, що дозволяють аналізувати поведінку гравців і автоматично коригувати ігрові параметри. Проаналізовано інтелектуальні алгоритми прийняття рішень у процесі балансування, які забезпечують підтримку оптимального рівня складності та динаміки гри. Також досліджено архітектуру систем автоматичного балансування ігрових механік та визначено основні компоненти таких систем. У результаті встановлено ефективні підходи до реалізації адаптивного балансування в сучасних ігрових застосунках.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ СИСТЕМИ АВТОМАТИЧНОГО БАЛАНСУВАННЯ ТА ОЦІНКА ЇЇ ЕФЕКТИВНОСТІ

3.1 Проєктування та реалізація системи адаптивного балансування

Як згадувалося в попередньому розділі, відповідним застосуванням AdaptiveSGA є гра, оскільки AdaptiveSGA є моделлю ігрового агента. Також було визначено, що для достатнього тестування моделі потрібна гра з достатньою складністю. Таким чином, обраною грою був симульований футбол (див. рис. 3.1). Ідея гри полягає в тому, щоб імітувати футбольний матч, в якому дві команди протистоять одна одній, і жодна з них не контролюється користувачем.

Рисунок 3.1 - Знімок екрана симуляційної футбольної гри.

Причиною вибору футболу як тестового середовища для AdaptiveSGA є те, що футбол представляє собою динамічне середовище в реальному часі. Воно створює ситуацію, коли кожен агент на полі має обмежений час для прийняття рішень. Наприклад, якщо агент має м'яч, і агент витрачає забагато часу на обчислення наступного рішення, інший агент (тобто суперник) забере м'яч. Таким чином, агенти мають на меті приймати рішення якомога швидше, переконавшись, що прийняті ними рішення також корисні.

Крім того, футбол також є недетермінованою грою. Наприклад, якщо агент вирішує передати м'яч товаришу по команді, немає гарантії, що коли м'яч досягне призначеного місця призначення, товариш по команді все ще буде там.

В інших випадках передбачуваний суперник може перехопити пас.

Отже, ніколи не гарантується, що обрана дія завжди призведе до певного стану. Зрештою, швидка апроксимація скінченного набору можливих станів є нетривіальним завданням у динамічному та недетермінованому середовищі. Тому вважається, що таке середовище є достатньо складним для ретельного тестування AdaptiveSGA [36].

Правила гри. Як видно на рисунку 3.1, є дві команди: жовта команда та синя команда. Головне завдання кожної команди - перемогти. Щоб команда виграла матч (або гру), вона повинна забити більше голів, ніж команда суперника (тобто, якщо жовта команда має два голи, а синя команда - один гол наприкінці матчу, то жовта команда перемагає, а синя команда програє).

Щоб забити гол, команда повинна направити м'яч так, щоб він пройшов повз стійки воріт команди суперника. Стійки воріт синьої команди знаходяться на правій стороні поля. Стійки воріт жовтої команди знаходяться на лівій стороні поля. Воротарів немає м'яча з-за бокової лінії та кутових ударів не застосовуються. Коли м'яч вибивають за межі ігрового поля, він автоматично знову з'являється в центрі поля. Початкового удару немає. На початку під час матчу м'яч розміщується в центрі поля, і обидві команди мають шанс першими дістатися до м'яча. Аналогічно, після забитого голу м'яч знову з'являється в центрі поля.

У грі також немає поняття фолу. Тому немає штрафних ударів, а це означає, що гравці можуть зіткнутися (це буде обговорено далі в розділі про алгоритм флокування) один з одним, не призводячи до штрафного удару. Крім того, немає жодних уточнень щодо того, як гравці повинні підбирати один одного за м'яч. Правило офсайду також не застосовується. Гравець може передати м'яч будь-якому іншому гравцеві на полі.

Як вже, можливо, стало очевидним, у гру вбудовані лише прості правила, щоб уникнути включення складних правил. Включення складних правил, безумовно, покращило б реалістичність гри [37]. Однак такі зусилля не зроблять суттєвого внеску в досягнення мети експерименту.

Мета цього експерименту - виявити досяжність динамічного

балансування складності на основі штучного інтелекту в адаптивній грі за допомогою SGA. Впровадження гри з простими правилами усуває необхідність зосереджувати значну частину зусиль на її розробці. Натомість зусилля перенаправляються на тестування та аналіз Adaptive SGA.

Структура рішення. Мета полягає в тому, щоб представити дизайн прототипу концептуально та з точки зору загального огляду. Після попереднього обговорення структури, детальне вивчення деталей програмного забезпечення стає легшим для розуміння та інтуїтивно зрозумілим.

Лекції з програмування. Прототип було створено за допомогою Java 8. Графічний інтерфейс користувача (GUI) було розроблено за допомогою JavaFX. Що стосується зберігання даних, єдиною інформацією, яка постійно зберігалася, була історія сприйняття. Таким чином, система управління базами даних не використовувалася. Натомість для зберігання кожного запису, представленого агентом-симбіонтом сприйняття, використовувався файл CSV (дані, розділені комами).

Структура рішення. Будь ласка, зверніться до рисунка 3.2 для зображення структури рішення. Структура проілюстрована з використанням підходу «зверху вниз». Кінцевий користувач (тобто гравець-людина) взаємодіє з верхнім шаром, який є графічним інтерфейсом користувача. Аналогічно, графічний інтерфейс користувача знає лише (чорні стрілки вказують на компоненти, про які знає кожен компонент) про ігрові сутності, а також про ігровий об'єкт тощо. У наступних підрозділах описується кожен компонент.

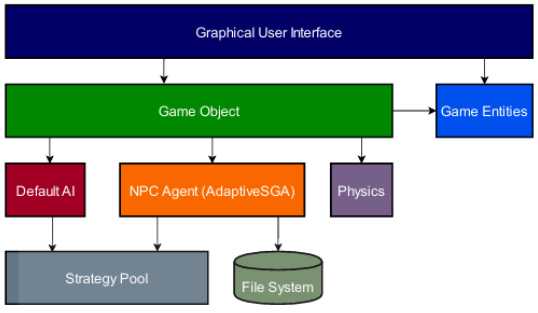

Рисунок 3.2 - Концептуальна основа.

Графічний інтерфейс користувача

Графічний інтерфейс користувача відображає інформацію, необхідну для перегляду симуляції під час виконання. Ця інформація відображається в текстових полях (див. рис. 3.3) і включає такі дані:

1. Кількість зіграних (симульованих) ігор на даний момент.
2. Одиниці часу, що залишилися для поточної симульованої гри.
3. Рахунок кожної команди за поточну гру.

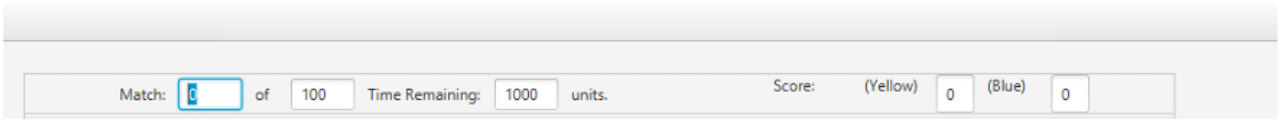

Рисунок 3.3 - Корисна інформація про виконання, що відображається в графічному інтерфейсі користувача.

Відразу після закінчення однієї гри починається інша. Послідовність ігор продовжується, доки кількість зіграних ігор не досягне загальної кількості ігор, дозволених для цього конкретного експерименту. Наприклад, якщо максимальна дозволена кількість ігор становить 100, симуляція не зупиниться, доки не буде зіграно 100 ігор, якщо користувач не призупинить симуляцію.

Кожна одиниця часу визначається як час, необхідний для переходу між кадрами [38]. Розглянемо потік ігрового об'єкта, який виконується кожні 100 мілісекунд. Таким чином, кожне оновлення відбувається кожні 100 мілісекунд, що означає, що гра тривалістю 600 одиниць часу повинна тривати приблизно 1 хвилину.

Щоразу, коли команда забиває гол (тобто виводить м'яч повз стійки воріт команди суперника), рахунок цієї команди збільшується на 1. Таким чином, кінцевий рахунок для кожної команди дорівнює сумі голів, забитих відповідною командою. Кінцевий рахунок (рахунок жовтої та синьої команд відповідно) визначає результат матчу. Наприклад, якщо кінцевий рахунок {жовта команда = 10; синя команда = 5}, то перемогою буде жовта команда. Якщо кінцевий рахунок

нічия (тобто {жовта команда = 10; синя команда = 10}), то результатом буде нічия.

Щоб розпочати симуляцію, користувачеві потрібно натиснути кнопку «Пуск» . Це призведе до запуску симуляції, доки не мине максимальна кількість ігор. Кнопка «Пауза» призведе до зависання симуляції до натискання кнопки «Пуск» , щоб симуляція могла продовжитися [39]. Кнопка «Зберегти» використовується для збереження поточного стану (знімку) AdaptiveSGA (до його потенційних змін) для подальшого аналізу, якщо це буде потрібно.

Класи контролерів використовуються для обробки взаємодії між користувачем і системою, і вони показані на рисунку 3.4. HomeController відповідає за обробку подій та операцій, пов'язаних з головним екраном. SimulationController відповідає за обробку подій та операцій, пов'язаних з екраном симуляції.

