Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Метод розробки програмних застосунків для симуляції та створення стратегій
Назва теми
"дилеми в`язня" на основі нейронних мереж

Рівень вищої освіти Другий (магістерський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.2301112.01.01.ПЗ

Виконав студент 2 курсу, група ІПЗм-23-1 Підпис Ярослав БАДЬОРА
Ім'я, ПРІЗВИЩЕ

Керівник кандидат тех. наук, доцент
Науковий ступінь, звання
Підпис Юрій ФОРКУН
Ім'я, ПРІЗВИЩЕ

Нормоконтролер к. пед. наук, доцент
Підпис Наталія Проворська
Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри
інженерії програмного забезпечення
Підпис Леонід БЕДРАТЮК
Ім'я, ПРІЗВИЩЕ

6 грудня 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Другий (магістерський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Л. П. Бедратюк

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бадьора Ярослав Михайлович
Прізвище, ім'я, по батькові здобувача

1. Тема роботи Метод розробки програмних застосунків для симуляції та створення стратегій «дилеми в'язня» на основі нейронних мереж.

Керівник роботи Форкун Ю. В. , кандидат технічних наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету 26.08.2024 р. № 60
2. Строк подання студентом роботи на кафедру 02.12.2024 р.
3. Вихідні дані до роботи Матеріали науково-дослідної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
1. Дослідження предметної області та постановка завдання
2. Розробка методу розробки програмних модулів для генерації динамічних стратегій вирішення дилеми в'язня
3. Технологія реалізації методу створення програмних модулів для генерації динамічних стратегій
4. Реалізація та тестування програмного засобу
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Презентаційні матеріали (слайди)

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Антиплагіат	Форкун Ю.В. канд. техніч. наук		
Нормоконтроль	Граворська Ж.Т. доцент, к. пед. наук		

7. Дата видачі завдання «2» вересня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Вивчення предметної області; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження;	02.09-10.09.2024	02.09-10.09.2024
2 Робота над розділом 1 кваліфікаційної роботи - вивчення літературних та Інтернет-джерел; аналіз відомих моделей, методів та засобів за темою роботи;	11.09-25.09.2024	
3 Робота над розділом 2 кваліфікаційної роботи - розробка моделей, методів та алгоритмів вирішення задачі; висновки до розділу	26.09-10.10.2024	
4 Робота над науковими статтями	11.10-30.10.2024	
5 Робота над розділом 3 кваліфікаційної роботи - розробка інформаційної технології вирішення задачі	11.10-26.10.2024	
6 Робота над розділом 4 кваліфікаційної роботи - програмна реалізація спроектованих рішень, результати експериментів та їх аналіз;	27.10-17.11.2024	
7 Попередній захист кваліфікаційної роботи	Листопад (згідно графіка)	
8 Узгодження постановки задачі, отриманих результатів та виснов- ків; оформлення пояснювальної записки та графічних матеріалів згідно вимог чинних стандартів	18.11-30.11.2024	
9 Перевірка роботи на наявність плагіату; нормоконтроль; брошу- рування пояснювальної записки; підготовка супровідних документів	02.12-04.12.2024	
10 Підготовка до захисту кваліфікаційної роботи	з 02.12.2024 р.	

Студент _____ Я. М. Бадьора
Підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ Ю. В. Форкун
Підпис Ім'я, ПРІЗВИЩЕ

РЕФЕРАТ

Тема дипломної роботи: «Метод розробки програмних застосунків для симуляції та утворення стратегій "дилеми в'язня" на основі нейронних мереж».

Автор: Бадьора Ярослав Михайлович.

Керівник: Форкун Юрій Вікторович.

Магістерська робота складається з: 116 с., 8 рис., 0 табл., 3 дод., 30 джерел.

ДИЛЕМА В'ЯЗНЯ, НЕЙРОННІ МЕРЕЖІ, ГЕНЕТИЧНІ АЛГОРИТМИ, СИМУЛЯЦІЯ, СТРАТЕГІЇ.

Об'єктом є процес розробки програмних застосунків стратегій поведінки в «дилемі в'язня».

Методи створення програмних засобів генерації стратегій її вирішення є предметом цього дослідження.

Мета дослідження: розробка методу розробки програмних застосунків симуляції стратегій "дилеми в'язня", що поєднує нейронні мережі для адаптації стратегій у реальному часі з генетичними алгоритмами для їх оптимізації.

Методи дослідження:

– математичне моделювання, формалізація, експеримент;

– використання сучасних інструментів розробки на основі штучного інтелекту;

– симуляція у програмному середовищі.

Під час дослідження було проаналізовано сучасні методи вирішення задач "дилеми в'язня", виявлено прогалини у статичних підходах і запропоновано адаптивний метод для симуляції стратегій. Визначено вимоги до нейронних мереж та реалізовано архітектуру на основі моделей LSTM і GRU.

Розроблений метод дозволить стратегій до змінних умов гри через використання нейронних мереж і генетичних алгоритмів. Реалізовано симуляційне середовище, яке дозволяє гнучко налаштовувати параметри гри та тестувати різні сценарії.

Програмна реалізація виконана за допомогою Python, TensorFlow та бібліотек

для генетичних алгоритмів. Тестування довело ефективність розробленого методу у підвищенні рівня співпраці серед агентів та гнучкості стратегій у динамічних умовах.

Результати можуть бути використані для вирішення задач моделювання взаємодій у складних багатокористувацьких системах, управління ресурсами та розробки колаборативних систем.

26.11.24.

Підпис

Дата

ABSTRACT

Topic of the thesis: "A method of developing software applications for simulation and formation of "prisoner's dilemma" strategies based on neural networks."

Author: Badyora Yaroslav Mykhailovych.

Head: Yuriy Viktorovych Forkun.

The master's thesis consists of: 115 p., 8 fig., 0 table, 3 appendix, 30 sources.

PRISONER'S DILEMMA, NEURAL NETWORKS, GENETIC ALGORITHMS, SIMULATION, STRATEGIES.

The object is the process of developing software applications of behavior strategies in the "prisoner's dilemma".

Methods of creating software tools for generating strategies for its solution are the subject of this study.

The purpose of the research: development of a method for developing software applications for simulating "prisoner's dilemma" strategies, which combines neural networks for real-time strategy adaptation with genetic algorithms for their optimization.

Research methods:

– mathematical modeling, formalization, experiment;

– use of modern development tools based on artificial intelligence;

– simulation in the software environment.

During the research, modern methods of solving "prisoner's dilemma" problems were analyzed, gaps in static approaches were identified, and an adaptive method for simulating strategies was proposed. The requirements for neural networks were determined and the architecture based on LSTM and GRU models was implemented.

The developed method will allow strategies for variable game conditions through the use of neural networks and genetic algorithms. A simulation environment has been implemented that allows you to flexibly adjust game parameters and test different scenarios.

The software implementation is made using Python, TensorFlow and libraries for genetic algorithms. Testing proved the effectiveness of the developed method in

increasing the level of cooperation among agents and the flexibility of strategies in dynamic conditions.

The results can be used to solve the problems of modeling interactions in complex multi-user systems, resource management and development of collaborative systems.

26.11.24.

Signature

Date

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ....8
ВСТУП....9
1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ..13
1.1. Аналіз предметної області, останніх досліджень та джерел....13
1.2. Аналіз існуючих методів та засобів забезпечення відмовостійкості програмних систем....16
1.3. Методологічний підхід до вирішення задачі дослідження стратегій вирішення "дилеми в'язня"....24
1.4. Висновки. Постановка задачі....26
2. РОЗРОБКА МЕТОДу розробки програмних модулів для генерації динамічних стратегій ВИРІШЕННЯ ДИЛЕМИ В'ЯЗНЯ....29
2.1. Розгляд концепцій вирішення задачі....29
2.2. Використані математичні моделі....33
2.3. Методика створення адаптивних стратегій. Розробка алгоритмів, заснованих на нейронних мережах і генетичних алгоритмах....35
2.4. Висновки....40
3. ТЕХНОЛОГІЯ РЕАЛІЗАЦІЇ МЕТОДУ СТВОРЕННЯ ПРОГРАМНИХ МОДУЛІВ ДЛЯ ГЕНЕРАЦІЇ ДИНАМІЧНИХ СТРАТЕГІЙ....43
3.1. Визначення функціональних і нефункціональних вимог....43
3.2. Реалізація модулів генерації та навчання....47
3.3. Висновки....49
4. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСОБУ....51
4.1. Програмна реалізація....51
4.2. Реалізовані модулі....52
4.3. Тестування та експериментальні результати....70
4.4. Висновки....76
ВИСНОВКИ....78
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ....80
ПРОГРАМНИЙ КОД ОСНОВНИХ МОДУЛІВ....83
КОПІЇ НАУКОВИХ ПУБЛІКАЦІЙ....99
ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ....107

ПЕРЕЛІК СКОРОЧЕНЬ

LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
AI	Artificial Intelligence
IN	Input Neuron (в контексті архітектури нейронних мереж)
GRASP	Generalized Reduced Algorithmic Search Process (у контексті алгоритмів)

ВСТУП

Моделювання взаємодій між агентами у складних системах є одною з актуальних проблем в сучасному науковому колі. Такі системи охоплюють широкий спектр явищ, від соціальних і економічних процесів до технічних інфраструктур і біологічних екосистем. У цих умовах часто виникає конфлікт між індивідуальними інтересами окремих учасників і колективним благом, яке є метою системи в цілому. У таких ситуаціях поведінка агентів стає надзвичайно складною для прогнозування, оскільки залежить не лише від їхніх власних мотивів, але й від зовнішнього середовища та дій інших учасників. Одним із найефективніших інструментів аналізу таких взаємодій є теорія ігор, бо дозволяє формалізувати ситуації, у яких учасники мають зробити вибір, результат якого залежить не лише від їхнього власного рішення, але й від стратегій інших гравців.

Класичною проблемою з теорії ігор є «дилема в'язня», яка часто відображає взаємодію людей в політичному світі або у тісних соціальних стосунках де потрібно обирати стратегію дій, як себе поводити: кооперуватись чи протидіяти.

Ця проблема є актуальна, зараз в тому числі, через велику кількість загострень відносин в геополітичній сфері нашого світу, бо саме ця дилема вивчає сферу вирішення подібних конфліктів, де обидві сторони повинні діяти не егоїстично, а тільки співпрацюючи можна добитись найвигідніших результатів.

Також все актуальнішою проблемою є поведінка алгоритмів штучного інтелекту автопілотів транспортних засобів, що потребують програмних рішень та їх досліджень для вирішення проблемних дорожніх ситуацій. Якраз стратегії розглянутої дилеми чудово моделюють такі відносини, а формування більш гнучких та динамічних стратегій допоможе в розв'язанні даної проблеми.

Крім того з розвитком гейм розробки все частіше спливає потреба в розробці штучного інтелекту для неігрових персонажів, які мають так чи інакше робити дії на основі своєї або загальної вигоди. Стратегії дій у «дилемі в'язня» чудово відображають таку поведінку, а запропонований метод дозволить створювати програмні модулі для генерації динамічних стратегій та їх використання у розробці

штучного інтелекту в сфері гейм-розробки та інших програмних рішеннях що потребують моделювання природних соціальних відносин.

Також, дослідження цієї теми дозволить краще розуміти природу поведінки у прикладних ситуаціях, в сфері соціології та політики та дозволить легше знаходити способи максимізації позитивних результатів у таких ситуаціях. Наприклад, класично, вивчення поведінки агентів у середовищі гри у «дилему вязня» дозволяє розуміти, аналізувати та розв'язувати складні справи у судовій системі, тобто використовуючи різні прийоми з цієї теорії можна допомогти ходу слідства, що робить цю тему актуальною на даний момент також.

Тому розробка програмних засобів що дозволить формувати стратегії котрі можуть краще моделювати поведінку реальних геополітичних відносин, вважається актуальною метою досліджень.

Сучасні дослідження дилеми в'язня фокусуються на адаптації вже існуючих алгоритмів поведінки агентів у цій «грі», мінімально їх модифікуючи щоб підстроїти їх під власні потреби. Традиційні підходи, засновані на статичних стратегіях, таких як "око за око" або "завжди зраджуй", демонструють обмежену ефективність, оскільки не враховують зміни в середовищі або здатність гравців навчатися з досвіду. Для подолання цих обмежень у сучасних дослідженнях мають використовуватися більш гнучкі засоби для генерації алгоритмів поведінки.

Метою цієї роботи є розробка методу створення програмних модулів для симуляції та створення адаптивних стратегій для вирішення дилеми в'язня на основі нейронних мереж і генетичних алгоритмів, що дозволить генерувати та краще розуміти динамічну поведінку агентів і ситуаціях до яких можна застосувати «дилему в'язня», дозволить генерувати природніші штучні моделі поведінки та наблизить розуміння поведінки сторін у політичних конфліктах.

Такий метод повинен адаптуватись до різних середовищ, параметрів та вхідних даних, бути гнучким та вміти достатньо стабільно відображати конкретну стратегію поведінки.

Запропонований підхід дозволяє подолати обмеження традиційних методів, забезпечуючи ефективне моделювання динамічної поведінки агентів. У рамках

роботи передбачено детальне вивчення існуючих підходів, розробку адаптивної архітектури нейронних мереж, інтеграцію генетичних алгоритмів, створення симуляційного середовища та оцінку ефективності запропонованого методу у порівнянні з традиційними статичними стратегіями. Результати дослідження матимуть практичне значення для моделювання соціальних, економічних та технологічних процесів.

Об'єктом є процес розробки програмних застосунків стратегій поведінки в «дилемі в'язня».

Методи створення програмних засобів генерації стратегій її вирішення є предметом цього дослідження.

Для досягнення мети цього дослідження у роботі використовувались як теоретичні, так і емпіричні методи, що дозволило забезпечити всебічний підхід до розробки методу симуляції та утворення адаптивних стратегій для дилеми в'язня. Абстрагування дозволило зосередитися на найбільш суттєвих аспектах задачі, відкидаючи несуттєві параметри, що не впливали на результативність моделі. Аналіз і синтез сприяли розбивці задачі на окремі компоненти: моделі нейронних мереж, правила взаємодії між агентами та еволюційні механізми генетичних алгоритмів. Після детального дослідження кожної частини вони були об'єднані в єдину систему, що забезпечує їхню ефективну взаємодію. Формалізація моделей у вигляді програмного коду дозволила перейти від теоретичних концепцій до практичної реалізації, забезпечивши можливість їхнього тестування та використання у симуляційному середовищі.

Наукова новизна:

— вперше використано моделі машинного навчання та генетичних алгоритмів для створення ПЗ по генерації стратегій вирішення «дилеми в'язня»

Цей підхід забезпечує агентам можливість адаптувати свою поведінку до динамічних умов гри завдяки використанню моделей LSTM, що дозволяють прогнозувати дії супротивників і враховувати минулий досвід. Інтеграція генетичних алгоритмів у модель сприяла оптимізації стратегій шляхом еволюційного відбору найкращих рішень, що дозволило досягти високого рівня

кооперації серед агентів. Розроблене симуляційне середовище забезпечило умови для дослідження взаємодії у масштабних системах, де поведінка кожного агента залежить від дій інших. Використання клітинних автоматів у контексті моделювання дилеми в'язня дозволило дослідити вплив локальних взаємодій на глобальну поведінку системи, виявляючи умови, за яких співпраця стає домінуючою стратегією.

Практична значущість роботи полягає у можливостях використання даного методу розробки програмних модулів в низці важливих сфер. Цей метод дозволить розробляти гнучкі програмні моделі стратегій поведінки у «дилемі в'язня», що можуть бути використані для моделювання поведінки у соціальних, політичних та економічних системах, управління транспортними мережами, розробки колаборативних систем штучного інтелекту, вивчення еволюційних процесів тощо.

1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Аналіз предметної області, останніх досліджень та джерел

Класичні розв'язання дилеми ув'язненого свідчать про прийняття статичних стратегій, таких як «око за око» або «ніколи не співпрацювати, завжди відмовлятися». Як зазначено в «Properties of Winning Iterated Prisoner's Dilemma Strategies»[1], вони демонструють свою застосовність у статичних середовищах, без урахування плинності середовища та можливих моделей поведінки гравців з часом. Це робить звичайні стратегії непридатними для моделювання ситуацій реального світу, де стратегії гравців є міжчасовими та розвиваються у відповідь на дії інших. Наприклад, дослідження «Analyzing the Impact of Strategic Behavior in an Evolutionary Learning Model Using a Genetic Algorithm» [3] показують, що в складних системах, які включають довгострокові взаємодії, статичні стратегії з самого початку не обіцяють бажаних рішень, отже, причина, чому існує потреба в нових підходах до стратегічного моделювання.

Застосування нейронних мереж у модифікації стратегій дилеми ув'язненого є новим підходом, оскільки воно формує такі типи стретагій, які можуть навчатися та змінювати свою поведінку відповідно до попередніх взаємодій. Вони також можуть з'ясувати складні та складні нелінійні зв'язки між рішеннями, прийнятими гравцями, та результатами таких рішень [1]. Подібним чином результати дослідження в статті «Optimal Tag-Based Cooperation Control for the "Prisoner's Dilemma"» [4] демонструють, що динамічні стратегії здатні досягти більшої гнучкості та адаптивних реакцій у поведінкових стратегіях гравців, ніж звичайні методи, що є вирішальним в умовах високої динаміки та високої невизначеності.