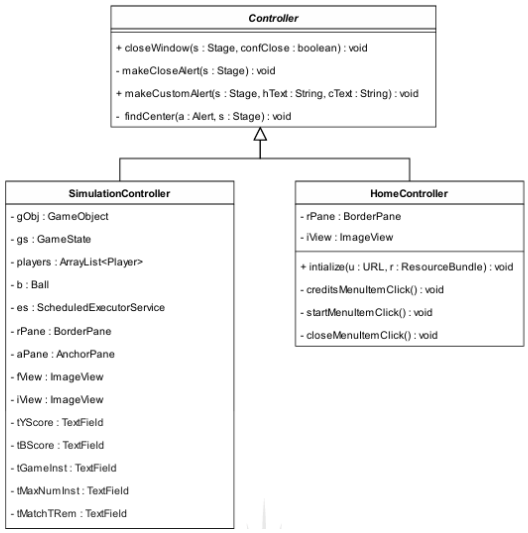

Рисунок 3.4 - Зв'язок між класами контролерів графічного інтерфейсу користувача (GUI).

Ігрові сутності

Ігрові сутності – це NPC та інші об'єкти, що складають ігровий світ. Кожен гравець є NPC. Об'єктами, які існують у ігровому світі, окрім NPC, є футбольне поле та футбольний м'яч. Як графічний інтерфейс користувача, так і ігровий об'єкт знають про ігрові сутності. На рисунку 3.5 зображено класи, що представ-

ляють усі сутності. Однак футбольне поле явно не представлене як клас у коді.

Ігровий об'єкт

Ігровий об'єкт відповідає за полегшення симуляції. Для цього ігровий об'єкт повинен відстежувати стан гри під час кожного оновлення. Щоб згенерувати новий стан гри, ігровий об'єкт спочатку повинен надати поточний стан гри контролерам NPC (тобто штучному інтелекту за замовчуванням, який керує жовтою командою, та AdaptiveSGA, який керує синьою командою). Діаграма класів ігрового об'єкта проілюстрована на рисунку 3.5.

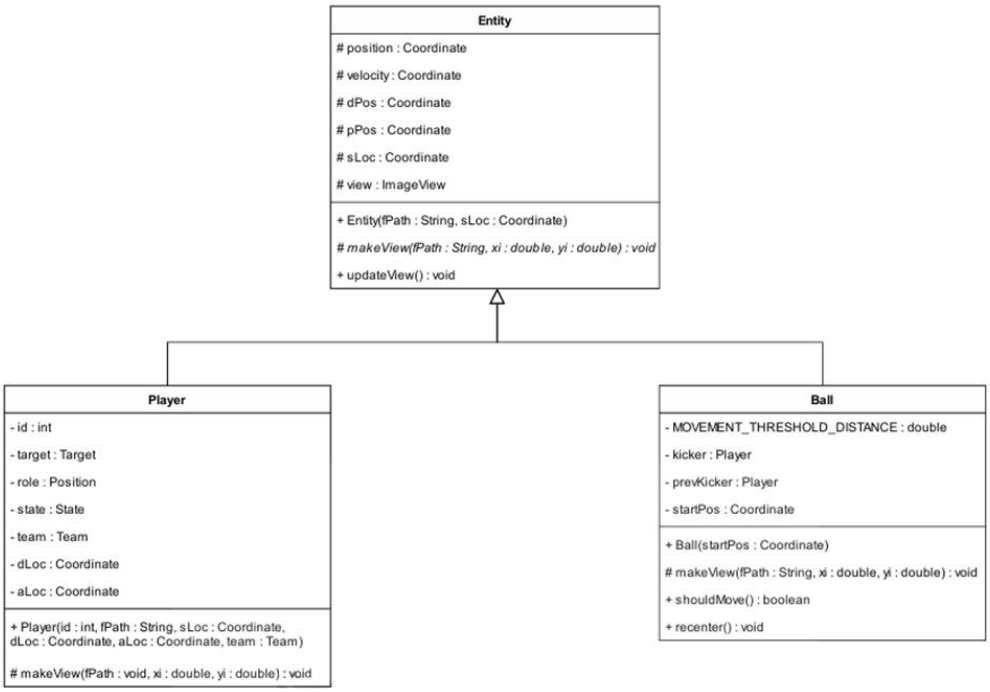

Рисунок 3.5 - Діаграма класів, що представляє сутності.

Отримавши поточний стан гри, контролер NPC використовує його, щоб визначити, що повинен робити кожен з NPC. Після прийняття рішення контролер NPC повертає вибрану дію ігровому об'єкту. Ігровий об'єкт використовуватиме вибрану дію контролера NPC, щоб виконати дію NPC за допомогою фізичного об'єкта [40]. Це призведе до зміни стану гри. Змінений [новий] стан гри потім буде надіслано до графічного інтерфейсу користувача для відображення на екрані.

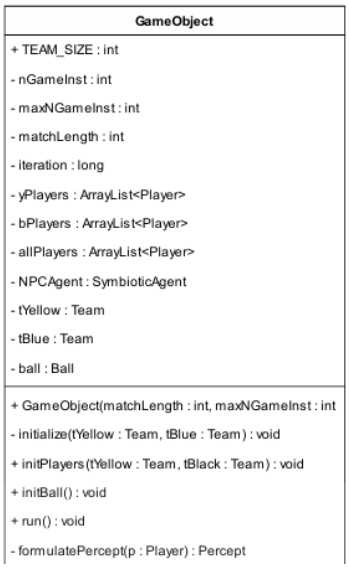

Рисунок 3.6 - Діаграма класів, що представляє ігровий об'єкт.

Клас фізики використовується ігровим об'єктом, і більшість його властивостей і функцій є статичними. Це пояснюється тим, що клас фізики був розроблений не для підтримки будь-якого стану, який би вимагав керування його життєвим циклом користувачем (тобто ігровим об'єктом). Щоразу, коли ігровому об'єкту потрібно використовувати клас фізики, він викликає його функції статично. Деякі приклади використання - обчислення відстані між двома NPC та визначення того, чи бере участь NPC у зіткненні. Клас фізики зображено на рисунку 3.7.

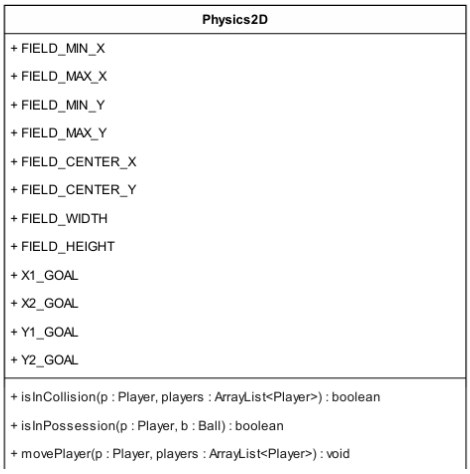

Рисунок 3.7 - Діаграма класів, що представляє клас фізики.

Навіть у недетермінованих умовах життєво важливо, щоб певні закони завжди діяли. Наприклад, якщо м'яч перетинає лінію воріт між двома стійками воріт, це гол для команди, яка забиває. Певні аспекти гри завжди повинні бути передбачуваними. Щоб це встановити, було необхідним включення курсу фізики.

Штучний інтелект за замовчуванням

ШІ за замовчуванням – це модуль, який діє як контролер для жовтої команди. ШІ за замовчуванням незмінний, тобто він не адаптується з часом. Тільки AdaptiveSGA динамічно змінює поведінку синьої команди з часом. ШІ за замовчуванням існує виключно як механізм, що використовується для перевірки функціонування AdaptiveSGA. Найпростіший спосіб перевірити, чи може AdaptiveSGA підтримувати рівну гру з суперником, – це виключити адаптивність як якість суперника. Діаграма класів, що зображує ШІ за замовчуванням, показана на рисунку 3.8.

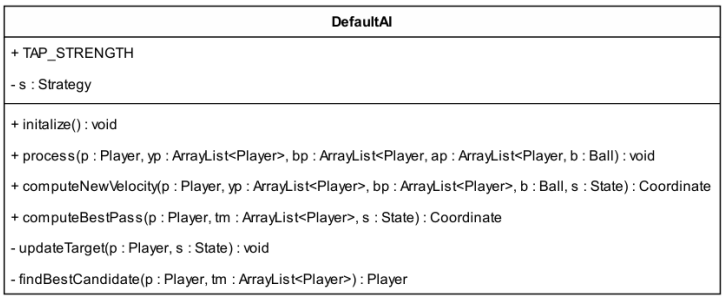

Рисунок 3.8 - Діаграма класів, що представляє клас штучного інтелекту за замовчуванням.

На початку симуляції ігровий об'єкт відповідає за виклик етапу ініціалізації штучного інтелекту за замовчуванням. Під час ініціалізації штучний інтелект за замовчуванням створює екземпляр своєї стратегії. Стратегія – це, по суті, алгоритм, який приймає рішення для NPC. Такий алгоритм може бути кінцевим автоматом, деревом рішень, деревом поведінки або будь-яким іншим алгоритмом прийняття рішень, обраним для експерименту. Стратегії існують у пулі стратегій. Користувач прототипу може вибрати будь-яку стратегію, доступну в пулі

стратегій, як стратегію штучного інтелекту за замовчуванням.

NPC-агент

Варто наголосити, що кожен NPC у команді не має свого виділеного контролера. Натомість кожна команда має виділений контролер. Виділений контролер жовтої команди є штучним інтелектом за замовчуванням. Виділений контролер синьої команди є екземпляром AdaptiveSGA. Як обговорювалося в розділі 9, AdaptiveSGA складається з чотирьох симбіонтних агентів, спільної пам'яті та комунікаційного модуля мембрана (див. рис. 3.10). Один клас симбіотичних агентів керує життєвим циклом кожного клас симбіонтних агентів (виконання, прийняття рішень, сприйняття та класифікація симбіонтних агентів).

Варто також зазначити, що класи симбіонтних агентів є абстрактними класами через функцію ***act*** . Це пояснюється тим, що функція ***act*** поводитиметься по-різному залежно від того, який симбіонтний агент працює в середовищі симбіонта. Наприклад, первинний симбіонтний агент, що приймає рішення, поводитиметься не так, як розширений симбіонтний агент, що приймає рішення. Таким чином, атрибути класу симбіотичного агента повинні бути об'єктами конкретних класів (тобто BasicDecisionMakingAgent).

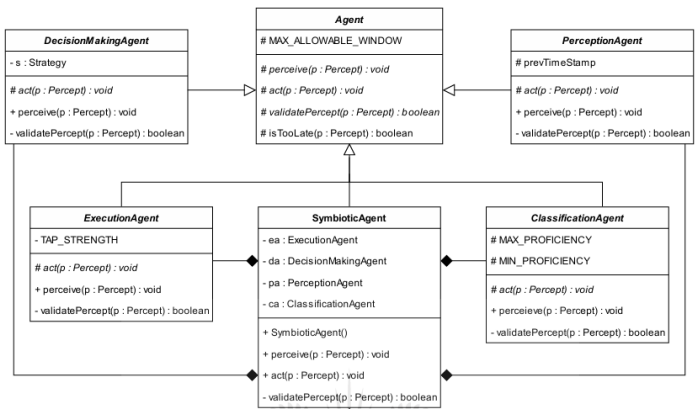

Рисунок 3.9 Діаграма класів, що показує зв'язки між класами в екземплярі.

Комунікаційна мембрана сприяє потоку інформації в середовище симбіонта та з нього. Крім того, комунікаційна мембрана сприяє потоку інформації між агентами-симбіонтами всередині середовища симбіонта. Ігровий об'єкт знає

лише про клас симбіотичного агента. Коли ігровий об'єкт надсилає сприйняття симбіотичному агенту, симбіотичний агент оцінює сприйняття та пересилає його комунікаційній мембрані, лише якщо воно є дійсним.

Функція ініціалізації комунікаційної мембрани призначає конкретні версії симбіонтних агентів змінним або атрибутам, що зберігаються в комунікаційній мембрані. Клас симбіотичного агента надає конкретні версії симбіонтних агентів комунікаційній мембрані. Після ініціалізації симбіотичний агент дає команду комунікаційній мембрані розпочати передачу сприйняттів різним симбіонтним агентам, щоб визначити, що NPC повинен робити далі.

Пул стратегій

Пул стратегій – це модуль, що містить класи стратегій. Стратегія – це алгоритм, який використовується для прийняття рішень для NPC, враховуючи поточний стан гри. Це може бути кінцевий автомат, нейронна мережа, дерево рішень, дерево поведінки або будь-який інший метод, який дозволяє NPC вибирати дії під час гри. На рисунку 3.11 показано абстрактний клас Стратегія .

Будь-який клас стратегії повинен відповідати вимогам, встановленим абстрактним класом **Стратегія** . Повинна бути функція ініціалізації, функція для визначення того, чи є умова (тобто, що агент має м'яч) істинною, а також функція для оцінки стану, до якого слід перейти, на основі поточного стану. Це допомагає створювати підтримувані стратегії, які є принаймні певною мірою передбачуваними.

Файлова система

У прототипі можна постійно зберігати лише два типи даних. Поточний стан/конфігурація AdaptiveSGA – це перший тип даних, який можна зберігати у файловій системі.

Цього можна досягти, натиснувши кнопку «Зберегти », що призведе до створення бінарного файлу, який буде збережено у файловій системі.

Історія сприйняття – це другий тип інформації, який можна зберігати у файловій системі. Історію сприйняття також можна зберігати у файловій системі у вигляді CSV-файлу. Таким чином, не було потреби в реляційній базі даних чи

іншому складному механізмі зберігання. Введення та виведення файлів обробляється класом FilelOHandler (див. рисунок 3.10).

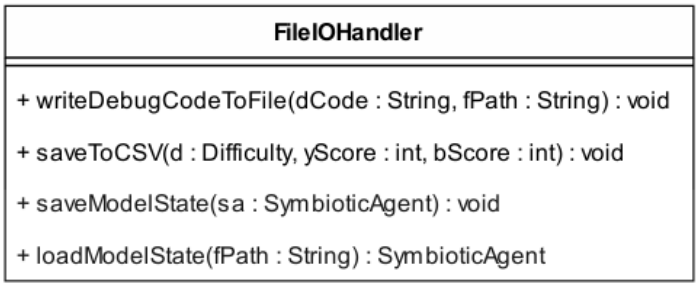

Рисунок 3.10 - Діаграма класів, що зображує клас обробника файлового вводу-виводу.

У підрозділі описано структуру рішення для DiskiSimulation та розглянуто кожен із компонентів прототипу. Для опису кожного компонента також була включена діаграма класів, що ілюструє кількість класів у компоненті, а також їхні атрибути та операції. Показано повну діаграму класів для DiskiSimulation.

Цей розділ присвячено поясненню, які частини прототипу досягають динамічного балансування складності на основі адаптивного ігрового штучного інтелекту. Комунікаційна мембрана є основою досягнення як адаптивного ігрового штучного інтелекту, так і динамічного балансування складності, що зрештою призводить до динамічного балансування складності на основі адаптивного ігрового штучного інтелекту. Комунікаційна мембрана відповідає за заміну (перемикання в середовище симбіонтів та з нього) агентів симбіонтів.

Заміна агентів залежить від того, чи відповідають агенти, які наразі присутні в середовищі симбіонта, рівню складності, рекомендованому класифікаційним симбіонтним агентом.

Оскільки симбіонтні агенти, що приймають та виконують рішення, визначають поведінку NPC, лише їх заміна відображатиме зміну складності ігрового середовища.

Заміна агента-симбіонта сприйняття впливає лише на ефективність зберігання сприйняттів у спільній пам'яті. Аналогічно, заміна агента-симбіонта класифікації впливає лише на ефективність класифікації майстерності опонента.

Діаграма активності ілюструє логіку, якої дотримується комунікаційна мембрана, визначаючи, чи слід замінити будь-яких симбіонтних агентів (які в цьому випадку є агентами прийняття рішень або виконання, оскільки цей тип заміни зумовлений лише рівнем складності).

Коли гра закінчується, комунікаційна мембрана надсилає сприйняття, що описує стан гри, агенту-симбіонту сприйняття для зберігання у спільній пам'яті. Потім комунікаційна мембрана надсилає сприйняття агенту-симбіонту класифікації, що описує результат гри.

Після того, як агент-симбіонт класифікації передбачить рівень складності опонента, він надсилає сприйняття на комунікаційну мембрану, що вказує на бажаний рівень складності, спонукаючи комунікаційну мембрану переглянути поточний рівень складності.

Після того, як комунікаційна мембрана оновить рівень складності, вона вирішить, чи потрібно замінити поточного агента-симбіонта, відповідального за прийняття рішень або виконання, або обох, щоб задовольнити новий рівень складності. Якщо заміна має відбутися, комунікаційна мембрана зробить її.

Як приклад сценарію, розглянемо гру, яка щойно завершилася, де остаточний рахунок становить 2 бали для жовтої команди та 0 для синьої команди. 1 це означає, що жовта команда перемогла, а синя команда програла. Ця інформація буде надіслана агенту-симбіонту класифікації для прогнозування майстерності гравця.

Припустимо, що агент-симбіонт класифікації передбачає, що гравець більше не аматор, а професіонал. Агент класифікації підкаже комунікаційній мембрані змінити рівень складності на середній. Отримавши цю інформацію, комунікаційна мембрана повинна буде вирішити, яких агентів замінити, щоб задовольнити рівень складності.

Якщо комунікаційна мембрана вирішить замінити симбіонтного агента,

відповідального за прийняття рішень, більш просунутим аналогом, поточний агент, відповідальний за прийняття рішень, буде вилучений із середовища симбіонта. Більш просунутий симбіонт, відповідальний за прийняття рішень, буде замінено на середовище симбіонта. Коли почнеться наступна гра, агент NPC матиме більш просунутого симбіонта, відповідального за прийняття рішень, який працюватиме всередині середовища симбіонта.

Після того, як ми розглянули, як можна замінити симбіонтних агентів залежно від рівня складності, тепер розглянемо, як це також призводить до адаптивного ігрового штучного інтелекту. Як приклад, розглянемо випадок, коли комунікаційна мембрана замінює симбіонтного агента, який приймає рішення. Впровадження складнішого симбіонтного агента, який приймає рішення, вже передбачає, що загальна поведінка NPC зміниться. Отже, не тільки рівень навичок NPC зміниться, але й їхня поведінка також зміниться в процесі.

Отже, масштабування складності таким чином також призводить до створення адаптивного ігрового ШІ. Оскільки комунікаційна мембрана замінює агентів після зміни складності, поведінка NPC також обов'язково змінюється. Це відрізняється від масштабування складності гри шляхом простого збільшення швидшості або сильнішості NPC (тобто налаштування параметрів). Натомість, динамічне балансування складності на основі адаптивного ігрового ШІ досягається шляхом маніпулювання поведінкою NPC для відстеження складності, зберігаючи при цьому розважальну цінність. Більше того, прототип досягає цього за допомогою симбіотичного ігрового агента.

3.2 Алгоритми оптимізації ігрових параметрів у реальному часі

Алгоритм зграйкового руху був включений у прототип, щоб запровадити певний рівень цілеспрямованої (невипадкової) поведінки в поведінці NPC. Кінцева швидкість кожного NPC була векторною сумою 4 різних компонентів швидкості. Перший компонент швидкості був спрямований до центру мас команди NPC (згуртованість). Другий компонент швидкості був спрямований до

уникнення зіткнень (розділення).