Інтеграція нейронних мереж і генетичних алгоритмів видається ефективною з точки зору оптимізації стратегії в дилемі ув'язненого. Генетичні алгоритми дозволяють шукати оптимальні структури нейронних мереж і параметри їх навчання, а також розвивати стратегії в конкурентному середовищі. Так, у дослідженнях «Optimal Tag-Based Cooperation Control for the "Prisoner's Dilemma"»

та «A symbiosis between cellular automata and genetic algorithms» [4] наголошується на застосуванні генетичних алгоритмів для еволюції кооперативних стратегій у повторюваній дилемі ув'язненого «всі або жодна», що дозволяє знайти більш надійні стратегії, які можуть пристосовуватися до змін середовища або інших гравців. . Це дає гарну можливість вивчити еволюцію довгострокової стратегії, особливо вплив на неї оточуючих та інших агентів.

Ще однією важливою перевагою використання нейронних мереж є їх здатність моделювати складні соціальні та економічні взаємодії, де поведінка агентів залежить не лише від поточних дій, але й від минулого досвіду та очікуваних майбутніх змін. Сучасні дослідження [7] вивчають алгоритми онлайн-навчання, які дозволяють моделям адаптуватися до мінливих умов під час гри, змінюючи свої стратегії у відповідь на дії інших агентів і зміни середовища. Це дозволяє створювати точніші моделі взаємодії в соціальних системах, де гравці можуть змінювати свої стратегії в залежності від умов гри.

Дослідження також підкреслюють важливість адаптивних стратегій моделювання дилеми ув'язненого, особливо в ситуаціях, коли правила гри змінюються або коли інформація про дії інших гравців обмежена, що підтверджується дослідженнями [6, 7]. Нейронні мережі можна навчити та адаптувати до нових умов, враховуючи не лише поточні, але й майбутні події, що дозволяє їм приймати більш оптимальні рішення у складних та невизначених середовищах.

У практичному контексті використання нейронних мереж для моделювання адаптивної поведінки в дилемі ув'язненого може бути корисним для вирішення проблем колективного управління ресурсами, моделювання економічного співробітництва та прийняття рішень в умовах невизначеності. Дослідження [5] вказують на те, що адаптивні стратегії, засновані на нейронних мережах, можуть підвищити ефективність колективного прийняття рішень за таких обставин, враховуючи зміни в моделях поведінки та міжагентній взаємодії. Це дозволяє розробляти нові способи вирішення завдань у складних системах, де зміна поведінки гравців може критично вплинути на результат взаємодії.

Важливим аспектом цього дослідження є розуміння різних факторів, таких як соціальні норми чи винагороди, які можуть вплинути на еволюційні стратегії в динамічних середовищах. Дослідження [2] показує, що моделювання адаптивної поведінки в таких системах може допомогти зрозуміти, як різні стратегії співпраці та зради розвиваються в умовах конкуренції; що є важливим для розробки ефективних моделей управління в складних соціально-економічних системах. Зокрема, врахування змін у винагороді за співпрацю чи відмову дозволяє точніше моделювати поведінку агента та ідентифікувати, які стратегії є найбільш надійними в довгостроковій перспективі.

Важливим аспектом розробки систем штучного інтелекту (ШІ) для вирішення повторюваної дилеми в'язня є розробка нових стратегій, які враховують минулі взаємодії. Наприклад, концепція пам'яті-один, яка використовується при моделюванні взаємодії людини з комп'ютером [9], дозволяє агентам враховувати минулі кроки під час прийняття рішень. Це дає їм змогу будувати системи, які передбачають поведінку опонента на основі минулого досвіду, тим самим підвищуючи ефективність кооперативних стратегій у довгостроковій перспективі. Такий підхід демонструє значні переваги порівняно зі звичайними моделями без урахування історії попередніх дій.

Поєднання нейронних мереж з еволюційними алгоритмами, такими як генетичні, є ще одним перспективним підходом. Наприклад, дослідження показали, що генетичні алгоритми разом з нейронною мережею дозволяють розробляти адаптивні стратегії в мінливому середовищі, а також здатні розвиватися, що обов'язково важливо для стабільної співпраці між агентами [3]. Крім того, такий підхід дозволяє агентам швидко знаходити оптимальні стратегії в нових умовах, реагуючи на зміни в поведінці опонента.

Адаптивність стратегій також забезпечується завдяки онлайн-навчанню, що дозволяє агентам змінювати свої дії в реальному часі. Використання підкріплювального навчання у поєднанні з нейронними мережами відкриває нові можливості для створення агентів, які можуть не лише ефективно імітувати людську поведінку, але й адаптуватися до її змін у динамічних умовах гри [7]. Це

особливо важливо у ситуаціях, де поведінка інших гравців може бути непередбачуваною, що вимагає швидкої реакції та корекції власної стратегії.

Online-навчання забезпечує адаптацію стратегій, що дозволяє агентам змінювати свої дії в режимі реального часу. Використання навчання з підкріпленням у поєднанні з нейронними мережами відкриває нові шляхи для створення агентів, які можуть не тільки ефективно моделювати поведінку людини, але й адаптуватися до змін у динамічному ігровому середовищі [7]. Це особливо важливо в ситуаціях, коли поведінка інших гравців непередбачувана і вимагає швидкої реакції та коригування власної стратегії.

Використовуючи нейронні мережі для моделювання їхніх стратегій, можна вивести приховані стимули, що стоять за цими діями, а також дозволити їм швидко адаптуватися в новому середовищі. Ці моделі здатні використовувати великі історичні набори даних взаємодії завдяки методам штучного інтелекту машинного навчання, що потенційно дає змогу прогнозувати майбутні дії опонентів на основі попередніх взаємодій між кількома агентами [1]. Це допомагає вдосконалювати розроблені стратегії, які можуть витримати будь-які мінливі обставини гри.

Таким чином, останні дослідження показують, що нейронні мережі, інтегровані з еволюційними методами, такими як генетичні алгоритми та клітинні автомати, забезпечують потужні засоби вирішення дилеми ув’язненого. Вони створюють адаптивні моделі, які ефективно реагують на зміни в поведінці інших акторів і забезпечують стабільну співпрацю в динамічних середовищах [7][9]. Цей підхід відкриває нові горизонти для досліджень теорії ігор і ШІ, спрямованих на розробку надійних і гнучких стратегій для вирішення складних проблем у багатоагентних системах.

1.2. Аналіз існуючих методів та засобів забезпечення відмовостійкості програмних систем

Дилема в’язня у повторюваних сценаріях є складною та багатогранною моделлю, яка дозволяє досліджувати динаміку адаптивної поведінки агентів у

системах із взаємозалежними діями. На відміну від одноразових ігор, де кожен гравець робить вибір лише один раз, у повторюваних іграх кожна нова гра дозволяє учасникам переглядати свої стратегії, враховуючи попередні дії своїх супротивників. Це додає до гри елемент навчання та довгострокової адаптації, що робить повторювану дилему в'язня надзвичайно актуальною для моделювання реальних систем. У соціальних процесах, економічних відносинах і навіть біологічних системах взаємодія між агентами часто відбувається не одноразово, а в рамках довгострокових відносин. Повторюваність створює умови, за яких довіра, співпраця чи конфлікти можуть виникати і змінюватися залежно від накопиченого досвіду, що дозволяє дослідникам вивчати складні поведінкові патерни у тривалих взаємодіях [1].

У базових дослідженнях повторюваної дилеми в'язня стратегія "око за око" є однією з найвідоміших і найчастіше аналізованих. Вона полягає у простому принципі взаємності: гравець починає зі співпраці, але в наступних раундах повторює дії супротивника. Якщо супротивник співпрацює, гравець продовжує співпрацю, але якщо зраджує, гравець відповідає зрадою. Така стратегія добре працює у стабільних умовах, оскільки створює сильний стимул для співпраці: супротивник знає, що кожна його дія матиме наслідки. Однак стратегія "око за око" має обмеження, зокрема її вразливість до помилок або шуму у грі. Якщо через випадкову помилку гравець неправильно інтерпретує дію супротивника, це може призвести до серії взаємних зрад, які зруйнують співпрацю. Цей недолік став стимулом для розробки складніших адаптивних стратегій [3].

Прикладом розробки реалізацій класичних стратегій можна навести одну з найпростіших, але в той же час ефективних стратегій — "око за око" (Tit-for-Tat) [1, 2, 8]. Її ключовий принцип — взаємність, яка сприяє довготривалій кооперації. Проте, як виявилося під час подальших експериментів, стратегія має вразливості. Наприклад, у середовищах із частими помилковими діями вона може потрапити у цикл помсти, коли обидва гравці безупинно карають один одного.

Щоб подолати ці недоліки, з'явилися вдосконалені версії. Наприклад, "щедра око за око" (Generous Tit-for-Tat) включає імовірність пробачення. Ця стратегія

дозволяє уникнути нескінченних циклів помсти. Однак щедрість робить її вразливою до експлуатації, особливо якщо опоненти використовують агресивні стратегії на кшталт "завжди зраджуй" (Always Defect).

Паралельно з цим виникли і інші прості стратегії, такі як "завжди кооперуй" (Always Cooperate) та "завжди зраджуй" (Always Defect). Перша підходить для високотрестових середовищ, але її слабкість полягає у надмірній відкритості до експлуатації. Друга, навпаки, максимізує вигоду в короткостроковій перспективі, але не дозволяє досягти стабільної кооперації. Наприклад, у дослідженнях Montero-Porras та інших було показано, що Always Defect зазнає значних втрат у середовищах, де переважають стратегії, спрямовані на підтримання співпраці [5].

Також було проаналізовано концепцію Sink State стратегій, що базуються на ідеї створення станів, із яких неможливо повернутися до кооперації після досягнення певного рівня зради. Було досліджено що вони моделюють ситуацію, коли гравець переходить до "безвихідного" стану у відповідь на агресивну поведінку опонента. Така концепція була вперше запропонована у контексті теорії ігор для вивчення наслідків необоротних рішень у повторюваних іграх.

У таких стратегіях стан "поглинання" (англ. sink state) настає, коли гравець накопичує достатню кількість негативного досвіду з опонентом. Наприклад, якщо опонент кілька разів поспіль вибирає зраду, стратегія переходить у стан, у якому всі наступні ходи будуть виключно зрадницькими, незалежно від подальшої поведінки опонента.

Sink State стратегії відзначаються високою стійкістю до агресивних опонентів, таких як Always Defect. Вони створюють механізм покарання для опонентів, які не дотримуються принципів кооперації. Однак головним недоліком є їх нездатність відновлювати кооперацію, навіть якщо опонент змінює свою поведінку. Цей аспект робить їх менш ефективними у змішаних популяціях, де існують як кооперативні, так і зрадницькі гравці.

Rui Dong та його колеги застосували Sink State стратегії [4] для моделювання поведінки у великих популяціях, де гравці мають обмежені ресурси та ризик потрапляння у стан дефіциту. Їхній підхід показав, що такі стратегії ефективні для

захисту власних ресурсів у середовищах із великою кількістю зрадницьких опонентів.

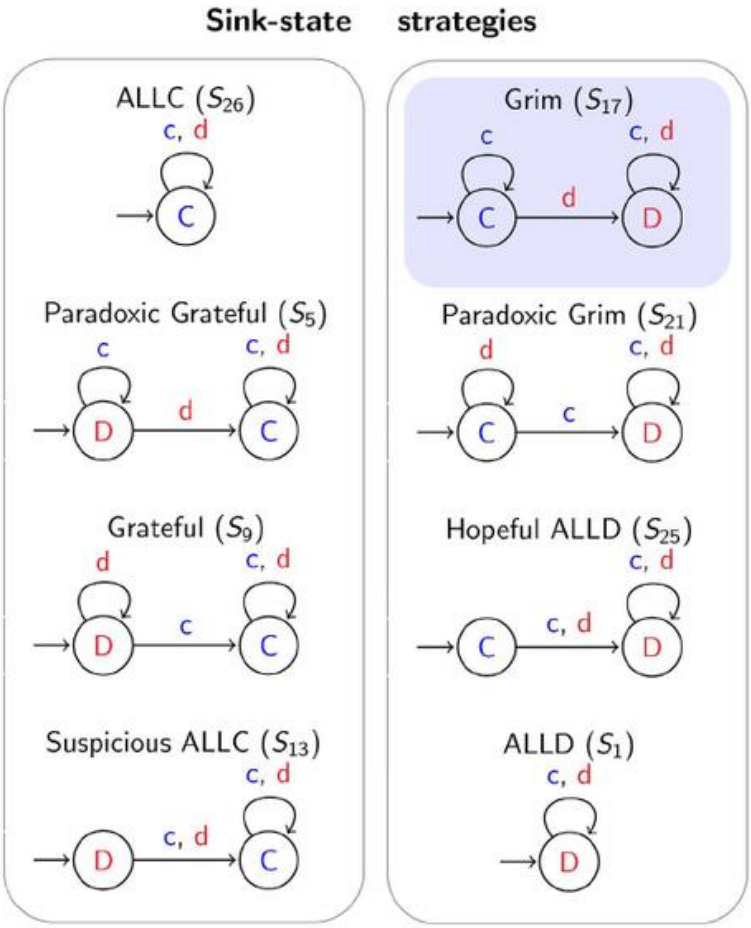


Рисунок 2.1.1 — Схеми роботи Sink State стратегій

Ще одним прикладом може бути концепція Suspicious стратегій виникла з прагнення зменшити ризики, пов’язані з початковим довірливим ставленням до опонента. Їхня головна ідея полягає у тому, щоб починати гру з більш обережного ставлення, припускаючи можливість зради з боку опонента, і лише поступово підвищувати рівень довіри.

На відміну від класичних стратегій, таких як Tit-for-Tat, Suspicious стратегії розпочинають взаємодію з першого ходу з обережного рішення, наприклад, зради або нейтрального ходу. Цей підхід дозволяє мінімізувати втрати у випадку агресивного опонента, але водночас не блокує можливості для встановлення довготривалої кооперації.

Suspicious стратегії є гнучкими у середовищах із великою кількістю

невідомих опонентів. Вони демонструють високу стійкість у перших циклах взаємодії, коли поведінка іншого гравця ще не зрозуміла. Однак така стратегія може створювати враження агресивності, що знижує ймовірність встановлення кооперації з кооперативними опонентами.

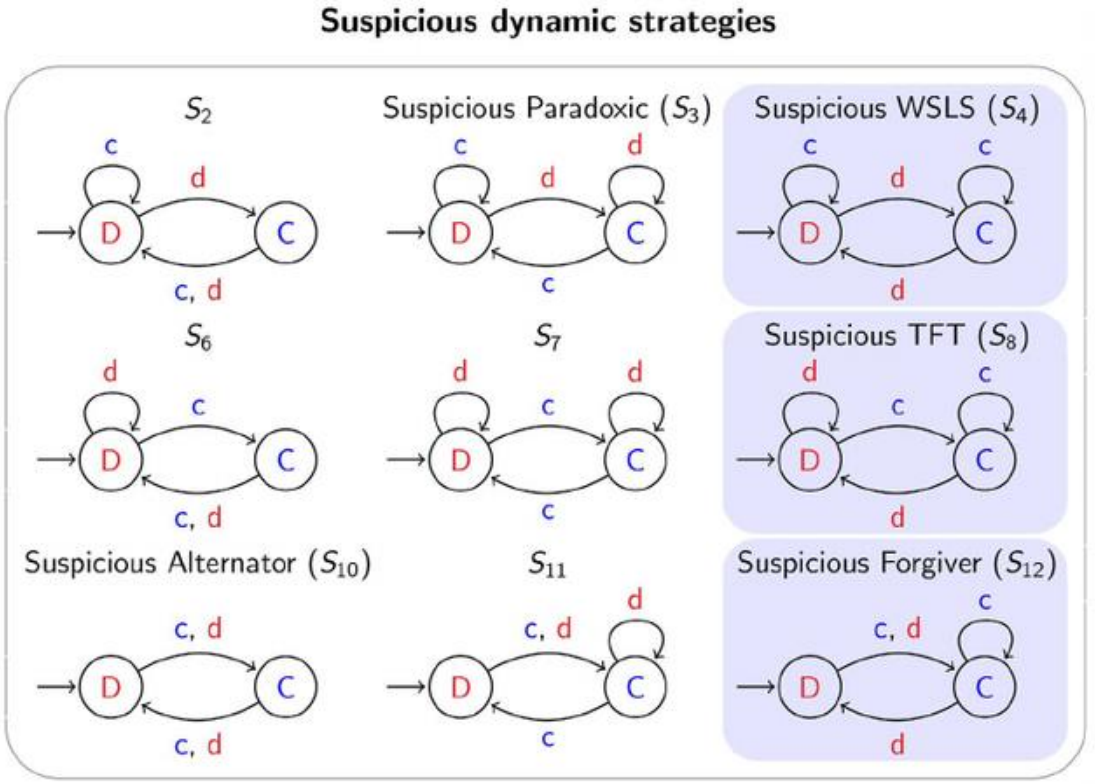


Рисунок 2.1.2 — Схеми роботи Suspicious динамічних стратегій

Suspicious стратегії можуть використовуватися у гетерогенних середовищах для виявлення агресивних гравців та їх ізоляції. Зокрема, стратегія "Підозріливий початок із перевіркою" демонструє ефективність у встановленні кооперації після короткого періоду тестових ходів.