Третій компонент швидкості був спрямований на уточнення швидкості NPC шляхом вирівнювання NPC з його товаришами по команді (вирівнювання) (з точки зору швидкості). Четвертий компонент швидкості був спрямований до цільового пункту призначення NPC. Алгоритм реалізації кожного з компонентів швидкості був адаптований. Реалізація цільового компонента швидкості була адаптована. У наступних підрозділах описано результати різних компонентів.

Слідування центру маси (згуртованості)

Якщо кожен NPC у команді відстежує центр мас команди, для команди стає природним підтримувати свою формацію. Тому було важливо врахувати цю складову швидкості. Для розрахунку складової центру мас було виконано такі кроки:

1. Обчисліть центр мас команди.
2. Визначте, наскільки сильно NPC буде притягуватися до центру мас.

Розрахунок центру мас команди проводився шляхом підсумовування розташування окремих членів команди (тобто підсумовування x-координат усіх членів команди та підсумовування y-координат членів команди відповідно). Після отримання суми її ділили на кількість членів команди (тобто окремо ділили x- та y-суми).

Після обчислення центру мас команди наступним кроком було віднімання розташування NPC від середнього розташування команди. Ця операція була важливою, оскільки вона забезпечувала «напрямок» до центру мас. Однак, перед відніманням розташування гравця, розташування центру мас множилося на невелику константу, щоб виразити, наскільки сильно NPC буде притягуватися до центру мас. Нарешті, отримана складова швидкості нормалізувалася в одиничний вектор.

Вплив цієї складової швидкості можна було виявити під час моделювання. NPC, що належать до однієї команди, рухалися як єдине ціле і часто знаходилися поблизу місця, де знаходився м'яч. Таким чином, реалізація цієї складової швидкості дала очікувані результати. Однак було важливо знайти

правильну константу для контролю сили тяжіння до центру мас.

Без множення на невелику константу, NPC занадто сильно притягувалися до центру мас, і це призводило до того, що NPC скупчувалися біля м'яча, не розтікаючи його по полю. Тому, чим менше значення використовуваної константи, тим менше NPC притягувалося до центру мас команди.

Уникнення зіткнень

Якщо NPC збираються в грaї близько один до одного, логічно очікувати, що деякі з них зіткнуться. Таким чином, було важливо також включити цей компонент швидкості до алгоритму грaї. Компонент швидкості був реалізований за допомогою таких кроків:

1. Обчисліть суму відстаней між NPC та кожним з інших NPC.
2. Визначте, наскільки сильно NPC буде відштовхуватися від інших NPC.

Щоб обчислити суму відстаней від гравця до інших NPC, координати відповідного NPC віднімали від координат кожного з інших NPC. Потім відстані додавали, щоб знайти суму. Наступним кроком було поділити розташування NPC на кількість інших NPC та зазначити результат. Нарешті, отриману складову швидкості нормалізували в одиничний вектор.

Результат

Включення цього компонента швидкості призвело до того, що NPC не збиралися надто близько один до одного. Це був бажаний ефект. Однак успішне функціонування цього компонента швидкості тісно залежало від компонента центру мас. Наприклад, сильніше тяжіння до центру мас команди призводило до зіткнення деяких NPC.

Крім того, свою роль відігравала складова швидкості, пов’язана з переслідуванням цілі. Якщо NPC не відчував сильного тяжіння до своєї цілі, він неминуче зміщувався ближче до центру маси своєї команди, що збільшувало ймовірність зіткнень. З іншого боку, якщо NPC надто сильно тягнуло до мети, існувала висока ймовірність зіткнення з іншими NPC. Тому пошук відповідних значень для констант, що відповідають за притягання (або відштовхування),

мав вирішальне значення.

Уточнення швидкості (вирівнювання)

Ще однією важливою проблемою щодо алгоритму зграй є узгодження швидкості NPC зі швидкостями його товаришів по команді, щоб вся команда демонструвала прийнятний рівень координації. Для уточнення швидкості NPC зі швидкістю його товаришів по команді було зроблено такі кроки:

1. Обчисліть суму швидкостей членів команди NPC.
2. Поділіть суму швидкостей на кількість учасників команди.

Як видно з наведених вище кроків, реалізація цієї складової швидкості є простою. Першим кроком було створення вектора швидкості, координати x та y якого є сумами координат x та y членів команди NPC відповідно. Потім вектор швидкості був поділений на кількість членів команди. Нарешті, вектор був нормалізований.

Результат

Вплив цього компонента швидкості проявлявся в тому, що NPC, що належать до однієї команди, демонстрували помітну координацію з точки зору швидкості. Звичайно, також було важливо знайти правильний рівень узгодженості між NPC. Тим не менш, загальна координація була очікуваною.

Слідування за цільовою метою

У футбольному матчі NPC не завжди переслідуватимуть одну й ту саму ціль. Наприклад, поки один NPC женеться за м'ячем, інший NPC може бігти вперед. Тому було важливо чітко визначати ціль для кожного NPC під час кожного оновлення. Було п'ять можливих цілей, за якими NPC могли переслідувати: *М'ЯЧ, ПОЗИЦІЯ НАПАДУ, ПОЗИЦІЯ ЗАХИСТУ, АВТОГОЛИ, ГОЛИ З МОЖЛИВОСТЕЙ.* Рисунок 3.11 ілюструє цю концепцію.

Якщо ціль – *М'ЯЧ,* це означає, що метою NPC є керування у напрямку футбольного м'яча. Тому, хоча інші фактори впливають на рух NPC (тобто рух ближче до центру зграї та уникнення зіткнень), NPC буде значною мірою рухатися у напрямку м'яча. Кожен NPC має позицію захисту *(DEFENCEPOSITION)* та позицію нападу *(OFFENCE_POSITION),* призначені йому на початку гри.

Логічно, що NPC повинен рухатися до своєї позиції захисту, коли команда NPC перебуває під атакою. Крім того, NPC повинен рухатися до своєї позиції нападу, коли команда NPC координує атаку.

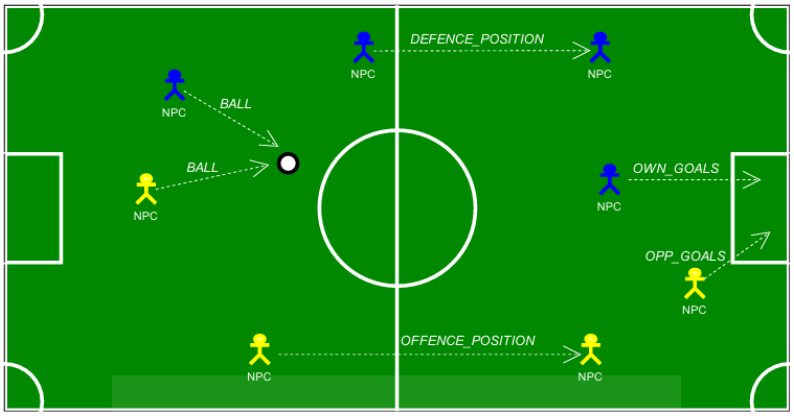

Рисунок 3.11 - NPC, що прямують до своїх відповідних цілей.

Якщо ціллю NPC є OWN_GOALS (ВЛАСНІ_ГОЛИ), NPC швидко рухатиметься до своїх воріт. Це корисно, коли NPC хоче прикрити ворота своєї команди, щоб запобігти забити гол. Крім того, якщо ціллю NPC є OPP_GOALS (ОПЕРАТИВНІ_ГОЛИ), NPC буде сильно рухатися до воріт суперника.

Важливо зазначити, що навіть якщо NPC з команд суперників можуть рухатися до м'яча, кожен NPC все одно буде притягнутий до центру своєї команди. Мотивація для того, щоб NPC завжди збиралися поблизу центру мас своїх команд, полягає в тому, щоб, коли NPC має м'яч, у нього швидко з'явилися опції навколо нього, куди він може передати м'яч.

Завдання вибору цілі для кожного NPC було доручено контролеру. Таким чином, контролер NPC відповідав за вибір завдань для синіх NPC, тоді як штучний інтелект за замовчуванням відповідав за вибір цілей для жовтих NPC. З точки зору зграї, можна було спостерігати рух NPC до певних цілей. Вплив цілі на рух NPC мав регулюватися з урахуванням трьох інших компонентів швидкості. Кожен NPC мав наближатися до своєї цілі з достатньою терміновістю, дотримуючись загального «темпу» команди.

3.3 Оцінка ефективності та експериментальна валідація системи

Цей розділ має на меті обговорити загальні висновки, зроблені в результаті впровадження та тестування AdaptiveSGA. Вкрай важливо критично оцінити загальний результат моделі, щоб визначити, чи є вона належною реалізацією для NPC-агента. Ігри часто динамічні, і тому алгоритми, які керують NPC, є критично важливими.

Повільний час відгуку може призвести до того, що NPC пропускатимуть виконання дій протягом значної кількості циклів оновлення. Як обговорювалося, реалізація прототипу використовувала динамічне середовище, щоб зробити тестування AdaptiveSGA максимально реалістичним у контексті гри. Тому для гри було важливо продовжуватися незалежно від того, чи зможе NPC прийняти рішення.