Також було розглянуто концепцію Hopeful стратегій, як приклад методу розробки програмних імплементацій стратегій. Він базується на оптимістичному підході до взаємодії. Вони передбачають, що всі опоненти спочатку мають потенціал для кооперації, і намагаються побудувати співпрацю, навіть у відповідь на агресивні дії.

Hopeful стратегії починають гру з кооперації та продовжують відповідати кооперативно, доки рівень агресивної поведінки опонента не перевищує певного

порогу. Їхня мета — створити середовище, у якому навіть агресивні опоненти будуть змушені пристосуватися до кооперації для отримання вигоди.

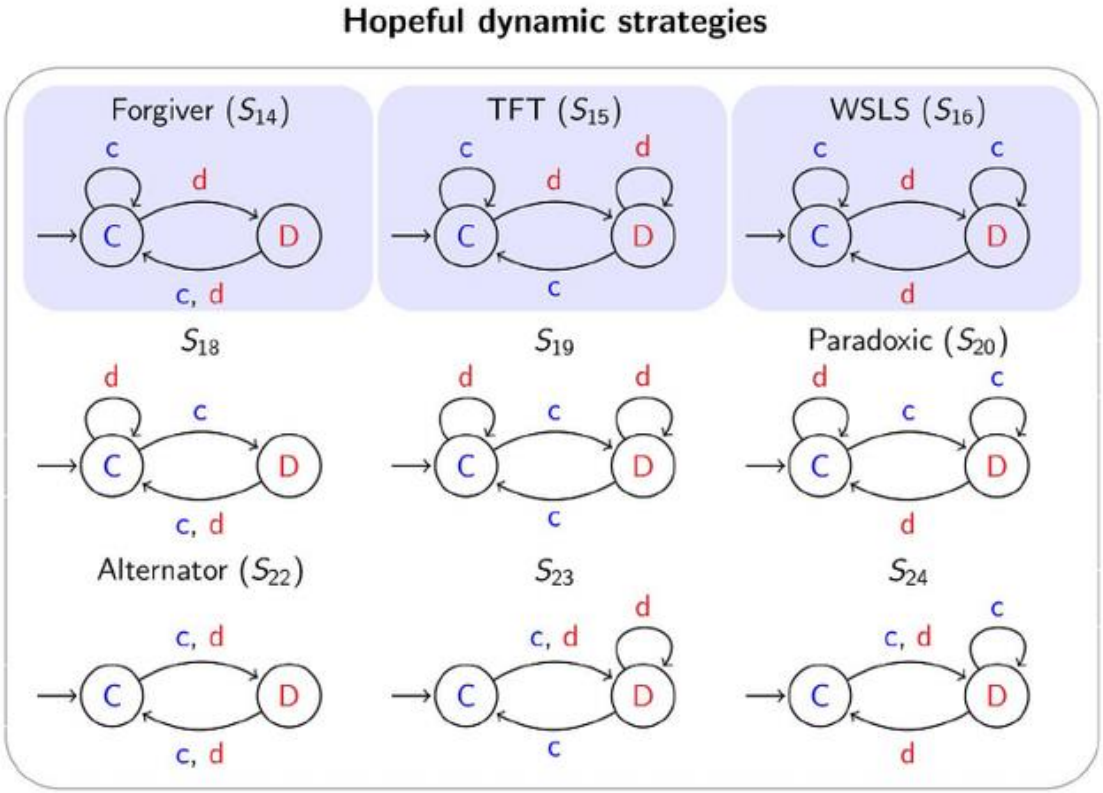


Рисунок 2.1.3 — Схеми роботи Hopeful динамічних стратегій

Hopeful стратегії демонструють високий рівень адаптації у кооперативних середовищах, сприяючи створенню стабільних союзів. Водночас вони є вразливими до стратегій на зразок Always Defect, які повністю ігнорують кооперативні спроби.

Hopeful стратегії у контексті еволюційних моделей сприяють формуванню кооперативних кластерів навіть у ворожих середовищах. Одним із прикладів є стратегія "Тривала кооперація з адаптивним покаранням", яка підтримує кооперацію, але карає опонентів, які систематично зраджують.

Складніші підходи виникли з появою ймовірнісних моделей, що враховують варіативність дій опонента. Наприклад, стратегія "куленепробивна око за око" (Tit-for-Two-Tats) була розроблена для роботи у середовищах з високою частотою помилок. Вона ігнорує одну зраду, але реагує після другої. Такий підхід дозволяє зберігати кооперацію, але його недоліком є схильність до експлуатації стратегій,

які систематично обирають зраду кожні два ходи [3].

Іншим підходом стала імовірнісна кооперація (Probabilistic Cooperation). Ця стратегія визначає ймовірність співпраці залежно від історії дій опонента. У роботах Ferraz і Pitz показано, що ця модель дозволяє досягти балансу між ризиком та вигодою, особливо в середовищах із великою кількістю гравців [3]. Проте складність реалізації й обчислювальна вимогливість роблять її менш ефективною у реальних умовах.

Цікавою варіацією імовірнісних стратегій є Win-Stay, Lose-Shift, запропонована Nowak і Sigmund у 1990-х роках. Ця стратегія змінює свою поведінку залежно від успішності попереднього ходу: якщо попередній результат був вигідним, стратегія повторює свою дію; у разі провалу — змінює її. Вона виявилася ефективною в середовищах із невеликою кількістю учасників, але зазнавала труднощів у складніших сценаріях [4].

Сучасні підходи до моделювання стратегій дедалі більше покладаються на адаптивність і здатність до навчання. Одним із найперспективніших методів є використання штучних нейронних мереж. Завдяки їхній здатності аналізувати послідовності даних і виявляти закономірності, нейронні мережі дозволяють агентам вивчати поведінку супротивників і адаптувати свої стратегії відповідно до змін у середовищі. Архітектури LSTM та GRU є особливо ефективними у цьому контексті, оскільки вони враховують довгострокові залежності у даних і можуть запам’ятовувати важливі аспекти минулої взаємодії. Наприклад, агент, який використовує нейронну мережу, може виявити, що супротивник має тенденцію змінювати свою поведінку через певну кількість раундів, і відповідно налаштувати свою стратегію. Це забезпечує значну перевагу у порівнянні з класичними статичними методами, які не враховують динаміку поведінки супротивників [6].

Генетичні алгоритми додають до процесу моделювання елемент еволюційного вдосконалення. У контексті дилеми в’язня ці алгоритми дозволяють створювати агентів, які адаптуються до середовища через механізми мутації, схрещування та відбору. Кожна стратегія в популяції агентів може бути закодована у вигляді хромосоми, яка представляє набір правил чи параметрів. Успішні

стратегії, які демонструють високу ефективність у середовищі, передаються наступним поколінням, тоді як менш успішні поступово усуваються. Це дозволяє системі еволюціонувати, знаходячи оптимальні або наближені до оптимальних стратегії. Генетичні алгоритми також дозволяють досліджувати, як взаємодія між різними стратегіями впливає на динаміку гри, наприклад, як кооперативні агенти можуть захоплювати систему навіть у середовищі, де переважає зрада [3, 10].

Клітинні автомати, зі свого боку, забезпечують ще один підхід до моделювання взаємодії агентів у дилемі в'язня. Такі моделі дозволяють вивчати, як локальні патерни поведінки можуть впливати на глобальні результати. Вони також є корисними для аналізу стійкості кооперації у системах із високим рівнем зради або шуму [2, 10].

У дослідженні «Inferring strategies from observations in long iterated Prisoner's dilemma experiments»[5] також піднімається тема про гібридні стратегії, які поєднують переваги кількох підходів, демонструють ще більший потенціал. Наприклад, стратегія, що комбінує принципи "око за око" з адаптивними алгоритмами на основі нейронних мереж, дозволяє починати зі співпраці, але поступово адаптуватися до змін у поведінці супротивників. Це особливо корисно у динамічних середовищах, де умови гри змінюються, або коли супротивники використовують складні стратегії, які неможливо ефективно передбачити за допомогою статичних методів. Іншим перспективним напрямком є використання тегів або міток, які дозволяють агентам оцінювати історію взаємодії з іншими гравцями. Це забезпечує додатковий механізм для підтримки кооперації, оскільки агенти можуть уникати співпраці з "зрадниками" і зосереджувати свої ресурси на взаємодії з надійними партнерами

Адаптивні стратегії також знаходять застосування у середовищах із високим рівнем невизначеності. Наприклад, у системах, де агенти мають обмежену інформацію про дії супротивників або середовище є непередбачуваним, використання алгоритмів машинного навчання та еволюційних методів дозволяє забезпечити гнучкість і ефективність поведінки. Такі підходи не лише дозволяють агентам адаптуватися до нових умов, але й дають змогу розробляти стратегії, які є

стійкими до шуму та зовнішніх змін як це показано у досліджені [7].

Таким чином, дослідження стратегій у повторюваній дилемі в’язня демонструє важливість використання сучасних технологій для аналізу та моделювання взаємодій у складних системах. Поєднання нейронних мереж, генетичних алгоритмів і клітинних автоматів відкриває нові можливості для створення ефективних стратегій, які враховують складність середовища, адаптивність агентів та їхню здатність до навчання. Це дозволяє не лише досягати кооперації у динамічних умовах, але й аналізувати поведінкові патерни, які можуть мати практичне застосування у соціальних, економічних та технічних системах.

1.3. Методологічний підхід до вирішення задачі дослідження стратегій вирішення "дилеми в’язня"

Методологічний підхід до вирішення задачі дослідження стратегій вирішення "дилеми в’язня" базується на поєднанні сучасних інструментів і підходів, таких як нейронні мережі, генетичні алгоритми та клітинні автомати. Це інтегроване використання інструментів дозволяє створювати інноваційні моделі, здатні до адаптації, прогнозування поведінки агентів і оптимізації стратегій в складних середовищах з множинними взаємодіями. Такий підхід не лише враховує сучасні тенденції у галузі теорії ігор і штучного інтелекту, але й робить вагомий внесок у вирішення практичних завдань моделювання кооперації та конфліктів у реальному світі.

У випадку повторюваних ітерацій дилеми, відомих як "ітеративна дилема в’язня", складність зростає через необхідність аналізу та адаптації до змінюваної поведінки супротивників. Ця модель дозволяє глибше вивчити, як агенти пристосовуються до середовища, що динамічно змінюється, і які стратегії є найефективнішими в умовах невизначеності [1, 3].

Нейронні мережі є основою можливого розвитку методів створення динамічних стратегій. А особливе значення мають рекурентні архітектури, такі як LSTM (Long Short-Term Memory), бо вони здатні зберігати інформацію про

довгострокові залежності в послідовностях даних [4]. Це дозволить агентам використовувати історію попередніх взаємодій для прийняття оптимальних рішень. Якщо нейронна мережа аналізує, що супротивник після кількох раундів співпраці починає діяти дефекційно, вона може передбачити цю тенденцію і відповідним чином коригувати свою стратегію. В результаті, досягається не лише короткострокова вигода, але й зберігається ефективність у довгостроковій перспективі. Важливо зазначити, що адаптивність, яку забезпечують LSTM, робить їх ідеальними для використання в ігрових сценаріях, де поведінка супротивника може змінюватися залежно від умов гри.

Також було розглянуто ще один концепт, який потенційно допоможе збільшити гнучкість та витривалість стратегій вирішення дилеми - генетичні алгоритми, оскільки вони забезпечують пошук оптимальних стратегій через механізми еволюції. Так як ці алгоритми базуються на принципах природного відбору, включаючи такі процеси, як селекція, кросовери та мутації, вони можуть забезпечити постійне вдосконалення стратегій, навіть у складних і динамічних середовищах, що підтверджується в проаналізованих дослідженнях [3, 7]. У контексті ітеративної дилеми в’язня генетичні алгоритми дозволяють оптимізувати параметри нейронних мереж та аналізувати, які поєднання стратегій є найефективнішими для досягнення стійкої кооперації або максимізації вигоди в конфліктних ситуаціях.

Вивчивши роботу клітинних автоматів було знайдено що ця сфера також може допомогти в розвитку стратегій ц. Вони дозволяють вивчати, як локальні дії кожного агента впливають на загальну поведінку системи. Це досягається завдяки структурі клітинних автоматів, де кожна клітинка представляє агента, а її стан залежить від стану сусідніх клітин. У контексті дилеми в’язня клітинні автомати можуть бути використані для вивчення динаміки взаємодій, де одна стратегія може спричинити ланцюгову реакцію змін у поведінці інших агентів. Наприклад, модель може аналізувати, як відмова одного агента від співпраці впливає на рівень кооперації в спільноті, і навпаки, як підтримка кооперації декількома агентами може стабілізувати систему [10].

Симуляційне середовище, що використовується для дослідження ітеративної дилеми в'язня, дозволяє створювати сценарії з різними умовами, включаючи обмеження інформації та ресурси агентів. Це забезпечує можливість тестування стратегій у реалістичних умовах, де існує невизначеність і можливість випадкових змін у середовищі. Наприклад, у сценаріях, де агенти мають обмежену інформацію про попередню поведінку інших учасників, можна досліджувати, які моделі здатні забезпечити ефективну адаптацію до таких обмежень. Це важливо для оцінки стійкості стратегій у реальних системах, таких як ринкові взаємодії або соціальні мережі [2, 8].

Інтеграція онлайн-навчання в нейронні мережі є ще одним важливим аспектом методології. Онлайн-навчання дозволяє моделям швидко адаптувати свої параметри в режимі реального часу на основі нових даних. Це особливо корисно в ситуаціях, де поведінка супротивника або умови середовища змінюються динамічно. Наприклад, якщо модель помічає, що певна стратегія більше не є ефективною через зміну поведінки супротивника, вона може миттєво перебудувати свою стратегію, мінімізуючи збитки та підтримуючи конкурентоспроможність. Цей підхід робить моделі не лише ефективними, але й універсальними для застосування в різних середовищах [7].

Практичне застосування запропонованої методології охоплює широкий спектр завдань, починаючи від моделювання взаємодій між конкуруючими компаніями на ринку до оптимізації роботи автономних систем. Наприклад, у транспортних мережах моделі можуть використовуватися для координації дій автономних транспортних засобів, забезпечуючи ефективний розподіл ресурсів і уникнення конфліктів. У соціальних науках [5] цей підхід може допомогти зрозуміти механізми формування кооперації між різними групами людей, що має важливе значення для розробки політики та управління спільнотами.

1.4. Висновки. Постановка задачі

Отже було проаналізовано адаптивні підходи до розв'язання дилеми в'язня,

які дозволяють агентам змінювати свої стратегії залежно від змін середовища і взаємодій з іншими учасниками. Використання нейронних мереж та генетичних алгоритмів дало змогу створити динамічні стратегії, що враховують історичні дані та здатні швидко адаптуватися до нових умов гри [1, 4]. Це має значний потенціал у моделюванні складних багатогранних взаємодій у соціальних, економічних та біологічних системах.

Класичні статичні стратегії, такі як "око за око" або "завжди зраджуй", виявили свою обмеженість у змінних середовищах, де поведінка гравців та умови гри можуть змінюватися. Статичний підхід недостатньо враховує складність реальних ситуацій, що робить його менш ефективним для тривалих і динамічних взаємодій [2, 5]. У цьому контексті дослідження довели, що інтеграція адаптивних моделей, таких як нейронні мережі, дозволяє досягти значних покращень у прогнозуванні поведінки опонентів та вдосконаленні кооперативних стратегій [6, 7].

Використання нейронних мереж для розробки динамічних стратегій забезпечує гнучкість у моделюванні складних нелінійних залежностей між діями гравців і результатами їхніх взаємодій. Завдяки цим моделям можливо не лише ефективніше адаптувати стратегії до змін, але й виявляти приховані закономірності в поведінці опонентів, що раніше були недоступні для традиційних методів. Це дозволяє агентам краще реагувати на нові виклики та досягати більшої ефективності у складних сценаріях, таких як багатокористувацькі середовища з високим ступенем невизначеності [3, 9].

Генетичні алгоритми виконують важливу роль в оптимізації параметрів нейронних мереж і розвитку стратегій у конкурентному середовищі. Цей підхід дозволяє моделювати еволюційні процеси у багатоагентних системах, знаходячи оптимальні стратегії для кооперації або конкуренції. Наприклад, використання генетичних алгоритмів для адаптації стратегій у повторюваних ігрових ситуаціях забезпечує більш стійку поведінку гравців навіть за умов динамічних змін середовища [4, 10].

На основі аналізу було виявлено низку актуальних проблем, таких як

необхідність у розробці стратегій, здатних адаптуватися до змінних умов гри, та недостатня точність традиційних методів моделювання. Для вирішення цих проблем запропоновано підхід, який інтегрує нейронні мережі та генетичні алгоритми, дозволяючи створювати стратегії, що не лише адаптуються до змін середовища, але й здатні враховувати довгострокову динаміку взаємодій між гравцями [8].

Відповідно до поставленої мети визначено такі задачі:

— аналіз класичних і сучасних стратегій вирішення дилеми в'язня;

— розробка адаптивних стратегій на основі нейронних мереж і генетичних алгоритмів;

— створення симуляційного середовища для тестування розроблених стратегій;

— порівняння ефективності адаптивних і класичних стратегій у динамічних умовах гри.

Таким чином, методологічний підхід до вирішення ітеративної дилеми в'язня базується на поєднанні адаптивності, оптимізації та моделювання. На основі проведеного аналізу, вирішено що використання нейронних мереж та генетичних алгоритмів допоможе створити стратегії що можуть показувати природню поведінку в умовах невизначеності. Ці методи забезпечують як теоретичну новизну, так і практичну застосовність у різних галузях. Такий підхід сприяє розвитку нових моделей кооперації та конфлікту, що є актуальними для вирішення широкого спектра завдань у сучасному світі.

Наступним етапом роботи стане вдосконалення адаптивних алгоритмів, їхня оптимізація та розширення для більш складних сценаріїв.

2. РОЗРОБКА МЕТОДУ РОЗРОБКИ ПРОГРАМНИХ МОДУЛІВ ДЛЯ ГЕНЕРАЦІЇ ДИНАМІЧНИХ СТРАТЕГІЙ ВИРІШЕННЯ ДИЛЕМИ В'ЯЗНЯ

2.1. Розгляд концепцій вирішення задачі

В ході проектування засобів для генерації гнучких стратегій вирішення «дилеми в'язня» було виділено декілька головних підходів. Метою використання цих підходів є створення «динамічних», тобто гнучких стратегій з природньою поведінкою що залежить від середовища виконання. Метод складається з набору концепцій та рішень, що дозволяють створювати програмні засоби для генерації таких стратегій.

Концепція еволюційних стратегій:

Ця концепція вирішення задачі дозволить агентам симуляції природньо еволюціонувати, що призведе до плавного досягнення бажаного результату або степеню навченості стратегії, та дозволить стратегіям набувати природньої поведінки для «виживання» в середовищі навчання. Цю концепцію було обрано через результат проведеного аналізу існуючих досліджень на цю тему. Так, першим застосування еволюційних алгоритмів для вирішення дилеми в'язня було значним проривом. Генетичні алгоритми, такі як ті, що розглядалися Cerruti та його колегами у 2020 році [10], використовують механізми природного добору для розробки оптимальних стратегій. У їхній роботі було показано, що такі алгоритми можуть створювати стратегії, здатні адаптуватися до змін середовища. Наприклад, шляхом мутації та кросовера створювалися нові стратегії, які демонстрували вищу стійкість до експлуатації порівняно з класичними підходами.

Навчаючи стратегії у великих популяціях, як це зроблено у роботах Montero-Porras, де досліджувалися memory-one стратегії, які враховують лише останній крок взаємодії [5], можна спростити обчислення, але це може призводити до втрати інформації про довгострокову поведінку опонентів.

Також заслуговує на увагу модель етикеткової кооперації (Tag-Based Cooperation), запропонована Rui Dong у 2020 році [4]. Вона базується на

еволюційній теорії та використовує додаткові "теги" для визначення схожості між гравцями. Цей підхід виявився ефективним у гетерогенних популяціях, але його недоліком є необхідність попередньої класифікації агентів, що зробило його недоцільним для запропонованого комплексу методів.

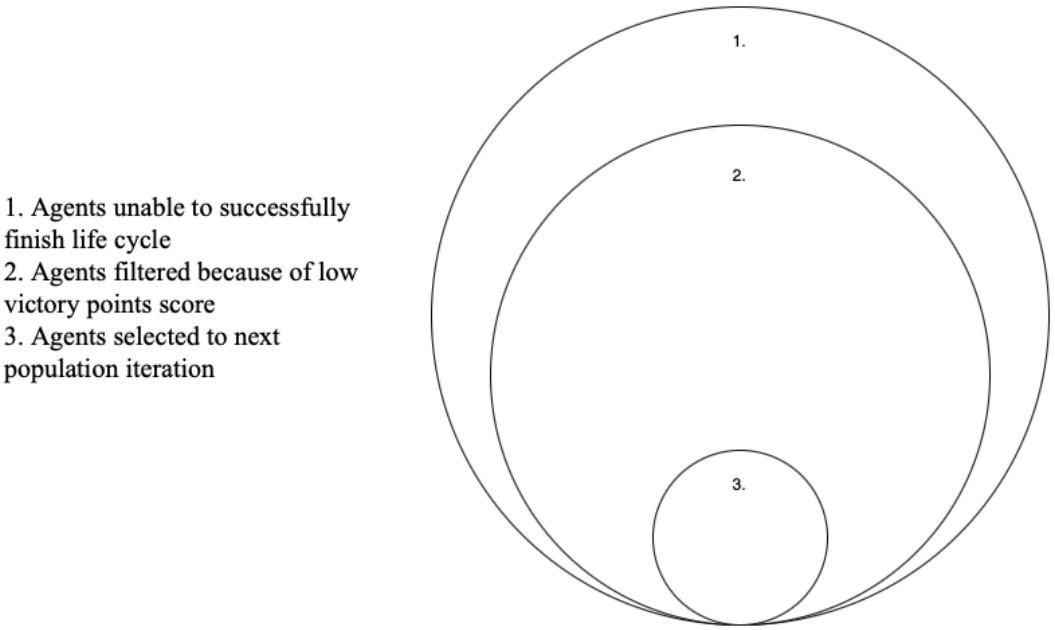


Рисунок 2.1.4 — Візуалізація як штучний відбір використовується для відбору найуспшініших динамічних стратегій

Концепція нейронних мереж у вирішенні дилеми в'язня:

Нейронні мережі дозволяють серйозно змінити підхід до генерації стратегій вирішення «дилеми в'язня». Проаналізовані досягнення у галузі штучного інтелекту дозволили застосовувати нейронні мережі для розробки цих стратегій, що робить такі стратегії гнучкими, відкрити до змін навіть після створення та динамічними відносно середовища виконання. Для створення даного методу, було використано проаналізовані в попередньому розділі дослідження з даної сфери, нейронні мережі використовуються для прогнозування майбутніх дій опонентів [7]. У досліджених роботах було показано, що такі мережі, поєднані з підкріплювальним навчанням, дозволяють створювати стратегії, які динамічно адаптуються до поведінки опонентів.

Крім того, використання convolutional neural networks (CNN) у просторі

дилеми в’язня дозволило аналізувати поведінку великих груп агентів. Наприклад, CNN застосовуються для класифікації стратегій і визначення їхньої ефективності в реальному часі. Основною перевагою цього підходу є його масштабованість, хоча потреба у великих обчислювальних ресурсах обмежує його використання.

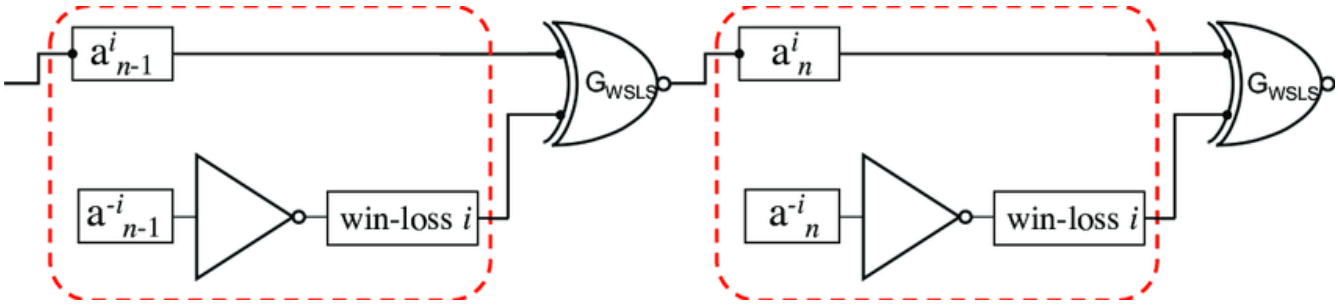


Рисунок 2.1.5 — Приклад простого персептрона як стратегії дилеми в’язня

Концепція використання генетичних алгоритмів та штучного відбору для вирішення дилеми в’язня:

Для генерації максимально ефективних стратегій, запропонований метод використовує генетичні алгоритми (ГА), які імітують принципи природного добору. Цей метод було обрано через те що він є одним із найпотужніших інструментів для розробки стратегій вирішення дилеми в’язня, що доведено проаналізованими дослідженнями. Ці алгоритми базуються на принципах еволюції: виживають і розмножуються лише найсильніші стратегії, що демонструють найкращі результати у середовищі гри.

У методі створення стратегій «дилеми в’язня» ГА використовуються для еволюційного пошуку оптимальних стратегій. Основними компонентами запропонованого генетичного алгоритму є:

— хромосоми: кожна стратегія кодується у вигляді хромосоми, яка може бути представлена як бінарний вектор або ряд чисел. Наприклад, поведінка гравця у відповіді на дії опонента може бути закодована у вигляді послідовності: "00 — кооперація, 01 — зрада" тощо;

— популяція: група стратегій, що взаємодіють у багатьох ігрових циклах. У процесі ітерацій деякі з них елімінаються, а інші покращуються через генетичні

операції;

— операції відбору: після кожної серії ігор відбувається відбір найуспішніших стратегій. У цьому процесі використовуються механізми фітнес-функцій, які оцінюють ефективність кожної стратегії, враховуючи її виграш або досягнутий рівень співпраці;

— кросовер: механізм обміну генетичної інформації між двома стратегіями. Наприклад, якщо одна стратегія демонструє високу кооперативність у перших кроках, а інша — у пізніх, їх об'єднання може створити більш збалансовану стратегію;

— мутація: випадкові зміни у генетичному коді стратегії для уникнення локального оптимуму. Наприклад, стратегія, яка завжди зраджує, може змінитися, дозволивши одну кооперацію у відповідь на кооперативний хід опонента.

За прикладом Umberto Cerruti та його колеги, що представили дослідження [10], яке поєднувало генетичні алгоритми з клітинними автоматами, було адаптовано їх рішення про використання генетичних алгоритів для еволюційного дослідження стратегій та їх поведінки. Саме використання їхній підходу у нашому методі, дозволив створити стратегії, здатні до адаптації в умовах змінного середовища. Зокрема, вони показали, що стратегії, навчені за допомогою ГА, демонструють вищу стійкість до агресивних опонентів, таких як Always Defect, порівняно з традиційними підходами.

Одним із найбільш успішних прикладів використання стало створення стратегій із динамічною адаптацією до типу опонента. Наприклад, Ferraz і Pitz розробили стратегію [3], яка змінює свою поведінку залежно від виграшів у попередніх іграх. Така стратегія успішно співпрацює з Tit-for-Tat, але переходить до зради проти Always Defect. Саме даний принцип сформував концепт додання різних метаданих про опонента на вхід нейромережі агента стратегії, що створило основу запропонованого нами методу.

Штучний відбір є доповненням до ГА і дозволяє моделювати специфічні сценарії шляхом зміни умов виживання стратегій. Його було використано, щоб у

тому числі штучно підвищити виживаність стратегій, які демонструють високий рівень кооперації, або, навпаки, віддати перевагу агресивним моделям для перевірки їхньої стійкості, у процесі генерації стратегій. Штучний відбір було використано після аналізу робіт Lin та ін., де було показано, що використання штучного відбору дозволяє створювати стратегії, які враховують довгострокову вигоду та мінімізують ризик помсти [7]. Наприклад, в цих дослідженнях, модель "адаптивного кооператора" навчається реагувати на поведінку інших, використовуючи накопичені дані з попередніх ігор.

Стратегії згенеровані програмними засобами за описаним методом мають наступні характеристики відносно уже існуючих методів:

— адаптивність: здатність створювати нові стратегії для невідомих середовищ;

— гнучкість, тобто можливість змінювати налаштування (рівень мутацій, розмір популяції тощо) залежно від специфіки задачі;

— висока ефективність у складних середовищах, де традиційні стратегії виявляються неефективними.

Але й присутні недоліки:

— висока обчислювальна складність, особливо для великих популяцій і довготривалих ігор;

— нестабільність результатів та залежність від початкових умов і параметрів алгоритму;

— схильність до локальних максимумів, що іноді вимагає значної кількості ітерацій для досягнення оптимального рішення.

2.2. Використані математичні моделі

Розробка динамічних стратегій вирішення дилеми в’язня передбачає використання нейронної мережі, яка моделює поведінку гравця, та генетичних алгоритмів для її навчання. Ці математичні моделі базуються на принципах

адаптації та оптимізації.

Математична модель динамічної стратегії:

Динамічна стратегія визначається як функція f, що використовує історію попередніх дій для прогнозування наступної дії:

$$H_t = \{(a_1, b_1), \ldots, (a_{\{t-1\}}, b_{\{t-1\}})\}$$

$$a_t = f(H_t; \theta),$$

де θ — набір параметрів нейронної мережі, яка реалізує f.

Функція пристосованості:

$$F(g_i) = \frac{1}{T}\sum_{t=1}^{T} u_t,$$

де u_t — виграш у раунді t, T — кількість раундів.

Селекція:

$$P(g_i) = \frac{F(g_i)}{\sum_{j=1}^{N} F(g_j)},$$

де N — розмір популяції, а ймовірність відбору $P(g_i)$ пропорційна пристосованості.

Кросовер:

$$g_{\text{new}} = \alpha \cdot g_i + (1 - \alpha) \cdot g_j,$$

де $\alpha \in [0,1]$, а нова особина g_{new} утворюється через комбінацію батьків.

Мутація:

$$g_i' = g_i + \mathcal{N}(0, \sigma),$$

де $\mathcal{N}(0, \sigma)$— нормальний розподіл, g_i — особина що підлягає мутації

Штучний відбір спрямований на стимулювання стратегій, що забезпечують стійку співпрацю або максимальний виграш.

Функція вибіркової адаптивності:

$$S(g_i) = \lambda_1 F(g_i) + \lambda_2 \text{Adapt}(g_i),$$

де $\text{Adapt}(g_i)$ оцінює адаптивність, λ_1 і λ_2 — вагові коефіцієнти.

2.3. Методика створення адаптивних стратегій. Розробка алгоритмів, заснованих на нейронних мережах і генетичних алгоритмах

Після аналізу існуючих методів розробки програмних застосунків для формування стратегій вирішення «дилеми в'язня» стало зрозуміло, що ці стратегії досягають максимальної ефективності лише у стабільних та малозмінних середовищах. Водночас, їхня математична простота ускладнює застосування у реальних сценаріях, що обмежує їх практичну цінність. Щоб вирішити цю проблему, ми запропонували метод, який підвищує гнучкість у генерації та навчанні таких стратегій. Стратегії, створені за цим методом, можуть адаптуватися до динамічних змін середовища та демонструвати поведінку, близьку до реальних умов.

Було досліджено що у створенні адаптивних стратегій вирішення дилеми в'язня одним із найефективніших підходів буде поєднання нейронних мереж та генетичних алгоритмів. Ця методика забезпечує створення моделей, які адаптуються до змін середовища та поведінки інших агентів, демонструючи значну перевагу над статичними стратегіями, які обмежені за своїм функціоналом і застосуванням.

Основою запропонованого методу є: агент, тобто імплементація динамічної

стратегії, що виступають основними моделями, здатними аналізувати нелінійні залежності між змінними, що є ключовим для складних ігрових сценаріїв.

Цей агент складається з кількох нейронних та LSTM моделей, що дозволяють йому аналізувати стан гри та приймати рішення. До вхідних даних такої мережі агента симуляції входять поточний стан гри, результати попередніх ітерацій та характеристики інших агентів.