Досягнення адаптивного ігрового штучного інтелекту

Поведінкові риси та ігрові умови

Для досягнення адаптивного ігрового штучного інтелекту (ШІ) для агента NPC було реалізовано такі моделі поведінки:

1. Кидок м'яча до воріт команди суперника
2. Передача м'яча товаришу по команді, який знаходиться попереду NPC
3. Передача м'яча випадковому партнеру по команді
4. Передача м'яча найближчому партнеру по команді
5. Здійснення атакуючого прориву
6. Здійснення захисного прориву
7. Переслідування м'яча
8. Обігравши суперника, ведучи м'яч
9. Пошук простору

Щоб NPC задовольняв певний тип поведінки, має бути виконана певна умова. Наступні умови були введені в прототип для оцінки алгоритмами контролера під час вибору дій для NPC:

1. NPC знаходиться поруч із воротами команди суперника
2. NPC близький до цілей своєї команди
3. Товариш NPC по команді має м'яч
4. М'яч має суперник NPC
5. NPC має м'яч
6. NPC знаходиться поруч із одним зі своїх товаришів по команді.
7. Наразі попереду NPC є товариш по команді NPC.
8. NPC знаходиться біля м'яча
9. Противники скупчуються навколо NPC
10. NPC має м'яч, але його закривають суперники
11. М'яч у воротах команди суперника (команда NPC щойно забила гол)
12. NPC розташований на призначеній йому оборонній позиції
13. NPC розташований у призначеній для нього позиції для атаки

Алгоритми контролера прийняття рішень

Використовуючи вищезазначені поведінкові компоненти, NPC було зроблено адаптивним шляхом реалізації трьох різних алгоритмів контролера, де кожен алгоритм визначає загальну поведінку керованих NPC. Перший використаний алгоритм був кінцевим автоматом, і він складався лише з п'яти з дев'яти можливих поведінкових ознак:

1. Кидок м'яча до воріт команди суперника
2. Здійснення захисного прориву
3. Здійснення атакуючого прориву
4. Переслідування м'яча
5. Передача м'яча випадковому партнеру по команді

Ілюстрацію скінченного автомата показано на рисунку 3.13.

Слід припустити, що генератор псевдовипадкових чисел, який використовується протягом реалізації прототипу, використовує лінійну конгруентну формулу, яка може створювати рівномірно розподілену послідовність цілих чисел, якщо прямо не зазначено інше. Як показано на рисунку 3.13, кінцевий автомат використовував лише 5 з 13 можливих умов:

1. М'яч має суперник NPC
2. NPC знаходиться біля м'яча
3. NPC має м'яч
4. М'яч має суперник NPC
5. Товариш NPC по команді має м'яч

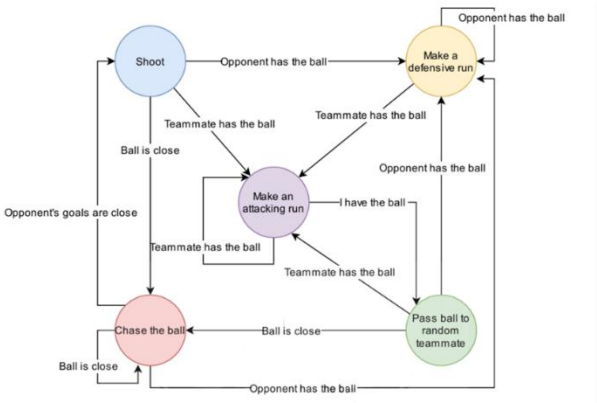

Рисунок 3.12 - Кінцевий автомат, що використовується в DiskiSimulation.

Враховуючи кількість можливих станів та кількість умов, що використовуються для переходу між станами, поведінка NPC, що генерується кінцевим автоматом, була відносно простою. Однак кінцевий автомат дійсно давав бажані результати, оскільки NPC діяли досить логічно (тобто вони могли ганятися за м'ячем, передавати м'яч товаришам по команді та забивати голи).

Другим контролером NPC, який було реалізовано для керування поведінкою NPC, було дерево рішень. Дерево рішень було реалізовано як ідеальне бінарне дерево з наступними 7 з дев'яти можливих поведінкових ознак:

1. Кидок м'яча до воріт команди суперника
2. Здійснення захисного прориву
3. Здійснення атакуючого прориву
4. Передача м'яча товаришу по команді попереду NPC
5. Обігравши суперника, ведучи м'яч
6. Переслідування м'яча

7. Пошук простору

Дерево рішень оцінювало наступні 6 з 13 можливих умов для вибору певного виду поведінки – або дії:

1. М'яч має суперник NPC
2. NPC має м'яч
3. NPC знаходиться біля м'яча
4. Противники скупчуються навколо NPC
5. Наразі попереду NPC є товариш по команді NPC.
6. NPC знаходиться поруч із воротами команди суперника

Результатом використання дерева рішень як контролера стало те, що NPC демонстрували помітно складнішу поведінку порівняно з випадком, коли використовувався кінцевий автомат. NPC також рідше стикалися, оскільки вони могли розпізнавати скупчення суперників і бігти у вільний простір. Більше того, NPC мали можливість обганяти суперників або передавати м'яч товаришам по команді, які знаходилися попереду них, щоб збільшити шанси на гол.

Обмеження автомата з скінченною кількістю станів стали очевидними завдяки незначному зростанню складності, яке додало дерево рішень. Використання деревоподібної структури дозволило зручніше виражати більш складну логіку. Наприклад, недоліком, який одразу ж можна було виявити в структурі автомата з скінченною кількістю станів, було те, що неможливо було виразити вибір дій на основі більш ніж однієї умови.

Це було дуже обмежувальним, оскільки дозволяло переходи лише за умови виконання однієї з певної кількості умов (тобто, якщо A або B, перехід до C). Використання дерева рішень дозволяло вибирати дії на основі поєднання умов (тобто, якщо A та B, тоді перехід до C). Можливість виконувати такі оцінки давала дереву рішень перевагу над кінцевим автоматом. Це проявлялося у підвищенні інтелекту NPC під час керування деревом рішень.

Третім реалізованим контролером було дерево поведінки. Як вузли дерева поведінки було обрано такі можливі умови та дії. Умови були сформульовані у вигляді запитань, а дії – у вигляді тверджень для полегшення розрізнення.

Включені дії були наступними:

1. Обігравши суперника, ведучи м'яч
2. Кидок м'яча до воріт команди суперника
3. Передача м'яча товаришу по команді попереду NPC
4. Передача м'яча випадковому партнеру по команді

Включені умови були наступними:

1. NPC має м'яч, але його закривають суперники
2. NPC знаходиться поруч із воротами команди суперника

Ілюстрацію дерева поведінки показано на рисунку 3.14.

Використання дерева поведінки призвело до складнішого та інтелектуальнішого прийняття рішень порівняно з кінцевим автоматом та деревом рішень. Це можна пояснити тим, що дерево поведінки дозволяє ще більше виразити свої логіки.

Можливість створювати послідовності завдань у деревоподібній структурі дозволяє вказати певні поведінкові риси, які NPC повинен проявляти послідовно, де виконання майбутніх завдань залежить від успішного виконання попередніх завдань. Більше того, дерево поведінки запровадило можливість виконання деяких завдань лише як резервного варіанту на випадок, коли інші пріоритетні завдання зазнають невдачі.

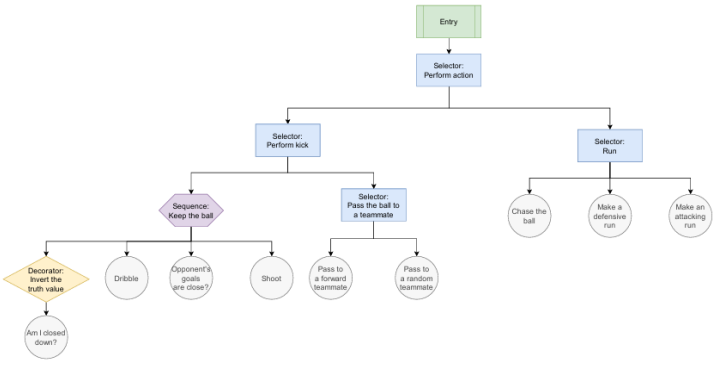

Рисунок 3.13 - Дерево поведінки, що використовується в DiskiSimulation.

Завдяки трьом алгоритмам контролера для керування поведінкою NPC, компонент адаптивного ігрового штучного інтелекту прототипу був успішно

реалізований. Крім того, кожен алгоритм контролера призвів до унікальної складності поведінки, що згодом призвело до помітної різниці в інтелекті NPC. Це стало корисним для досягнення динамічного балансування складності, яке обговорюється нижче.

Досягнення динамічного балансування складності

Налаштування складності

Для досягнення динамічного балансування складності спочатку потрібно було запровадити метод вимірювання складності. Тому інтелект NPC потрібно було розподілити за різними класами складності. Для простоти реалізації було визначено три рівні складності: легкий, середній та складний. Кожен рівень складності потім був безпосередньо пов'язаний з кожним із трьох рівнів майстерності суперника наступним чином: аматор ^ легкий ; кваліфікований ^ середній ; експерт ^ складний .

Вибір правильних контролерів

Кожному рівню складності було призначено алгоритм контролера. *Легкому* рівню складності було призначено кінцевий автомат, *середньому* – дерево рішень, а *складному* – дерево поведінки. Для порівняння алгоритмів прийняття рішень було проведено 100 ігор, у які було зіграно два алгоритми один проти одного. Кожна гра тривала 2000 одиниць часу, де кожна одиниця часу була встановлена на 80 мілісекунд. Таким чином, кожна гра тривала 2 хвилини 40 секунд.

Не було жодної мотивації для вибору саме такої тривалості кожної гри. Вибір був цілком довільним. Головним стимулюючим фактором було уникнення ігор, які занадто короткі або занадто довгі, щоб забезпечити збір змістовних даних.