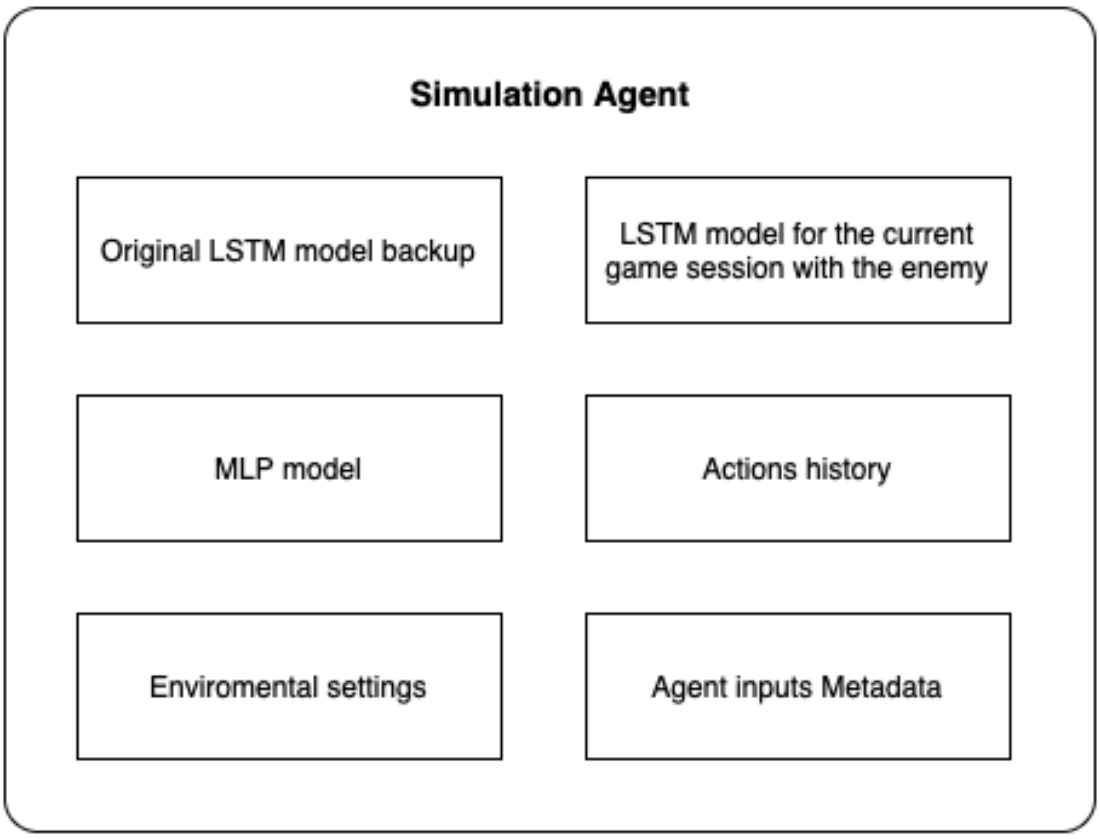


Рисунок 2.3.1 — Форма агента симуляції динамічної стратегії

Запропонований метод базується на використанні нейронних мереж для моделювання стратегій у дилемі в'язня. Нейронні мережі дозволяють створювати складні моделі поведінки, які ефективно коригують свої дії залежно від змін у середовищі. Цей підхід, названий методом динамічних стратегій, забезпечує агентів здатністю враховувати історію взаємодій і приймати рішення на основі поточних умов, сприяючи стабільній співпраці.

Основна ідея методу полягає у використанні нейронної мережі, яка приймає як вхідні дані інформацію про стратегію суперника та поточний стан гри. Ця

модель дозволяє агентам адаптувати свою поведінку у реальному часі завдяки алгоритму зворотного поширення помилки. Таким чином, агенти можуть реагувати на непередбачувані зміни в поведінці супротивника протягом тривалих ігрових сесій, що є ключовим у динамічних середовищах. На відміну від статичних стратегій, динамічні підходи з нейронними мережами враховують попередній досвід, роблять висновки та миттєво коригують свої дії, що робить їх значно ефективнішими у змінних умовах.

Нейронні мережі були обрані для цього підходу завдяки їх здатності моделювати складні нелінійні взаємозв'язки між вхідними даними та виходами. У контексті дилеми в'язня це дозволяє створювати стратегії, які враховують не лише поточні дії супротивників, але й їхні потенційні рішення в майбутньому. Наприклад, моделі LSTM та GRU забезпечують "пам'ять", яка дозволяє мережі враховувати минулі стани гри для прогнозування та адаптації до майбутніх сценаріїв. Це є одним із ключових елементів нашого методу.

Розробка такої нейронної мережі потребує врахування низки важливих аспектів. По-перше, модель має бути здатною опрацьовувати інформацію про взаємодії між агентами, що є визначальним фактором у прийнятті рішень. По-друге, архітектура мережі повинна враховувати як поточний стан гри, так і попередні дії, щоб забезпечити контекстуальну гнучкість. По-третє, нейронні мережі потребують ефективних алгоритмів навчання, таких як генетичні алгоритми, які забезпечують еволюційний відбір найкращих стратегій на основі їхньої ефективності у симуляціях.

Ядром запропонованого методу є схема роботи головної багатошарової нейронної мережі стратегії, що виглядає наступним чином (Рисунок 2.3.2):

Було додано нейрон IN1, який виконує функцію лічильника ходів, що дозволяє агенту орієнтуватися на різних стадіях гри — початковій, середній або кінцевій. Це критично важливо, адже поведінка гравців часто змінюється в залежності від етапу гри: на початку вони схильні до співпраці, але пізніше можуть переходити до агресивних дій. Таким чином, завдяки цьому нейрону агент може адаптувати свою стратегію до змін у тактиці супротивника.

Нейрони IN2 та IN3 для моніторингу балів агента та його суперника. Це допомагає оцінити, чи доцільно залишатися на кооперативній тактиці, чи слід перейти до конкурентної. Для більш точної оцінки стану гри можна використовувати різницю між балами, що подається через один нейрон, якщо нормалізація балів недоступна під час симуляції.

Для врахування історії дій супротивника та самого агента було додано нейрони IN4 та IN5. Вони дозволяють нейронній мережі навчатися на основі минулих взаємодій і коригувати свою стратегію. Це особливо важливо для створення адаптивних стратегій, які ефективно реагують на зміни поведінки супротивника. Результати досліджень свідчать, що стратегії, які враховують попередні дії опонента, суттєво покращують свої показники.

Нейрони IN6 та IN7, які аналізують схожість стратегій агентів. Це дозволяє агенту ідентифікувати партнерів, які, ймовірно, схильні до співпраці, та уникати конфліктів із гравцями, чиї стратегії є несумісними. Така функція підвищує як кооперативну, так і конкурентну ефективність агентів у багатокористувацьких середовищах.

Нейрон IN8 для обробки виходу моделі LSTM або GRU під час повторних взаємодій з одним і тим же супротивником. Цей нейрон виявляє поведінкові патерни опонента, що дозволяє агенту передбачати наступні дії супротивника та ефективно змінювати власну стратегію в реальному часі.

Ефективність стратегій згенерованих за нашим методом адаптуватися до змін у реальному часі. Це досягається шляхом використання механізму зворотного поширення помилки, який коригує ваги мережі на основі помилок, виявлених у результатах попередніх ітерацій. Онлайн-навчання в процесі гри дозволяє агентам змінювати свою поведінку в реальному часі, підвищуючи ефективність стратегій у динамічних середовищах, що було показано в проаналізованих досілженнях [7].

Одним із ключових аспектів цієї методики є використання симуляцій, які дозволяють оцінити ефективність стратегій у середовищах, що моделюють реальні взаємодії [9]. Такі симуляції включають велику кількість агентів з різними початковими стратегіями, які з часом змінюють свою поведінку на основі

взаємодій. Важливим елементом таких симуляцій є механізм штучного відбору, який імітує природний відбір у біологічних системах. Стратегії, які демонструють високу ефективність, мають більшу ймовірність бути збереженими в наступних поколіннях, тоді як менш ефективні стратегії замінюються.

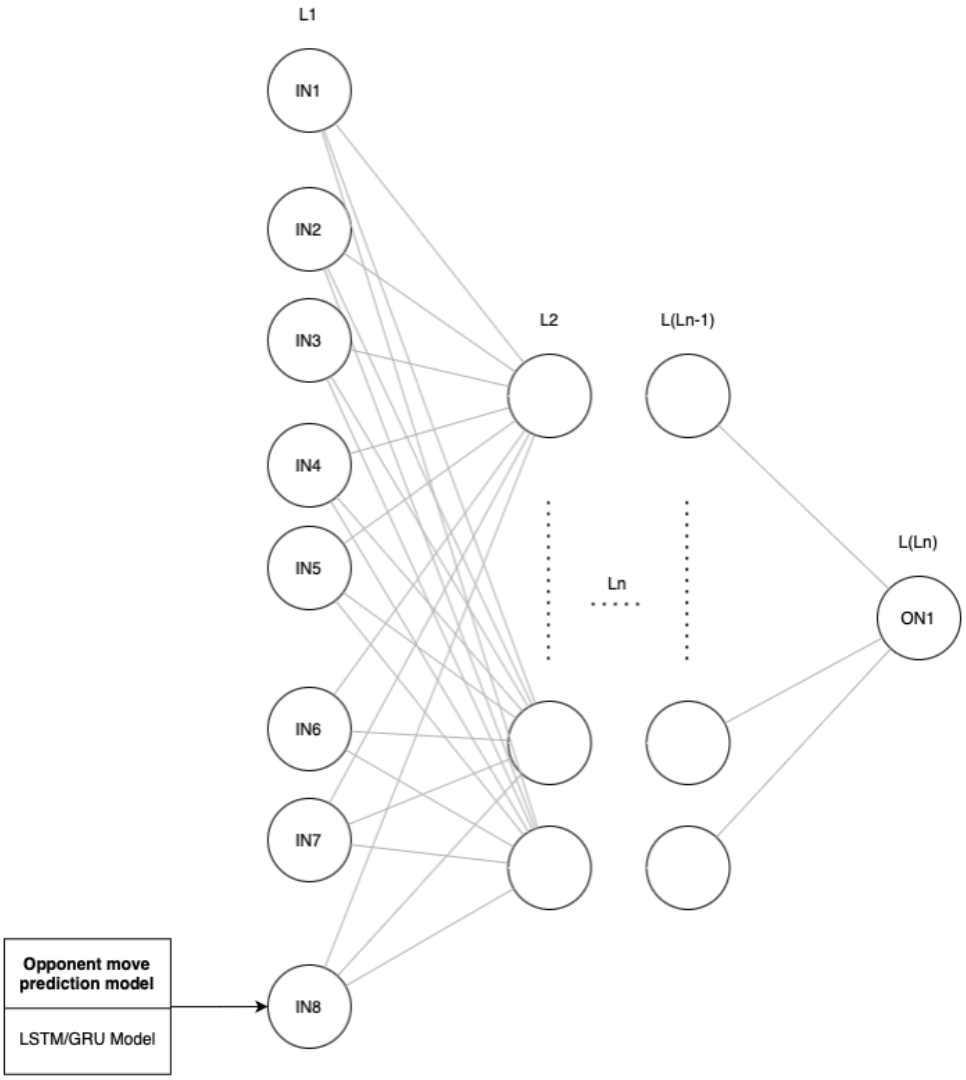


Рисунок 2.3.2 — Схема нейронної мережі динамічної стратегії

Надзвичайно важливою є здатність нейронних мереж виявляти приховані патерни у взаємодіях між агентами. Наприклад, стратегія може бути оптимізована для співпраці у певних сценаріях, одночасно запобігаючи надмірній експлуатації власних ресурсів іншими агентами, як це було показано в вивчених публікаціях подібних досліжень [4, 8]. Моделювання таких стратегій дозволяє краще зрозуміти, які фактори впливають на еволюцію співпраці та конкуренції у багатоагентних

системах.

Інтеграція генетичних алгоритмів з нейронними мережами також дає змогу створювати нові стратегії, які враховують поведінкові особливості супротивників. Наприклад, адаптація стратегій може базуватися на ідентифікації типу супротивника та коригуванні власної поведінки відповідно до отриманих даних. Це дозволяє агентам уникати конфліктів з більш агресивними супротивниками та сприяти співпраці з тими, хто має схожі стратегічні цілі [5].

Описаний підхід має широкий спектр застосувань у реальних системах, включаючи управління ресурсами, моделювання економічної поведінки та прогнозування соціальних взаємодій. Наприклад, стратегія, що базується на нейронних мережах і генетичних алгоритмах, може бути використана для створення моделей поведінки в умовах конкуренції за обмежені ресурси, що є важливим для управління великими соціально-економічними системами [1].

Таким чином, запропонований метод створення адаптивних стратегій на основі нейронних мереж і генетичних алгоритмів відкриває нові перспективи у вирішенні дилеми в’язня. Цей підхід забезпечує високу гнучкість і адаптивність моделей, що дозволяє ефективно вирішувати складні завдання в динамічних середовищах. Подальші дослідження в цьому напрямі можуть зосереджуватися на вдосконаленні архітектури нейронних мереж, оптимізації процесу навчання та дослідженні більш складних сценаріїв взаємодії.

2.4. Висновки

У результаті роботи було проведено дослідження та запропоновано методику створення динамічних стратегій для вирішення дилеми в’язня з використанням нейронних мереж та генетичних алгоритмів. Основною метою розробки стало створення гнучких і адаптивних моделей, здатних змінювати свою поведінку залежно від умов середовища та дій опонентів.

Було виділено кілька ключових концепцій, таких як еволюційні стратегії, нейронні мережі та методи штучного відбору. Концепція еволюційних стратегій

дозволяє агентам симуляції поступово адаптуватися до середовища, покращуючи свою ефективність через механізми мутації, кросоверу та відбору. Такі підходи, як використання memory-one стратегій та моделі етикеткової кооперації, продемонстрували свою ефективність у попередніх дослідженнях.

Ці методи створюють умови для природної адаптації агентів до складних сценаріїв, зокрема до гетерогенних популяцій із різними стратегіями. Однак вони також мають свої обмеження, такі як потреба в попередній класифікації агентів або спрощення обчислень за рахунок втрати інформації про довгострокову поведінку.

Застосування нейронних мереж для моделювання стратегій вирішення дилеми в'язня стало важливим проривом у дослідженнях. Використання глибоких нейронних мереж, зокрема LSTM та CNN, дозволяє створювати динамічні стратегії, які враховують як історію взаємодій, так і поточний стан гри. Це забезпечує моделям здатність адаптувати свою поведінку в реальному часі, що є ключовим для динамічних середовищ. Наприклад, використання CNN дозволяє аналізувати поведінку великих груп агентів і масштабувати моделі для багатокористувацьких сценаріїв.

Основна перевага таких підходів полягає у можливості адаптації до непередбачуваних змін у середовищі та в здатності прогнозувати поведінку супротивників. Проте, великий обсяг обчислень, необхідний для навчання таких моделей, залишається серйозним викликом.

Методи генетичних алгоритмів та штучного відбору стали основою для еволюційного пошуку оптимальних стратегій. Було розроблено механізми, що дозволяють моделювати адаптивні стратегії через взаємодію різних популяцій агентів. Генетичні алгоритми забезпечують еволюційний розвиток стратегій, використовуючи хромосоми для кодування поведінки агентів, механізми селекції для визначення найкращих стратегій, а також кросовер та мутацію для створення нових рішень. Штучний відбір доповнює ці алгоритми, дозволяючи моделювати специфічні сценарії, спрямовані на стимулювання кооперації або тестування стійкості агресивних моделей.

Запропоновані моделі демонструють такі переваги, як адаптивність,

гнучкість у налаштуванні параметрів та ефективність у динамічних середовищах. Водночас вони мають і певні недоліки: високу обчислювальну складність, нестабільність результатів через залежність від початкових умов та можливість локальної оптимізації, що потребує значної кількості ітерацій для досягнення найкращих результатів. Попри ці обмеження, використання нейронних мереж та генетичних алгоритмів відкриває нові можливості для розв'язання задач дилеми в'язня.

Порівняння розроблених методів зі статичними стратегіями, такими як «око за око» або «завжди зраджуй», показало значний потенціал динамічних моделей у змінних середовищах. Наприклад, динамічні стратегії з використанням LSTM забезпечують збереження інформації про минулі кроки супротивників, що дозволяє агентам адаптувати свою поведінку протягом тривалих ігрових сесій. Генетичні алгоритми дозволили оптимізувати структуру цих нейронних мереж та підвищити ефективність навчання.

Пропоновані моделі також враховують соціальні та економічні аспекти взаємодії агентів, що є особливо цінним для розв'язання реальних задач, таких як управління ресурсами, моделювання економічної поведінки або прогнозування соціальних взаємодій. Крім того розроблені за даним методом стратегії показують природнью поведінку, що дозволяє використовувати їх у симуляціях людської поведінки ти у соціологічних дослідженнях.

Даний метод також дозволяє регулювати навчаність результуючих стратегій, та рівень їх пристосованості до того чи іншого середовища.

Отже, запропонований метод розробки програмних застосунків для генерації та навчання адаптивних стратегій забезпечує високу гнучкість і адаптивність моделей, що дозволяє ефективно вирішувати складні завдання у динамічних середовищах. Подальші дослідження у цьому напрямі можуть бути спрямовані на вдосконалення архітектури нейронних мереж, оптимізацію процесу навчання та застосування моделей до більш складних сценаріїв взаємодії.

.

3. ТЕХНОЛОГІЯ РЕАЛІЗАЦІЇ МЕТОДУ СТВОРЕННЯ ПРОГРАМНИХ МОДУЛІВ ДЛЯ ГЕНЕРАЦІЇ ДИНАМІЧНИХ СТРАТЕГІЙ

3.1. Визначення функціональних і нефункціональних вимог

Формування функціональних і нефункціональних вимог для системи моделювання стратегій у дилемі в'язня є важливим етапом розробки. Ці вимоги визначають, які саме завдання система повинна виконувати, а також які характеристики повинні забезпечувати її ефективність, надійність та зручність у використанні. Враховуючи аналіз проведений у дослідженні [184], ключовими аспектами є забезпечення адаптивності стратегій, їх здатності до навчання, оптимізації та взаємодії в складних багатофакторних середовищах.

Основною функціональною вимогою є забезпечення точного моделювання стратегій у повторюваній дилемі в'язня. Система має підтримувати як статичні, так і динамічні стратегії. Динамічні стратегії повинні використовувати нейронні мережі для прийняття рішень та генетичні алгоритми для їх оптимізації. Це дозволяє моделювати поведінку агентів, що змінюється залежно від контексту гри, історії взаємодій і поведінки опонентів. Такий підхід забезпечує гнучкість, необхідну для аналізу поведінки в динамічних середовищах, як зазначено у [3, 184].

Система повинна забезпечувати підтримку багатьох агентів, які можуть взаємодіяти між собою в різних ігрових сценаріях. Це передбачає реалізацію симуляційного середовища, де агенти будуть змагатися чи співпрацювати, використовуючи різноманітні стратегії. Для цього необхідно передбачити можливість динамічного налаштування параметрів гри, таких як рівень винагороди за співпрацю чи зраду, ймовірність випадкових дій тощо. Це дозволить досліджувати вплив різних умов на еволюцію стратегій [4, 7].

Додатково, система повинна надавати функції автоматичного збору даних про результати симуляцій, аналізу їхньої ефективності та візуалізації результатів. Це забезпечить дослідникам можливість виявляти закономірності, аналізувати взаємодії агентів і виявляти найефективніші стратегії в конкретних сценаріях. Наприклад, у [5] підкреслюється важливість адаптації стратегій залежно від

контексту та надання дослідникам інструментів для аналізу цих процесів.

Нефункціональні вимоги включають забезпечення продуктивності та масштабованості системи. Симуляції можуть включати велику кількість агентів і різноманітних сценаріїв, що вимагає оптимізації алгоритмів та ефективного використання ресурсів обчислювальної техніки. Наприклад, нейронні мережі можуть бути ресурсомісткими, особливо якщо їх навчання відбувається в реальному часі. У [184] зазначається, що поєднання методів оптимізації, таких як генетичні алгоритми, може значно знизити обчислювальні витрати.

Зручність використання системи є ще однією важливою нефункціональною вимогою. Інтерфейс користувача повинен бути інтуїтивно зрозумілим, з можливістю налаштування параметрів симуляції та візуалізації результатів. Це дозволить дослідникам легко налаштовувати експерименти, змінювати умови гри та аналізувати дані без потреби глибоких знань у програмуванні.

Безпека системи є важливим фактором, особливо якщо її використовують для моделювання сценаріїв з чутливими даними чи для навчання моделей, що можуть бути застосовані у реальному світі. Це включає забезпечення захисту від несанкціонованого доступу та збереження конфіденційності даних симуляцій.

Додатково, система повинна підтримувати інтеграцію з існуючими інструментами для аналізу даних і машинного навчання. Це дозволить дослідникам використовувати знайомі їм платформи для додаткового аналізу чи підготовки даних, підвищуючи гнучкість та ефективність роботи. Наприклад, у [7, 9] розглядаються переваги інтеграції з бібліотеками машинного навчання для покращення адаптивності стратегій.

Таким чином, формування чітких функціональних і нефункціональних вимог є ключовим етапом у розробці системи для моделювання стратегій у дилемі в’язня. Це забезпечує її ефективність, адаптивність та зручність, дозволяючи проводити глибокі дослідження поведінкових моделей у різноманітних сценаріях.

Архітектура та компонентний дизайн

Розробка адаптивних стратегій для вирішення дилеми в’язня передбачає використання складної архітектури, яка об’єднує нейронні мережі для

прогнозування дій суперника та генетичні алгоритми для оптимізації параметрів навчання. Така інтеграція дозволяє створювати гнучкі та ефективні моделі, здатні до адаптації в умовах динамічних ігрових сценаріїв.

Архітектура системи ґрунтується на концепції багатоагентного середовища, де кожен агент репрезентує певну стратегію. Ключовими компонентами цієї архітектури є:

— симуляційне середовище, це базовий рівень, який забезпечує динамічну взаємодію між агентами. Воно включає інструменти для моделювання повторюваних ігор, контролю за параметрами середовища, такими як рівень винагороди, частота ітерацій, та змінюваність поведінки суперників;

— модуль нейронної мережі — основний елемент, що відповідає за прийняття рішень агентами. Нейронна мережа, побудована на основі LSTM (довга короткочасна пам'ять), дозволяє враховувати як попередні ходи суперників, так і змінні поточного ігрового стану. Кожен агент має власну унікальну нейронну мережу, яка відповідає за його стратегію;

— модуль генетичних алгоритмів — цей компонент здійснює оптимізацію структури нейронних мереж, їхніх вагових коефіцієнтів та функцій активації. Використання генетичних алгоритмів дозволяє ітеративно покращувати ефективність стратегій, моделюючи еволюцію у конкурентному середовищі;

— модуль навчання та адаптації — відповідає за оновлення параметрів нейронних мереж в режимі реального часу. Цей процес базується на алгоритмах підкріплювального навчання, які забезпечують здатність до самонавчання, враховуючи зміну поведінки інших агентів.

Архітектура нейронної мережі кожного агента включає кілька шарів, які виконують різні функції. Вхідний шар отримує дані про поточний стан гри, попередні дії обох гравців та їхні поточні бали. Приховані шари, зокрема рекурентні вузли LSTM, обробляють ці дані, виявляючи закономірності та формуючи прогнози щодо ймовірних майбутніх дій суперника. Вихідний шар генерує рішення агента — співпрацювати чи відмовитися.

Генетичні алгоритми доповнюють нейронні мережі шляхом оптимізації їхніх структурних параметрів. Наприклад, під час ініціалізації кожен агент отримує нейронну мережу з випадковими ваговими коефіцієнтами. Генетичний алгоритм, шляхом мутацій, кросоверів і відбору, поступово покращує ці параметри, зберігаючи найефективніші стратегії та адаптуючи їх до середовища. Цей процес моделює природний відбір, що дозволяє системі виробляти стратегії з високою адаптивністю.

Важливим компонентом архітектури є механізм обміну інформацією між агентами. У симуляційних експериментах агенти взаємодіють у парах, а результати їхніх ігор використовуються для оновлення параметрів мереж. Інформація про історію взаємодій зберігається в пам'яті LSTM, що дозволяє агентам приймати рішення на основі аналізу попередніх ітерацій.

Особлива увага приділяється проєктуванню процесу навчання. На початковому етапі кожен агент виконує базову стратегію, таку як "око за око". Під час симуляційної гри нейронна мережа агента поступово вдосконалюється, навчаючись на помилках та успішних діях. Цей процес навчання включає зворотне поширення помилки та оновлення вагових коефіцієнтів мережі.

Генетичні алгоритми застосовуються для адаптації до змінних середовищ. Наприклад, якщо в процесі гри середовище змінюється, агенти з менш ефективними стратегіями поступово "вибувають", а їхнє місце займають агенти, які продемонстрували вищу ефективність. Завдяки цьому симуляція дозволяє виявити найстабільніші та найбільш адаптивні стратегії.

Архітектура також передбачає використання багатошарових нейронних мереж з додатковими механізмами, такими як механізми уваги (attention mechanisms), що покращують здатність агентів прогнозувати дії суперника. Це дозволяє зосереджувати увагу на найбільш важливих аспектах ігрового середовища та приймати оптимальні рішення.

Таким чином, запропонована архітектура і компонентний дизайн забезпечують створення високоефективних адаптивних стратегій для вирішення дилеми в'язня. Інтеграція нейронних мереж та генетичних алгоритмів дозволяє

враховувати як індивідуальну поведінку агентів, так і вплив динамічних змін у середовищі, створюючи умови для ефективної кооперації навіть у конкурентних умовах.

3.2. Реалізація модулів генерації та навчання

Навчання стратегій використовуючи описаний метод для їх подальшого використання для вирішення дилеми ув’язненого, може зайняти багато ресурсів, бо навчання методом генетичних потребує багато ресурсів для роботи. Щоб уникнути зайвих витрат, та прискорити навчання, нами сформовано поетапний алгоритм для підготовки та навчання стратегій. Цей процес включає в себе такі етапи (Рисунок 3.2.1.):

— реалізація ефективних статичних стратегій, де впроваджуються стратегії, які вже довели свою ефективність у попередніх дослідженнях, такі як "око за око" або "безумовна співпраця". Цей крок є вирішальним, оскільки закладає основу для порівняння з динамічними стратегіями. Впровадження встановлених стратегій забезпечує базу, що дозволяє чіткіше оцінити ефективність динамічних стратегій, які будуть розроблені пізніше;

— розробка симуляційного середовища, коли буде створено симуляційне середовище для Дилеми ув'язненого з змінними параметрами, що дозволяє гнучко налаштовувати умови гри та тестувати різні стратегії. Ця гнучкість є необхідною для дослідження того, як різні стратегії працюють у різних сценаріях. Маніпулюючи екологічними змінними, ми можемо краще зрозуміти динаміку взаємодій агентів та умови, які сприяють або співпраці, або конкуренції;

— програмна реалізація динамічних стратегій, що передбачає інтеграцію моделей нейронних мереж з динамічним оновленням стратегій у реальному часі. Впровадження динамічних стратегій є необхідним для того, щоб агенти могли навчатися та адаптуватися під час гри. Ця здатність є життєво важливою в умовах, де супротивники можуть змінювати свої стратегії, оскільки вона дозволяє агентам ефективно реагувати на нові виклики та підвищувати свої шанси на успіх;

— генерація агентів, де на основі випадкових початкових значень ваг нейронних мереж генеруються динамічні стратегії для змагання зі статичними агентами. Цей процес генерації є вирішальним, оскільки він вводить різноманітність у поведінці агентів, що призводить до більш надійних симуляцій. Створюючи різноманітних агентів з різними початковими стратегіями, ми можемо спостерігати, як ці агенти взаємодіють і еволюціонують з часом, що надає цінні відомості про ефективність різних стратегічних підходів;

— наближення динамічних стратегій до роботи статичних, тобто використання глибокого навчання дозволяє динамічним стратегіям тісно узгоджуватися з поведінкою статичних стратегій, що полегшує початкову фазу навчання та оптимізації. Ця узгодженість є важливою, оскільки вона дозволяє динамічним стратегіям використовувати переваги перевірених статичних стратегій, зберігаючи при цьому гнучкість для адаптації до змін. Цей гібридний підхід дозволить покращити ефективність проходження наступних етапів навчання;

— симуляція ігор та навчання методом штучного відбору, під час симуляцій використовується метод клітинних автоматів для організації взаємодій агентів, моделюючи складні середовища з численними учасниками. Цей метод є важливим для оцінки того, як агенти з подібними або різними стратегіями взаємодіють у різних ситуаціях і як їхні стратегії еволюціонують з часом. Процес моделювання дозволяє досліджувати нові поведінкові патерни, які виникають внаслідок взаємодії агентів, що сприяє глибшому розумінню співпраці та конкуренції в багатагентних системах. Змінюючи покоління, методом штучнго відбору ми можем вивести цілі сімейства динамічних стратегій, що за допомогою відповідних нейронів зможуть відрізняти своїх від чужих, навіть без аналізу поведінки.

— збір та аналіз результатів, тобто останній етап, що передбачає аналіз отриманих результатів симуляцій для визначення найбільш ефективних динамічних стратегій для різних ігрових умов. Цей аналіз є критично важливим для

валідації запропонованих методів та розуміння, які стратегії дають найкращі результати в конкретних ситуаціях. Оцінюючи ефективність різних стратегій, ми можемо вдосконалити наші моделі та визначити напрямки майбутніх досліджень.

Рисунок 3.2.1 — Діаграма алгоритму процесу навчання динамічних стратегій

Слідуючи даній інструкції по формуванню програмного забезпечення для симуляції та формування динамічних стратегій, можна імплементувати засіб для погвної генерації гнучких та розумних стратегій.

3.3. Висновки

У даній частині роботи було проведено детальне визначення ключових вимог до розроблюваної системи, сформовано архітектуру та компонентний дизайн, а

також розроблено структуру динамічних стратегій. Результати аналізу функціональних і нефункціональних вимог забезпечили базис для створення ефективного рішення, яке відповідає критеріям адаптивності, продуктивності та надійності. Основна увага була приділена необхідності інтеграції нейронних мереж та генетичних алгоритмів для формування стратегій, що здатні динамічно реагувати на зміни ігрового середовища.

Розроблена архітектура охоплює багаторівневу модель, яка включає симуляційне середовище, модулі навчання та адаптації, а також механізми для оптимізації на основі генетичних алгоритмів. Використання рекурентних нейронних мереж із довготривалою короткочасною пам'яттю (LSTM) дозволило забезпечити врахування історії ітерацій у прийнятті рішень, тоді як генетичні алгоритми адаптують параметри стратегій до поточних умов середовища, сприяючи еволюції ефективних рішень.

Структура динамічних стратегій розроблена з урахуванням ключових аспектів адаптивності. Вона базується на поєднанні механізмів прогнозування поведінки суперника, оптимізації дій і стратегічної взаємодії між агентами. Запропонована структура враховує потребу в інтерактивній адаптації до змін середовища, використовуючи зворотне навчання та відбір на основі результатів симуляцій.

Таким чином, отримані результати демонструють практичну придатність запропонованого підходу до вирішення задач динамічного моделювання стратегій у дилемі в'язня. Визначені вимоги, запропонована архітектура та розроблена структура стратегій створюють міцну основу для подальшої реалізації та тестування системи, орієнтованої на максимальну адаптивність і ефективність у динамічному середовищі.

4. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСОБУ

4.1. Програмна реалізація

Програмна реалізація адаптивних стратегій для вирішення дилеми в'язня передбачає інтеграцію сучасних інструментів і технологій, які дозволяють забезпечити як ефективність обчислень, так і зручність розробки. Обраними компонентами реалізації є Node.js, TypeScript та TensorFlow, які забезпечують гнучкість у розробці ігрових симуляцій та алгоритмів навчання на базі нейронних мереж і генетичних алгоритмів.

Node.js було обрано як основну серверну платформу для реалізації симуляційного середовища завдяки її ефективності у роботі з асинхронними операціями. Node.js забезпечує можливість масштабованої взаємодії між агентами у багатокористувацькому середовищі. Це середовище підтримує обробку великої кількості одночасних ігрових сценаріїв, що є критично важливим для тестування стратегій у конкурентних умовах.

TypeScript, як мова програмування, яка розширює можливості JavaScript, забезпечує строгий контроль типів, що дозволяє мінімізувати кількість помилок при розробці складних моделей. Завдяки цьому, розробники можуть створювати чітко структурований код для опису поведінки агентів, їхніх стратегій та алгоритмів взаємодії. TypeScript також полегшує інтеграцію різних компонентів системи, таких як модулі для симуляції ігор, нейронних мереж і генетичних алгоритмів.

TensorFlow є основною бібліотекою для створення, навчання та використання моделей нейронних мереж. TensorFlow надає потужні інструменти для реалізації складних архітектур, таких як LSTM, які дозволяють агентам зберігати пам'ять про попередні дії та коригувати свої стратегії відповідно до змін у поведінці опонентів. Використання TensorFlow дозволяє ефективно працювати з великими наборами даних, оптимізуючи процес навчання агентів шляхом паралельних обчислень на графічних процесорах (GPU).

Середовище розробки, яке обирається для такої системи, зазвичай включає

Visual Studio Code завдяки його зручному інтерфейсу, підтримці розширень для TypeScript, Node.js і TensorFlow, а також можливості інтеграції з системами контролю версій, такими як Git. Це дозволяє розробникам зручно писати, налагоджувати та тестувати код, а також спільно працювати над проєктом.

Процес реалізації починається з визначення структури нейронної мережі для агентів. Використовуючи TensorFlow, створюються моделі, які враховують минулі стани гри, поточні дії агентів та можливі реакції суперників. Для цього використовуються вхідні дані, що репрезентують стан гри, результати попередніх раундів та інші релевантні фактори. Наприклад, для навчання агентів може використовуватися набір історичних даних про ігрові взаємодії, що дозволяє нейронним мережам ідентифікувати поведінкові патерни.

Генетичні алгоритми, реалізовані також у TensorFlow, забезпечують оптимізацію параметрів нейронних мереж. Вони використовуються для еволюції стратегій шляхом симуляції природного відбору. На кожній ітерації алгоритм відбирає найефективніші стратегії, засновані на показниках успішності, таких як коефіцієнт співпраці або середній виграш. Після цього створюються нові покоління стратегій, які комбінують характеристики попередніх, що дозволяє агентам адаптуватися до змін у середовищі гри.