Щоб ввести невелику кількість випадковості в симуляцію, існувала 20% ймовірність того, що кожен з контролерів вибере випадкову поведінку (з пулу поведінок, що постачається з алгоритмом прийняття рішень). Першими алгоритмами, які були зіграні один проти одного, були дерево рішень та кінцевий автомат. Кругова діаграма на рисунку 3.15 показує результати, отримані після 100 ігор.

Судячи з результатів, дерево рішень виявилося ефективнішим за автомат з кінцевим набором станів, що підтвердило обґрунтованість його віднесення до середнього рівня складності. Наступним кроком стало порівняння поведінкового дерева з деревом рішень. На круговій діаграмі на малюнку 3.14
показано результати після 100 ігор.

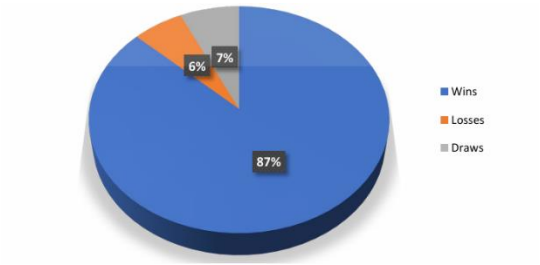

Рисунок 3.14 - Кругова діаграма, що відображає кількість перемог, нічиїх та поразок для контролера дерева рішень.

Як показано на рисунку 3.15, команда, що контролювала дерево поведінки, виграла 97% ігор проти команди, що контролювала дерево рішень. Крім того, команда, що контролювала дерево поведінки, забила загалом 678 голів, тоді як команда, що контролювалася деревом рішень, забила загалом 123 голи. Це призвело до загальної різниці голів у 555 голів, що означає середню різницю голів у 6 голів за гру. Модальна різниця голів становила 7 голів, що близько до середнього значення. На рисунку 3.15 показано стовпчасту діаграму, яка показує різницю рахунків для кожного зі 100 матчів.

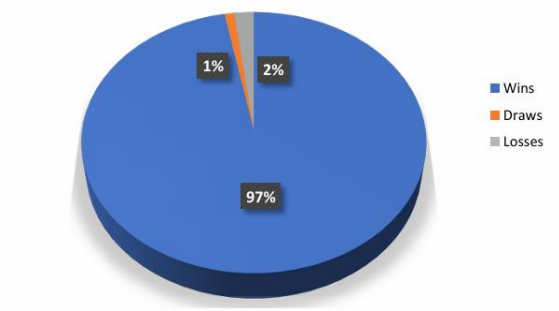

Рисунок 3.15 - Кругова діаграма, що відображає кількість перемог, нічиїх та поразок для контролера дерева поведінки

Судячи з результатів, зображених на обох діаграмах, контролер дерева поведінки був кращим за контролер дерева рішень, і це підтверджувало логіку, що стоїть за його пов'язуванням зі ***складним*** рівнем складності. Нарешті, щоб перевірити всі вищезазначені припущення, контролер дерева поведінки був зіграний проти контролера скінченного автомата. Результат показано на рисунку 3.17. Команда, керована деревом поведінки, виграла всі ігри проти команди, керованої скінченним автоматом.

На рисунку 3.16 наведено гістограму, яка ілюструє різницю в балах для кожного зі 100 конкурсів.

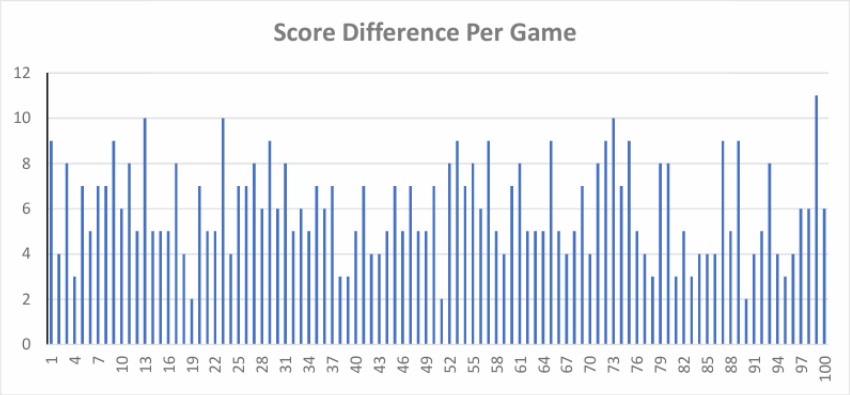

Рисунок 3.16 - Гістограма, що відображає різницю в балах за кожним конкурсом.

Балансування складності кодування в симбіотичному ігровому агенті

Тепер, коли кожному рівню складності було призначено власний алгоритм контролера, який забезпечував унікальну поведінку NPC, наступним кроком була спроба досягти динамічного балансування складності. Це було зроблено за допомогою архітектури AdaptiveSGA. Першим кроком була реалізація трьох різних симбіонтних агентів, що приймають рішення, на основі трьох різних контролерів, описаних вище.

Комунікаційна мембрана тоді мала б відповідальність за заміну агентів прийняття рішень у середовищі симбіонта та поза ним за потреби. Рисунок 3.17 ілюструє цей підхід. Основною метою було забезпечити, щоб заміна агентів

прийняття рішень відбувалася таким чином, щоб забезпечити динамічний баланс складності. Для досягнення цієї мети комунікаційна мембрана мала надіслати сприйняття агенту-симбіонту класифікації, що описує результат гри.

Завданням агента-симбіонта класифікації було використовувати надані дані для визначення поточного виміряного рівня майстерності гравця. Це допомогло б визначити, чи є поточний рівень складності відповідним. Якщо поточний рівень складності потрібно було змінити, агент-симбіонт класифікації спонукав би комунікаційну мембрану внести зазначену зміну. Комунікаційна мембрана перемикалася б на рівень складності, визначений агентом-симбіонтом класифікації, замінюючи його відповідним агентом-симбіонтом, який приймає рішення.

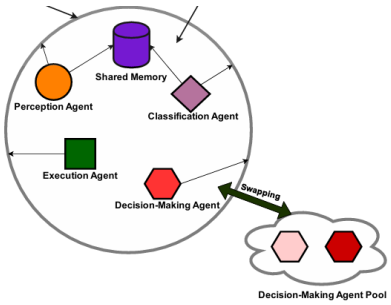

Рисунок 3.17 - Як у середовищі симбіонта може існувати лише один симбіонтний агент, що приймає рішення.

Наприклад, якщо агент-симбіонт класифікації спонукав комунікаційну мембрану змінити складність з легкої на середню, комунікаційна мембрана замінила б агента-симбіонта, що приймає рішення, чий алгоритм контролера відповідає легкому, на агента-симбіонта, що приймає рішення, чий алгоритм контролера відповідає середньому.

Після кодування цих правил в AdaptiveSGA, модель була використана проти статичного ігрового ШІ. Причина цього ґрунтується на роботі, де запропоновані контролери були протестовані з різними статичними контролерами, щоб з'ясувати, чи можуть запропоновані контролери виконувати динамічне масштабування складності.

Для керування жовтою командою було реалізовано клас «штучний

інтелект за замовчуванням», тоді як AdaptiveSGA керувала синьою командою. Штучний інтелект за замовчуванням – це статичний контролер. Штучний інтелект за замовчуванням (ШІ за замовчуванням) використовував дерево рішень, яке було описано в попередньому розділі як алгоритм контролера для середнього *рівня* складності. Мотивацією для цього було перевірити, чи зможе AdaptiveSGA успішно масштабувати складність, щоб вона відповідала рівню дерева рішень для 100 ігор.

Метрика для вимірювання динамічного балансування складності

Щоб визначити, чи може AdaptiveSGA успішно збалансувати складність змагання з суперником, було використано підхід гри на рівні. Ідея цього підходу полягає в тому, щоб NPC (AdaptiveSGA) намагався досягти рівної гри з суперником. Виявили, що гравці-люди вважали гру на рівні більш цікавою після гри з різними суперниками на базі штучного інтелекту в стратегічній грі в реальному часі.

Рівна гра – це гра, в якій майстерність гравця-людини відповідає майстерності суперника, керованого комп'ютером. Оскільки для кожної гри у футболі існує три можливі результати (виграш, поразка або нічия), рівняння, яке використовується для опису задоволеності гравця, було адаптовано і представлено як рівняння 3.1.

$$\mathrm{W} = \mathrm{L} = (\mathrm{n} - \mathrm{D})/2 \qquad (3.1)$$

W позначає кількість перемог, *L* – кількість поразок, *n* – кількість зіграних ігор, а *D* – кількість нічиїх. Ідея полягала в тому, щоб спробувати максимізувати кількість перемог і поразок, мінімізуючи загальну кількість нічиїх, оскільки вважалося, що вони радше дратують, ніж розважають.

Для цієї роботи поріг для нерівної гри було встановлено на рівні $|W - L| > 0{,}1\,n$. Тому, якщо різниця між виграними та програними іграми була більшою за 10% від загальної кількості зіграних ігор, змагання вважалося нерівним. Також, що стосується кількості нічиїх, якщо $D < 0{,}1n$, змагання вважалося цікавим; якщо

$D > 0{,}2n$, змагання вважалося таким, що дратує.

Що стосується кількості голів, забитих окремими учасниками змагань, бажаніше було мінімізувати різницю м'ячів між ними, одночасно максимізуючи кількість голів, забитих кожним з них. Наприклад, припустимо, що жовта команда виграє з рахунком 2 голи до 1; це добре, тому що різниця м'ячів - 1 - якомога менша.

Однак, рахунок 2 голи до 1 може бути не таким захопливим, як рахунок 5 голів до 4, де різниця голів все ще невелика, але кожен із учасників гри забив більшу кількість голів. Тому $|g_1 — g_2|$ має бути якомога меншим, тоді як $\max(g_1, g_2)$ - якомога більшим (де g_1 та g_2 - кількість голів, забитих відповідними учасниками гри наприкінці матчу).

Максимальна ідеальна різниця м'ячів для підтримки видовищності матчу була встановлена на рівні 30. Це пояснюється тим, що різниця м'ячів 30 або більше означає середню різницю рахунків у 3 голи або більше за гру. Отже, результат $|g_1 — g_2| > 30$ вважався таким, що негативно впливає на видовищність матчу.

AdaptiveSGA проти DefaultAI

Команда AdaptiveSGA проти команди DefaultAI провела 100 ігор, щоб визначити, чи зможе AdaptiveSGA досягти рівного результату проти DefaultAI. Результати змагання – з точки зору перемог, нічиїх та поразок – зображено круговою діаграмою на рисунку 3.18.

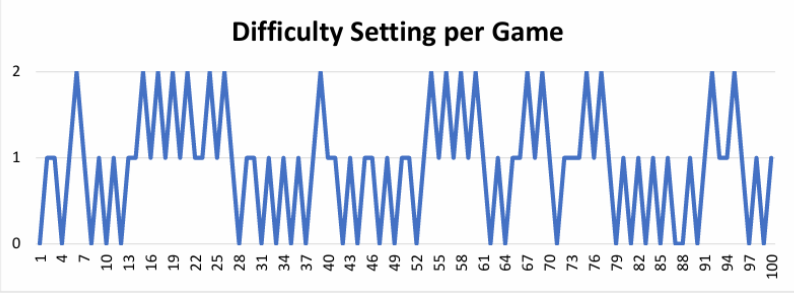

Рисунок 3.19 - Лінійна діаграма, що показує зміну рівня складності для кожної зі 100 ігор.

AdaptiveSGA виграла 45 ігор, програла 44 ігри та зіграла внічию 11 ігор проти DefaultAI. Оскільки \W — L\ < 0,1 n, AdaptiveSGA змогла підтримувати рівний результат з DefaultAI. Кількість нічиїх була більшою за 0,1n, але меншою за 0,2n. Таким чином, змагання не було ні цікавим, ні таким, що дратує (щодо кількості нічиїх).

AdaptiveSGA забила загалом 267 голів, тоді як DefaultAI забила загалом 289 голів. Абсолютна різниця м'ячів (\g 1 — g 2 \) становила 22 голи (середня різниця м'ячів 2 голи за гру), що нижче порогу для невидовищної гри з точки зору різниці м'ячів. Однак варто зазначити, що модальна різниця м'ячів становила 1, що менше за середню різницю м'ячів.

Лінійна діаграма на рисунку 3.19 показує коливання складності протягом 100 зіграних ігор між AdaptiveSGA та DefaultAI (0 означає *легкий* ; 1 означає *середній* ; 2 означає *складний).* AdaptiveSGA перемикалася в *легкий* режим 27 разів, *у середній* режим 55 разів і *в складний* режим 18 разів, що означає, що змагання здебільшого проходили в середньому режимі.

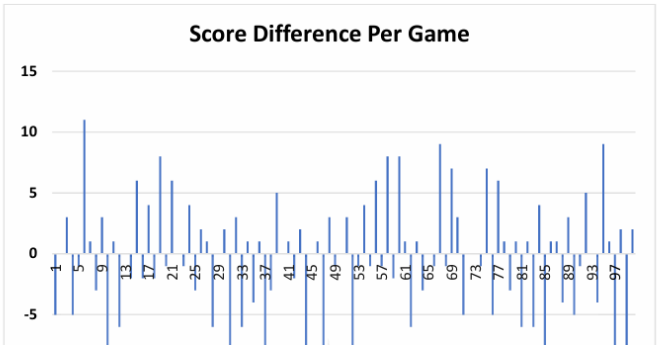

Рисунок 3.19- Гістограма, що ілюструє розподіл різниці в балах за 100 ігор.

Ще один ключовий висновок з рисунка 3.19 полягає в тому, що майже не траплялося, щоб один рівень складності зберігався більше двох ігор. У такому разі DefaultAI стикався з різноманітною поведінкою суперника. На рисунку 3.19 показано розподіл різниці в рахунках за 100 іграми між AdaptiveSGA та DefaultAI. Головним результатом, який слід відзначити на стовпчастій діаграмі,

є велика кількість забитих голів кожним учасником.

AdaptiveSGA можна застосувати до гри. Однак є фактори моделі, які не обговорювалися. Обговорення таких факторів забезпечить глибший погляд на практичність моделі з точки зору можливості її використання для вирішення різних видів проблем.

Гнучкість. Першим ключовим фактором є гнучкість. Вкрай важливо встановити, наскільки регульованою є AdaptiveSGA, щоб побачити, чи можна її змінювати у відповідь на різні типи ігор. Одним із важливих аспектів симбіотичних агентів є те, що архітектура дозволяє програмісту визначати власних симбіонтних агентів. Таким чином, модель симбіотичного ігрового агента є гнучкою за своєю суттю. Наприклад, програміст може додавати агенти до агентів. Симбіонтні агенти в AdaptiveSGA були призначені для покриття основних функцій для NPC. Як обговорювалося в розділах, AdaptiveSGA надає можливість реалізовувати різні типи архітектур агентів для забезпечення підвищеної гнучкості.

Масштабованість. Враховуючи середній час відгуку NPC-агента, до AdaptiveSGA можна впровадити більше функціональності. Для більших ігор зі складнішими середовищами можна впровадити більш досконалі симбіонтні агенти для масштабування AdaptiveSGA, щоб він відповідав вимогам.Однак існує верхня межа масштабування моделі. Наприклад, у динамічному середовищі агенти зазвичай мають часове вікно, протягом якого вони повинні прийняти рішення про дію. Тому наявність значно великої кількості агентів-симбіонтів, що працюють у симбіонтному середовищі, може призвести до того, що агенту NPC знадобиться більше часу для обчислення рішень.

Застосовність. У цій роботі AdaptiveSGA було застосовано до симуляційного футболу. Однак AdaptiveSGA можна застосовувати до різних видів ігор. Вимога полягає в тому, щоб гра була змагальною, оскільки модель базується на виконанні балансування складності. Це може варіюватися від перегонів і бійок до інших видів спорту. Однак у серйозних іграх, де ШІ призначений більше для навчання, ніж для змагання, AdaptiveSGA також може бути використаний. Класифікаційний симбіонтний агент все одно класифікуватиме навички гравця-

людини та спонукатиме комунікаційну мембрану відповідно масштабувати складність питань.

Хоча AdaptiveSGA і відповідала вимогам динамічного балансування складності на основі штучного інтелекту в адаптивній грі, модель можна покращити. Більше того, майбутня робота розглядає не лише можливі покращення, але й подальші експерименти, які можна провести для дослідження різних аспектів моделі. Першим прикладом є більш складний симбіонтний агент класифікації.

Класифікація симбіонтного агента. Класифікаційний симбіонтний агент, який використовувався в прототипі, працював прямолінійно: збільшуйте складність після програшу, зменшуйте складність після перемоги та не змінюйте складність після нічиєї. Хоча цей підхід працює, інші важливі елементи інформації, такі як кількість голів, забитих у попередній грі, не враховувалися. Більше того, прямолінійний підхід, найімовірніше, був причиною відносно великої кількості розіграшів. Незміна рівня складності після розіграшу, найімовірніше, була нерозумною ідеєю. Впровадження алгоритмів, які можуть враховувати більше параметрів перед вибором рівня складності, має шанс зменшити велику кількість розіграшів.

Різні типи виконання агентів-симбіонтів . Дослідження AdaptiveSGA також можна було б покращити шляхом впровадження різних типів агентів-симбіонтів виконання, щоб агент-симбіонт, що приймає рішення, не був єдиним агентом, якого замінюють для зміни поведінки NPC. Впровадження різних видів агентів-симбіонтів виконання має призвести до того, що NPC демонструватимуть цікавішу поведінку. Однак це також означає, що знадобиться більше зусиль, щоб забезпечити відповідність змін у поведінці NPC бажаному збільшенню або зменшенню складності.

Отже, впровадження різних видів агентів-симбіонтів виконання вимагатиме більш інтелектуальної комунікаційної мембрани. Це означає, що комунікаційна мембрана повинна буде робити більше, ніж просто пов'язувати рівень складності з певним симбіонтом прийняття рішень або виконання. Комунікаційна мембрана також повинна мати механізм для з часом навчання, які комбінації

агентів-симбіонтів прийняття рішень та виконання відповідають певним рівням складності. Обмежена кількість агентів прийняття рішень може призвести до того, що гравець-людина перевершить усіх агентів прийняття рішень за рівнем майстерності. Як контрзахід, можна впровадити різні еволюційні підходи для розвитку існуючих алгоритмів прийняття рішень у більш складні. Прикладами підходів є генетичний алгоритм, генетичне програмування та програмування експресії генів.

Еволюційні обчислення - не єдина альтернатива. Традиційне машинне навчання (тобто навчання з підкріпленням) також може бути використане для покращення існуючих алгоритмів прийняття рішень, якщо гравець продовжує вдосконалюватися з часом. Вибір алгоритму навчання суттєво залежатиме від типу використовуваного алгоритму прийняття рішень.

У роботі AdaptiveSGA було представлено як модель для досягнення динамічного балансування складності на основі штучного інтелекту в адаптивній грі шляхом зміни моделі симбіотичного ігрового агента таким чином, щоб вона стала адаптивною. Крім того, було успішно розроблено прототип для перевірки концепції (DiskiSimulation) для демонстрації доцільності AdaptiveSGA.