4.2. Реалізовані модулі

Розробка системи для вирішення дилеми в’язня на основі нейронних мереж та генетичних алгоритмів вимагала створення чітко структурованої програмної архітектури. Система побудована на платформі Node.js із використанням TypeScript як основної мови програмування. Для реалізації нейронних мереж застосовано бібліотеку TensorFlow.js, яка забезпечує гнучкість і продуктивність обчислень.

Проєкт структуровано за принципами модульності та відповідальності. Всі файли розміщені в директорії src, яка має наступні підкаталоги:

— types, містить опис типів і інтерфейсів;

— helpers, набір допоміжних функцій;

— constants, глобальні константи програми;

— models, реалізація нейронних мереж для моделювання стратегій;

— strategies, реалізація динамічних і статичних стратегій;

— simulation, функціонал для моделювання ігор.

Файли налаштовані для компіляції через TypeScript Compiler, що забезпечує типобезпеку і спрощує підтримку коду. Конфігураційні файли виглядають наступним чином:

```
// package.json
{
  "scripts": {
    "start": "tsc && node ./dist/index.js"
  },
  "dependencies": {
    "@tensorflow/tfjs-node": "^4.21.0",
    "typescript": "^5.6.3"
  }
}
// tsconfig.json
{
  "compilerOptions": {
    "rootDir": "src",
    "outDir": "dist"
  },
  "include": [
    "src/"
  ]
}
```

Модуль types відповідає за визначення типів та інтерфейсів, що забезпечують узгодженість даних між різними частинами програми.

```
// src/types/move.enum.ts
export enum EMove {
    POSITIVE = 1,
    NEGATIVE = 0
}

// src/types/game-move.interface.ts
```

```
import { EMove } from "./move.enum";

export interface IGameRoundResult {
    m1: EMove;
    m2: EMove;
}
export type TGameRoundsHistory = Array<IGameRoundResult>;
```

Модуль helpers включає функції для генерації випадкових чисел, перевірки значень на null та інші допоміжні дії.

```
// src/helpers/rand.helper.ts
export function randomIntFromInterval(min, max) {
    return Math.floor(Math.random() * (max - min + 1) +
min);
}

// src/helpers/is-null.helper.ts
export const isNull = <T>(val: T | null | undefined): val
is null | undefined  => val === null || val === undefined;
```

Модуль constants містить основні константи, що використовуються в програмі, такі як кількість раундів.

```
// src/constants.ts
export const MAX_ROUNDS_PER_GAME = 100;
```

Файл index.ts виступає точкою входу для виконання програми, ініціалізуючи основні компоненти системи.

Ця структура забезпечує легкість розширення та інтеграції нових функціональних можливостей у майбутньому. Наступні підрозділи розглянуть окремі компоненти, такі як моделі та стратегії, детальніше.

Розробка модулів для динамічних стратегій у системі вирішення дилеми в'язня базується на Node.js з використанням TypeScript і TensorFlow. Динамічні стратегії створюються за допомогою нейронних мереж і включають механізми адаптивної поведінки через використання моделей глибокого навчання (MLP та LSTM). Цей підхід дозволяє системі адаптуватися до змінних умов гри, прогнозувати поведінку суперника та генерувати оптимальні рішення залежно від

історії взаємодій.

Всі стратегії наслідують один клас Strategy. В ньому описані головні методи які повинна реалізовувати стратегія:

```
export abstract class Strategy {
    abstract makeMove(params: IMoveParams): Promise<EMove>
    abstract predict(history: TGameRoundsHistory):
Promise<EMove>
    abstract reset(): void
}
```

Метод makeMove отримує слідуючу дію Стратегії на основі її вхідних параметрів.

Метод predict отримує вхідний параметр – передбачення слідуючої дії супротивника, на основі історії взаємодії з ним.

Метод reset очищує пам'ять Стратегії перед новим суперником.

Основний модуль динамічних стратегій представлено класом DynamicStrategy, який поєднує дві моделі: багатошаровий перцептрон (MLP) для аналізу поточного стану гри та LSTM для обробки послідовностей історичних даних. Логіка модулів організована таким чином, щоб забезпечити максимально ефективне використання ресурсів під час тренування моделей і прогнозування.

```
export class DynamicStrategy extends Strategy {
    public readonly mlp: MLPModel;
    public readonly lstm: LSTMModel;

    constructor() {
        super();
        this.mlp = new MLPModel()
        this.lstm = new
LSTMModel(DefaultOpponentMovesPredictor.model)
    }
    reset(): void {}

    async makeMove(params: IMoveParams): Promise<EMove> {
        const prediction = await this.mlp.predict(params)
        const move = prediction >= 0.5 ? EMove.POSITIVE :
```

```
EMove.NEGATIVE
        return move
    }

    async predict(history: TGameRoundsHistory):
Promise<number> {
        if(!history.length) return 0.5
        const output = await this.lstm.predict(history)
        return output
    }

    async trainPredictor(history: TGameRoundsHistory) {
        if(history.length > 2) {
            const trainingData = []
            for(let i = 1; i < history.length; i++) {
                trainingData.push(
                    history.slice(0, i)
                )
            }
            console.log(trainingData)
            await this.lstm.trainOn(trainingData, 1)
        }
    }

    async train(trainData: [IMoveParams, EMove][]) {
        return this.mlp.train(trainData, 10)
    }
}
```

Властивість mlp це об'єкт моделі багатошарового перцептрону (MLPModel). Ця модель використовується для прогнозування наступного ходу на основі статичних параметрів, переданих у вигляді IMoveParams.

Властивсть lstm це об'єкт моделі LSTM (LSTMModel). Використовується для аналізу історії ходів гри (TGameRoundsHistory) і прогнозування дій опонента.

Метод makeMove є ключовим компонентом стратегії. Він приймає параметри гри params, обробляє їх за допомогою MLP та повертає хід. Якщо прогнозоване значення перевищує 0.5, стратегія обирає позитивний хід (EMove.POSITIVE), інакше – негативний (EMove.NEGATIVE). Таким чином, цей метод відповідає за прийняття рішення на основі поточного стану гри.

Метод predict використовується для аналізу послідовності попередніх

раундів. Якщо історія порожня, метод повертає нейтральне значення 0.5, що свідчить про невизначеність. Якщо історія доступна, LSTM використовується для генерації прогнозу про майбутній хід суперника на основі цієї історії. Це забезпечує врахування динаміки попередніх ходів під час прогнозування.

Метод trainPredictor відповідає за навчання LSTM на основі історії ігрових раундів. Якщо історія містить більше двох раундів, вона розбивається на послідовності, кожна з яких представляє підмножину попередніх раундів. Ці підмножини використовуються як дані для тренування LSTM, дозволяючи моделі покращувати прогнозування на основі послідовностей дій.

Метод train використовується для тренування MLP. Він приймає набір даних, що складається з пар параметрів гри та відповідних ходів, та виконує навчання моделі протягом 10 епох. Це дозволяє адаптувати MLP до специфіки гри, покращуючи точність її рішень.

Далі було реалізовано Модель багатошарового перцептрону (MLP). MLP відповідає за оцінку поточного стану гри на основі параметрів, що включають історію попередніх дій, рахунок, різницю в балах між гравцями тощо. Архітектура цієї моделі представлена трьома прихованими шарами з 16, 16 та 8 нейронами відповідно. Функції активації — ReLU для прихованих шарів та sigmoid для вихідного шару. Модель оптимізована за допомогою алгоритму Адама (adam):

```
export class MLPModel {
    public model = tf.sequential();

    constructor(model?: tf.Sequential) {
        if(model) {
            this.model = model
            return;
        }

        this.model.add(tf.layers.dense({
            inputShape: [7],
            units: 16,
            activation: 'relu'
        }));
        this.model.add(tf.layers.dense({
```

```
            units: 16,
            activation: 'relu'
        }));
        this.model.add(tf.layers.dense({
            units: 8,
            activation: 'relu'
        }));
        this.model.add(tf.layers.dense({
            units: 1,
            activation: 'sigmoid'
        }));
        this.model.compile({
            optimizer: tf.train.adam(),
            loss: 'binaryCrossentropy',
            metrics: ['accuracy']
        });
    }

    public prepTensors(trainData: [IMoveParams, EMove][]) {
        const inputSequences: TModelInput[] = [];
        const labels: number[] = [];

        trainData.forEach(([ params, move ]) => {

inputSequences.push(transformIMoveParamsToModelInput(params
));
            labels.push(move);
        });

        const inputTensor = tf.tensor2d(inputSequences);
        const labelTensor = tf.tensor1d(labels, 'int32');
        return { inputTensor, labelTensor }
    }

    public async train(trainData: [IMoveParams, EMove][],
epochs = 15) {
        const { inputTensor, labelTensor } =
this.prepTensors(trainData)
        const history = await this.model.fit(
            inputTensor,
            labelTensor,
            {
                epochs,
                batchSize: 32,
                validationSplit: 0.2,
```

```
                callbacks: [
                    tf.callbacks.earlyStopping({
                        monitor: "val_acc",
                        patience: 10,
                        minDelta: 0.01
                    }),
                ]
            }
        );
        return history
    }

    async predict(params: IMoveParams): Promise<EMove> {
        const inputTensor =
tf.tensor2d([transformIMoveParamsToModelInput(params)]);
        const predictions = this.model.predict(inputTensor)
as tf.Tensor;

        const predictedValues = predictions.dataSync();
        return Array.from(predictedValues)[0]
    }
}
```

На початку визначається тип TModelInput, який представляє вхідний вектор моделі як масив із семи чисел.

```
type TModelInput = [
    number,
    number,
    number,
    number,
    number,
    number,
    number,
]
```

Функція normalize виконує нормалізацію значень у заданий діапазон [0, 1], що важливо для стабільної роботи нейронної мережі. Далі функція transformIMoveParamsToModelInput перетворює об’єкт ігрових параметрів IMoveParams у формат TModelInput, нормалізуючи числові значення, використовуючи максимальні та мінімальні значення, а також замінюючи відсутні дані стандартним значенням 0.5.

```
const normalize = (val, max, min) => (val - min) / (max -
min);

const transformIMoveParamsToModelInput = (i: IMoveParams):
TModelInput => {
    return [
        normalize(i.roundNumber, i.maxRounds, 0),
        normalize(i.scoreDifference, i.maxScore, 0),
        i.last ? i.last.selfMove : 0.5,
        i.last ? i.last.opponentMove : 0.5,
        isNull(i.absDiff) ? 0.5 : normalize(i.absDiff,
i.maxAbsDiff, 0),
        isNull(i.avgDiff) ? 0.5 : normalize(i.avgDiff, 1,
0),
        isNull(i.opponentMovePredictorOutput) ? 0.5 :
i.opponentMovePredictorOutput,
    ]
}
```

Клас MLPModel реалізує модель MLP за допомогою бібліотеки TensorFlow.js. У конструкторі створюється нейронна мережа, що має три прихованих шари з активаційною функцією ReLU та вихідний шар із функцією sigmoid. Модель компілюється з використанням алгоритму оптимізації Adam, функції втрат binaryCrossentropyта метрики accuracy.

Метод prepTensors готує навчальні дані для моделі. Він перетворює вхідні параметри гри та відповідні метки класів у формат тензорів. Тензори використовуються як вхід для навчання моделі.

Метод train навчає модель на основі підготовлених даних. Він викликає функцію fit, що навчає модель за кілька епох, з використанням мінібатчів розміром 32. У процесі навчання використовується механізм ранньої зупинки, який припиняє навчання, якщо точність на валідаційному наборі перестає покращуватися після 10 епох.

Метод predict виконує прогнозування на основі вхідних параметрів гри. Вхідні дані конвертуються в тензор, який передається моделі для отримання прогнозів. Результат, який повертає модель, інтерпретується як імовірність певного результату, і використовується для прийняття рішення.

Наступною реалізовано Модель LSTM. LSTM використовується для аналізу послідовностей ходів, виконаних обома гравцями. Це дозволяє враховувати довгострокові взаємодії та прогнозувати дії суперника. Модель включає шар маскування для обробки змінної довжини послідовностей, шар LSTM з 50 нейронами та вихідний шар із функцією активації sigmoid.

```
export class LSTMModel {
    public model = tf.sequential();

    constructor(model?: tf.Sequential) {
        if(model) {
            this.model = model
        } else {
            this.model.add(tf.layers.masking({ maskValue: -
1, inputShape: [MAX_ROUNDS_PER_GAME, 2] }));
            this.model.add(tf.layers.lstm({ units: 50,
returnSequences: false }));
            this.model.add(tf.layers.dense({ units: 1,
activation: 'sigmoid' }));
        }

        this.model.compile({
            optimizer: 'adam',
            loss: 'binaryCrossentropy',
            metrics: ['accuracy']
        });
    }

    static padSequences(sequences: number[][][], maxLength:
number): number[][][] {
        return sequences.map(seq => {
            if((maxLength - seq.length) <= 0) {
                console.log(seq)
                console.log(maxLength)
            }
            const padding = Array(maxLength -
seq.length).fill([-1, -1]);
            return seq.concat(padding);
        });
    }

    public prepTensors(trainData: TGameRoundsHistory[]) {
```

```
        const inputSequences: number[][][] = [];
        const labels: number[] = [];

        trainData.forEach(gameHistory => {
            if(!gameHistory.length) return;
            const sequence = gameHistory.map(move =>
[move.m1, move.m2]);
            inputSequences.push(sequence);
            labels.push(gameHistory[gameHistory.length -
1].m2);  // Predict the m2
        });

        const paddedInputSequences =
LSTMModel.padSequences(inputSequences,
MAX_ROUNDS_PER_GAME);

        // Convert data to tensors
        const inputTensor =
tf.tensor3d(paddedInputSequences, [inputSequences.length,
MAX_ROUNDS_PER_GAME, 2]);
        const labelTensor = tf.tensor1d(labels, 'int32');
        return { inputTensor, labelTensor }
    }

    public async trainOn(games: TGameRoundsHistory[],
epochs = 30) {
        const { inputTensor, labelTensor } =
this.prepTensors(games);
        const history = await this.model.fit(inputTensor,
labelTensor, {
            epochs,
            batchSize: 32,
            validationSplit: 0.3,
            callbacks: [
                tf.callbacks.earlyStopping({
                    monitor: "val_acc",
                    patience: 5,
                    minDelta: 0.01
                }),
            ]
        });
        return history
    }

    async predict(history: TGameRoundsHistory):
```

```
Promise<number> {
        const { inputTensor } = this.prepTensors([history])
        const predictions = this.model.predict(inputTensor)
as tf.Tensor;

        const predictedValues = predictions.dataSync();
        return Array.from(predictedValues)[0]
    }
}
```

У конструкторі класу створюється послідовна модель TensorFlow.js (tf.Sequential). Якщо під час створення класу передається готова модель, вона використовується замість створення нової. Якщо ж модель не передається, створюється нова структура нейронної мережі. Першим шаром додається Masking, який ігнорує значення -1 у послідовності, що дозволяє обробляти послідовності змінної довжини. Далі йде шар LSTM з 50 нейронами, який обробляє послідовність і виводить результат як одне значення. Останній шар – це щільний (Dense) шар із функцією активації sigmoid, який видає ймовірність у діапазоні від 0 до 1. Модель компілюється з оптимізатором adam, функцією втрат binaryCrossentropy та метрикою точності.

Метод padSequences забезпечує приведення послідовностей ходів до однакової довжини шляхом додавання заповнювачів (-1), якщо початкова довжина послідовності менша за максимальну дозволену кількість раундів гри. Це необхідно для коректної роботи LSTM, яка очікує на вхід фіксовану довжину послідовності.

Метод prepTensors готує дані для навчання моделі. Він приймає історію ігрових раундів (де кожен раунд містить дії двох гравців), формує послідовності з дій обох гравців у вигляді масиву, а також створює мітки для навчання, які відповідають останньому ходу другого гравця. Потім ці дані конвертуються у тензори для передачі в модель.

Метод trainOn відповідає за навчання моделі на основі переданих історій ігрових раундів. У процесі навчання дані розділяються на навчальну та валідаційну вибірки, використовується зупинка навчання при досягненні стабільності

валідаційної точності. Параметри, такі як кількість епох та розмір батчу, можуть змінюватися.

Метод predict призначений для прогнозування на основі історії гри. Історія передається у вигляді послідовності, яка підготовлюється до вигляду, зручного для моделі, після чого модель здійснює передбачення наступного ходу супротивника. Результат передбачення – ймовірність, яку можна інтерпретувати як схильність супротивника до конкретної дії.

Ця модель має бути передгенерована як початкова модель для всіх стратегій. Це відбувається на початку когжної ітерації генерування стратегій. Ось так виглядає приклад збереженої та згенерованної LSTM моделі.