3.4 Висновок по розділу

У третьому розділі було розглянуто реалізацію системи автоматичного балансування ігрових механік та оцінку її ефективності. Проведено проєктування та реалізацію системи адаптивного балансування, яка забезпечує автоматичне налаштування параметрів гри в реальному часі. Досліджено алгоритми оптимізації ігрових параметрів, що дозволяють підтримувати баланс між складністю гри та рівнем підготовки користувача. Також виконано оцінку ефективності розробленої системи та проведено експериментальну валідацію її роботи. Отримані результати підтверджують доцільність використання методів штучного інтелекту для автоматичного балансування ігрових механік і покращення користувацького досвіду.

ВИСНОВКИ

У ході виконання бакалаврської роботи було досліджено та реалізовано підходи до застосування штучного інтелекту для автоматичного балансування ігрових механік. Робота є комплексним дослідженням, спрямованим на створення адаптивної системи, здатної динамічно змінювати параметри ігрового процесу з урахуванням поведінки гравця. У процесі роботи проаналізовано сучасні підходи до балансування ігор, досліджено можливості використання алгоритмів машинного навчання та інтелектуальних систем прийняття рішень, а також розроблено прототип системи автоматичного балансування.

На початковому етапі було проведено аналіз предметної області, визначено основні проблеми традиційного балансування ігор, зокрема залежність від ручного налаштування параметрів, суб'єктивність рішень та складність підтримки оптимального рівня складності для різних типів гравців. Було сформовано функціональні та нефункціональні вимоги до системи автоматичного балансування, які включають адаптивність, продуктивність, масштабованість та здатність працювати в реальному часі.

У теоретичній частині роботи розглянуто основи застосування штучного інтелекту в іграх, концепції адаптивного ШІ та методи балансування ігрових механік. Проведено аналіз підходів, таких як динамічне регулювання складності (Dynamic Difficulty Adjustment), використання агентів та моделей поведінки гравця. Це дозволило визначити найбільш ефективні методи для подальшої реалізації системи.

У другому розділі було досліджено методи машинного навчання та інтелектуальні алгоритми, які можуть бути застосовані для автоматичного балансування. Зокрема, розглянуто підходи на основі підкріплювального навчання, класифікації поведінки гравців та оптимізації параметрів гри. Також було спроєктовано архітектуру системи автоматичного балансування, яка забезпечує взаємодію між ігровим середовищем, модулем збору даних та модулем прийняття рішень.

У практичній частині реалізовано прототип системи автоматичного балансування, який здатний змінювати параметри гри в реальному часі. Було розроблено алгоритми оптимізації ігрових характеристик, що враховують рівень навичок гравця, швидкість проходження та інші поведінкові фактори. Реалізація дозволила забезпечити адаптивність ігрового процесу та покращити взаємодію користувача з грою.

Проведено тестування та оцінку ефективності запропонованого підходу, що показали покращення балансу складності гри, підвищення залученості гравців та зниження ймовірності передчасного завершення гри. Було виявлено, що використання штучного інтелекту дозволяє більш точно адаптувати ігровий процес у порівнянні з традиційними методами.

Загалом, розроблена система відповідає поставленим цілям і демонструє ефективність застосування штучного інтелекту для автоматичного балансування ігрових механік. Основними перевагами запропонованого підходу є адаптивність, можливість роботи в реальному часі та орієнтація на індивідуальні особливості гравця. Водночас існують певні обмеження, зокрема залежність від якості зібраних даних, складність налаштування моделей машинного навчання та потреба в додаткових обчислювальних ресурсах.

У подальшому доцільним є розширення функціональності системи шляхом використання більш складних моделей штучного інтелекту, інтеграції з аналітичними платформами, а також застосування отриманих результатів у різних жанрах ігрових систем. Це дозволить підвищити універсальність підходу та його практичну цінність у сучасній індустрії відеоігор.

СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Russell, S., & Norvig, P. (2021). Artificial Intelligence: A Modern Approach. pearson.com. https://www.pearson.com

2. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. mitpress.mit.edu. https://www.deeplearningbook.org

3. Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction. mitpress.mit.edu. https://mitpress.mit.edu

4. Yannakakis, G. N., & Togelius, J. (2018). Artificial Intelligence and Games. springer.com. https://www.springer.com

5. Millington, I., & Funge, J. (2019). Artificial Intelligence for Games. crcpress.com. https://www.crcpress.com

6. Togelius, J., Yannakakis, G. N., Stanley, K. O., & Browne, C. (2011). Search-based procedural content generation. IEEE Transactions on Computational Intelligence and AI in Games, 3(3), 172–186. https://doi.org/10.1109/TCIAIG.2011.2148116

7. Hunicke, R. (2005). The case for dynamic difficulty adjustment in games. Proceedings of the ACM SIGCHI. https://doi.org/10.1145/1056808.1056815

8. Andrade, G., Ramalho, G., Santana, H., & Corruble, V. (2005). Dynamic game balancing: An evaluation of user satisfaction. AIIDE Conference. https://www.aaai.org

9. Pedersen, C., Togelius, J., & Yannakakis, G. (2009). Modeling player experience in Super Mario Bros. IEEE CIG. https://doi.org/10.1109/CIG.2009.5286482

10. Yannakakis, G. N., & Hallam, J. (2009). Real-time game adaptation for optimizing player satisfaction. IEEE Transactions on Computational Intelligence, 1(2), 121–133. https://doi.org/10.1109/TCIAIG.2009.2026503

11. Shaker, N., Togelius, J., & Nelson, M. (2016). Procedural Content Generation in Games. springer.com. https://www.springer.com

12. Justesen, N., Bontrager, P., Togelius, J., & Risi, S. (2019). Deep learning

for video game playing. IEEE Transactions on Games, 12(1), 1–20. https://doi.org/10.1109/TG.2019.2896986

13. Mnih, V., et al. (2015). Human-level control through deep reinforcement learning. Nature, 518, 529–533. https://doi.org/10.1038/nature14236

14. OpenAI. (2023). Reinforcement learning research. openai.com. https://openai.com/research

15. Unity Technologies. (2025). ML-Agents Toolkit Documentation. unity.com. https://unity.com/ml-agents

16. Unreal Engine. (2025). AI and gameplay systems documentation. unrealengine.com. https://docs.unrealengine.com

17. Microsoft. (2025). Machine Learning Documentation. learn.microsoft.com. https://learn.microsoft.com/en-us/azure/machine-learning/

18. Google DeepMind. (2024). Game AI research. deepmind.com. https://deepmind.com/research

19. Silver, D., et al. (2017). Mastering the game of Go without human knowledge. Nature, 550, 354–359. https://doi.org/10.1038/nature24270

20. Browne, C., et al. (2012). A survey of Monte Carlo tree search methods. IEEE Transactions on CI and AI in Games, 4(1), 1–43. https://doi.org/10.1109/TCIAIG.2012.2186810

21. Sommerville, I. (2020). Software Engineering. pearson.com. https://www.pearson.com

22. Bishop, C. M. (2006). Pattern Recognition and Machine Learning. springer.com. https://www.springer.com

23. Murphy, K. P. (2012). Machine Learning: A Probabilistic Perspective. mitpress.mit.edu. https://mitpress.mit.edu

24. Géron, A. (2022). Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow. oreilly.com. https://www.oreilly.com

25. TensorFlow Documentation. (2025). tensorflow.org. https://www.tensorflow.org

26. PyTorch Documentation. (2025). pytorch.org. https://pytorch.org/docs

27. Kaggle. (2025). Machine learning datasets and competitions. kaggle.com. https://www.kaggle.com

28. Steamworks Documentation. (2025). partner.steamgames.com. https://partner.steamgames.com/doc

29. Valve. (2023). Player behavior analytics in games. valve.com. https://www.valvesoftware.com

30. IBM. (2024). AI in gaming industry. ibm.com. https://www.ibm.com/artificial-intelligence

31. NVIDIA. (2025). AI for game development. developer.nvidia.com. https://developer.nvidia.com

32. GameDev.net. (2024). Game AI articles and resources. gamedev.net. https://www.gamedev.net

33. GDC (Game Developers Conference). (2025). AI in games presentations. gdcvault.com. https://www.gdcvault.com

34. Yannakakis, G. (2012). Game AI revisited. Proceedings of the 9th Conference on Computing Frontiers. https://doi.org/10.1145/2212908.2212927

35. Drachen, A., et al. (2013). Player modeling using telemetry data. IEEE Transactions on Games. https://doi.org/10.1109/TCIAIG.2013.2263709

36. Spronck, P., et al. (2006). Adaptive game AI with dynamic scripting. Machine Learning, 63, 217–248. https://doi.org/10.1007/s10994-006-6205-6

37. Hunicke, R., LeBlanc, M., & Zubek, R. (2004). MDA framework: Mechanics, Dynamics, Aesthetics. https://users.cs.northwestern.edu/~hunicke/MDA.pdf

38. Salen, K., & Zimmerman, E. (2004). Rules of Play: Game Design Fundamentals. mitpress.mit.edu. https://mitpress.mit.edu

39. Adams, E. (2014). Fundamentals of Game Design. newriders.com. https://www.pearson.com

40. Schell, J. (2020). The Art of Game Design: A Book of Lenses. crcpress.com. https://www.crcpress.com

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: " Застосування ШІ для автоматичного балансування ігрових механік"

Обсяг пояснювальної записки: 85 аркушів

Дата закінчення бакалаврської роботи 10 червня 2026р.

Підпис студента _____