Ключовим в реалізації буде код роботи середовища для симуляції. Ось приклад того як реалізовано гру в дилему на мові TypeScript:

Клас PrisonersDilemmaGame реалізує модель гри "Дилема в'язня". Він забезпечує симуляцію раундів, аналіз результатів та взаємодію між двома стратегіями. Початкова частина класу визначає основні змінні та матрицю результатів для різних комбінацій ходів:

```
export class PrisonersDilemmaGame {
    public roundsHistory: TGameRoundsHistory;
    public params: TGameMoveParamsHistory;

    private readonly SCORE_MATRIX = {
        [EMove.POSITIVE]: {
            [EMove.POSITIVE]: 2,
            [EMove.NEGATIVE]: 0,
        },
        [EMove.NEGATIVE]: {
            [EMove.POSITIVE]: 3,
            [EMove.NEGATIVE]: 1,
        }
    }

    private s1Score = 0;
    private s2Score = 0;

    public get lastRound() {
        return this.roundsHistory[this.roundsHistory.length
```

```
- 1];
    }
```

Властивість SCORE_MATRIX відображає матрицю з математичної поделі дилеми, що показує скільки очків отримує кожен агент після того чи іншого результату ходу.

Властивість roundsHistory зберігає масив ходів, щоб на основі істрої можна було довчатись моделям передбачування.

Властивість params зберігає набір налаштувань гри та середовища.

Конструктор та метод reset дозволяють ініціалізувати та скинути стан гри. Вони очищають історію раундів, параметри і рахунок обох стратегій:

```
constructor() {
    this.reset();
}

reset() {
    this.roundsHistory = [];
    this.params = [];
    this.s1Score = 0;
    this.s2Score = 0;
}
```

Метод simulate виконує симуляцію гри для двох стратегій. Він приймає на вхід стратегії, кількість раундів, а також параметри для автоматичного скидання стану гри та збереження історії параметрів:

```
async simulate(
    strategy1: Strategy,
    strategy2: Strategy,
    rounds: number,
    autoReset = true,
    saveParamsHistory = false,
): Promise<TGameRoundsHistory> {
    const MAX_SCORE_PER_ROUND = 3;
    let cumulativeDiff: number;
    let diffCount: number;
    let avgDiff: number;
```

Якщо обидві стратегії є динамічними, розраховується середня різниця ваг

їхніх моделей. Це дозволяє оцінити, наскільки вони відрізняються за своїми внутрішніми станами. Після цього збираються всі інші вхідні параметри для стратегій:

```
        if(
            strategy1 instanceof DynamicStrategy
            && strategy2 instanceof DynamicStrategy
        ) {
            const weights1 =
strategy1.mlp.model.getWeights();
            const weights2 =
strategy2.mlp.model.getWeights();

            cumulativeDiff = 0;
            diffCount = 0;

            for(let weightsIndex = 0; weightsIndex <
weights1.length; weightsIndex++) {
                const w1 =
weights1[weightsIndex].arraySync();
                const w2 =
weights2[weightsIndex].arraySync();

                const calcDiff = (a, b) => {
                    if(a.length !== b.length) throw new
Error("Cannot calculate diff");
                    for(let i = 0; i < a.length; i++) {
                        if(typeof a[i] === 'number' &&
typeof b[i] === 'number') {
                            cumulativeDiff += Math.abs(a[i]
- b[i]);
                            diffCount++;
                        } else {
                            calcDiff(a[i], b[i]);
                        }
                    }
                };
                calcDiff(w1, w2);
            }

            avgDiff = cumulativeDiff / diffCount;

            console.log("cumulativeDiff: ",
cumulativeDiff);
```

```
            console.log("c: ", diffCount);
            console.log("avgDiff: ", avgDiff);
        }
```

Основний цикл гри виконує симуляцію кожного раунду. Стратегії отримують дані про поточний стан гри та обчислюють свої ходи:

```
        for (let i = 0; i < rounds; i++) {
            const lastRound: IGameRoundResult | null =
this.lastRound;

            const s1Prediction = await
strategy1.predict(this.roundsHistory);
            const s2Prediction = await
strategy2.predict(this.roundsHistory);

            const sharedParams = {
                history: this.roundsHistory,
                roundNumber: i,
                maxRounds: rounds - 1,
                avgDiff,
                absDiff: cumulativeDiff,
                maxAbsDiff: diffCount,
                maxScore: rounds * MAX_SCORE_PER_ROUND,
            };
            const s1Params: IMoveParams = {
                ...sharedParams,
                last: lastRound
                    ? {
                        opponentMove: lastRound.m2,
                        selfMove: lastRound.m1
                    }
                    : null,
                scoreDifference: this.s1Score -
this.s2Score,
                opponentMovePredictorOutput: s1Prediction
            };
            const s2Params: IMoveParams = {
                ...sharedParams,
                last: lastRound
                    ? {
                        opponentMove: lastRound.m1,
                        selfMove: lastRound.m2
                    }
```

```
                    : null,
                scoreDifference: this.s2Score -
this.s1Score,
                opponentMovePredictorOutput: s2Prediction
            };
            const move1 = await
strategy1.makeMove(s1Params);
            const move2 = await
strategy2.makeMove(s2Params);

            this.s1Score +=
this.SCORE_MATRIX[move1][move2];
            this.s2Score +=
this.SCORE_MATRIX[move2][move1];

            if(saveParamsHistory)
                this.params.push([s1Params, s2Params]);

            this.roundsHistory.push({
                m1: move1,
                m2: move2
            });
        }
```

Після завершення гри скидається стан стратегій та, за потреби, гри. Повертається повна історія раундів:

```
        strategy1.reset();
        strategy2.reset();

        const history = this.roundsHistory;
        if(autoReset)
            this.reset();
        return history;
    }
}
```

Також для навчання динамічних стратегій, за методикою, було реалізовано статичні стратегії у вигляді алгоритмів. Ось кілька прикладів:

```
export class TitForTatWithForgivenessStaticStrategy extends
StaticStrategy {
    async makeMove(params: IMoveParams): Promise<EMove> {
        if (isNull(params.last)) {
```

```
            return EMove.POSITIVE;  // Cooperate on the
first move
        }
        if (params.last.opponentMove === EMove.NEGATIVE &&
Math.random() < 0.25) {
            return EMove.POSITIVE;  // 25% chance to
forgive defection
        }
        return params.last.opponentMove;  // Mimic the
opponent's last move
    }
    async predict(history): Promise<EMove> {
        return null;
    }
    reset(): void {}
}

export class RandomStaticStrategy extends StaticStrategy {
    async makeMove(): Promise<EMove> {
        return Math.random() > 0.5 ? EMove.POSITIVE :
EMove.NEGATIVE;  // Randomly cooperate or defect
    }
    async predict(history): Promise<EMove> {
        return null;
    }
    reset(): void {}
}
export class GrimTriggerStaticStrategy extends
StaticStrategy {
    private defected: boolean = false;

    async makeMove(params: IMoveParams): Promise<EMove> {
        if (this.defected) {
            return EMove.NEGATIVE;  // Once defected,
always defect
        }
        if (params.last?.opponentMove === EMove.NEGATIVE) {
            this.defected = true;
            return EMove.NEGATIVE;
        }
        return EMove.POSITIVE;  // Cooperate unless
opponent defects
    }

    async predict(history): Promise<EMove> {
```

```
        return null;
    }
    reset(): void {
        this.defected = false;
    }
}
```

Навчання MLP виконується на основі даних про попередні ігрові раунди. Для LSTM послідовності історичних даних перетворюються на тензори фіксованої довжини. Моделі тренуються за допомогою функції втрат binaryCrossentropy із використанням зупинки за раннім припиненням для уникнення перенавчання.

Особливості цієї реалізації:

— програма має чітку ієрархію модулів, що включають моделі, стратегії, типи та допоміжні функції. Це спрощує тестування та масштабування системи;

— завдяки інтеграції нейронних мереж стратегії можуть швидко адаптуватися до змін у поведінці супротивника;

— використання бібліотеки TensorFlow.js забезпечує високу продуктивність обчислень.

Дані модулі забезпечують динамічне навчання стратегій, дозволяючи моделям враховувати як поточний стан гри, так і історичні дані. Це створює основу для подальших досліджень та вдосконалення адаптивних стратегій для складних багатокористувацьких сценаріїв.

4.3. Тестування та експериментальні результати

Розроблені модулі для створення, навчання та тестування динамічних стратегій були протестовані в умовах симуляційного середовища для дилеми ув’язненого. Основною метою експериментів було оцінити ефективність розроблених стратегій у порівнянні з класичними статичними підходами, а також перевірити здатність моделей до адаптації в умовах змінного середовища.

Усі модулі, реалізовані на NodeJS із використанням TypeScript та бібліотеки TensorFlow, були об’єднані в єдину систему, яка дозволяє моделювати взаємодії

між агентами. Для цього використовувались динамічні стратегії, побудовані на основі рекурентних нейронних мереж LSTM і багатошарових перцептронів (MLP), що проходили навчання за допомогою генетичних алгоритмів.

Тестування проходило у кілька етапів.

На першому етапі проводилося попереднє навчання динамічних стратегій з використанням історичних даних про ігри між статичними стратегіями. Навчання здійснювалось з використанням моделі LSTMModel, що була адаптована для врахування послідовності ходів і змін у поведінці опонентів.

Клас DefaultOpponentMovesPredictor реалізує механізм прогнозування наступних ходів опонента в грі "Дилема в'язня". Цей клас побудований навколо використання LSTM-моделі для аналізу історії ігрових ходів та передбачення стратегій опонентів. Код класу організований так, щоб забезпечити як навчання моделі, так і її тестування. Ось покроковий розбір функціоналу класу.

```
private static readonly VERSION = 11
private static readonly NAME = `lstm-default-opponent-move-
predictor-${this.VERSION}/model.json`
public static model: tf.Sequential = null;
```

VERSION визначає версію моделі, а NAME задає ім'я файлу, де зберігається модель. model є статичною змінною, яка використовується для зберігання завантаженої або створеної нейронної мережі.

```
public static async load() {
    this.model = await
tf.loadLayersModel(`file://./${this.NAME}`) as
tf.Sequential
}
```

Метод load завантажує попередньо збережену модель із файлу. Він використовує TensorFlow.js для десеріалізації моделі, яка зберігається у форматі JSON. Цей метод корисний для відновлення стану моделі без повторного навчання.

```
public static async trainOn(strategies: Strategy[]) {
    const lstm = new LSTMModel()
    const game = new PrisonersDilemmaGame()
```

Ця частина створює екземпляри LSTMModel (для прогнозування) і PrisonersDilemmaGame (для симуляції гри між стратегій). Вони є основними інструментами для навчання та тестування моделі.

```
const trainStrategiesPairs = randomPairs(strategies, 5000)
const trainData: TGameRoundsHistory[] = []
for(let [s1, s2] of trainStrategiesPairs) {
    const history = await game.simulate(
        s1,
        s2,
        randomIntFromInterval(1, MAX_ROUNDS_PER_GAME)
    )
    trainData.push(history)
}
```

Тут генерується навчальна вибірка. Метод randomPairs створює 5000 випадкових пар стратегій. Для кожної пари симулюється гра з випадковою кількістю раундів, і історія ходів зберігається у масиві trainData.

```
await lstm.trainOn(trainData)
```

Цей виклик передає зібрані дані в LSTM-модель для навчання. Навчання базується на обробці послідовностей, що є історією ігрових ходів.

```
const testStrategiesPairs = randomPairs(strategies, 100)
const testData: TGameRoundsHistory[] = []
for(let [s1, s2] of testStrategiesPairs) {
    const history = await game.simulate(
        s1,
        s2,
        randomIntFromInterval(1, MAX_ROUNDS_PER_GAME)
    )
    testData.push(history)
}
```

Після навчання створюється тестова вибірка з 100 пар стратегій. Симулюються ігри, і історія ходів зберігається у testData.

```
const { inputTensor, labelTensor } =
lstm.prepTensors(testData)
const labels = labelTensor.arraySync()
const predictions = lstm.model.predict(inputTensor) as
```

```
tf.Tensor;
```

Ця частина підготовлює дані тестової вибірки у вигляді тензорів, які можна передати в модель для прогнозування. labelTensor містить реальні мітки, а predictions — прогнозовані значення.

```
const predictedValues = predictions.dataSync();
const predictedMoves = Array.from(predictedValues).map(val
=> (val > 0.5 ? EMove.POSITIVE : EMove.NEGATIVE));
```

Тут прогнозовані ймовірності перетворюються на конкретні дії. Якщо значення більше за 0.5, модель передбачає позитивний хід, інакше — негативний.

```
let correctPredictions = 0;
labels.forEach((label, index) => {
    if (label === predictedMoves[index]) {
        correctPredictions++;
    }
});
const accuracy = correctPredictions / labels.length;
console.log(`Test Accuracy: ${accuracy * 100}%`);
```

Порівнюються передбачення моделі з реальними мітками, щоб обчислити точність моделі. Результат виводиться в консоль.

```
await lstm.model.save(`file://./lstm-${(new
Date()).toISOString().split('.').shift().replace(' ', '-
')}`)
```

Після успішного тестування модель зберігається у файл. Назва файлу включає поточну дату й час, щоб уникнути конфліктів із попередніми версіями.

Код навчання виглядає наступним чином:

```
const lstmModel = new LSTMModel();
const trainData = prepareTrainData(staticStrategies, 5000);
// Підготовка даних для навчання
await lstmModel.trainOn(trainData, 30); // Навчання
протягом 30 епох
```

Динамічні стратегії формувалися на основі випадкових ініціалізацій ваг нейронних мереж, а їх еволюція контролювалася генетичними алгоритмами.

Процес передбачав відбір найкращих стратегій за результатами ігор:

```
const strategies = generateDynamicStrategies(10); //
Генерація 10 динамічних стратегій
const evolutionResults = evolveStrategies(strategies,
generations); // Еволюція стратегій
```

Створені стратегії тестувалися у середовищі, де вони взаємодіяли між собою та зі статичними стратегіями, такими як Tit-for-Tat та Grim Trigger. Всі симуляції проводились у модулі PrisonersDilemmaGame:

```
const game = new PrisonersDilemmaGame();
const history = await game.simulate(dynamicStrategy1,
staticStrategy, 100);
```

Один із прикладів взаємодії між динамічною стратегією та Tit-for-Tat наведено нижче:

```
const dynamicStrategy = new DynamicStrategy();
const staticStrategy = new TitForTatStaticStrategy();

const game = new PrisonersDilemmaGame();
const history = await game.simulate(dynamicStrategy,
staticStrategy, 100);

console.log("Game history:", history);
```

Для проведення експериментів із використанням штучного відбору і генетичних алгоритмів у вирішенні дилеми в'язня реалізовано комплексний підхід. Основна мета — оптимізація стратегій у динамічних середовищах за допомогою адаптивних алгоритмів, побудованих на нейронних мережах та методах еволюційного навчання.

Штучний відбір імітує процес природної еволюції, адаптуючи стратегії через генетичні оператори: мутацію, кросинговер і селекцію. Кожен агент представляє окрему стратегію, закодовану у вигляді ваг нейронної мережі. Під час симуляції ігор між агентами обчислюється їхня результативність. Найкращі агенти (ті, що отримали максимальні значення функції пристосованості) передають свої параметри наступному поколінню.

Приклад фрагменту коду, що демонструє базову реалізацію селекції агентів:

```
import { randomPairs } from "../helpers/random-pairs-from-
array.helper";

function selectFittestAgents(agents: Agent[], scores:
number[]): Agent[] {
    const sortedAgents = agents.map((agent, idx) => ({
agent, score: scores[idx] }))
                                .sort((a, b) => b.score -
a.score);
    return sortedAgents.slice(0, agents.length /
2).map(entry => entry.agent);
}
```

Процес еволюції включає наступні етапи:

— ініціалізація популяції, створення початкового набору агентів із випадковими вагами нейронних мереж;

— симуляція ігор, де кожна стратегія бере участь у багатьох іграх проти інших стратегій;

— оцінка продуктивності, де результати кожної гри використовуються для обчислення функції пристосованості;

— відбір, де найкращі стратегії зберігаються, а решта видаляється;

— кросинговер і мутація, де створюються нові стратегії через комбінацію ваг батьків та невеликі випадкові зміни.

```
function mutateWeights(weights: tf.Tensor[]): tf.Tensor[] {
    return weights.map(w => {
        const array = w.arraySync() as number[][];
        const mutatedArray = array.map(row => row.map(value
=> value + Math.random() * 0.1 - 0.05));
        return tf.tensor(mutatedArray, w.shape);
    });
}
```

У процесі експериментів тестувалися такі аспекти:

— вплив розміру популяції;

— кількість поколінь;

— динаміка навчання: адаптація стратегій до змінних умов і нових типів супротивників.

```
import { PrisonersDilemmaGame } from "../simulation/game";
import { DynamicStrategy } from
"../strategies/dynamic/dynamic.stractegy";

async function simulatePopulation(population:
DynamicStrategy[], rounds: number): number[] {
    const scores = Array(population.length).fill(0);
    for (let i = 0; i < population.length; i++) {
        for (let j = i + 1; j < population.length; j++) {
            const game = new PrisonersDilemmaGame();
            const history = await
game.simulate(population[i], population[j], rounds);
            scores[i] += game.s1Score;
            scores[j] += game.s2Score;
        }
    }
    return scores;
}
```

4.4. Висновки

У цьому розділі було розглянуто сучасні підходи до створення адаптивних стратегій вирішення дилеми в’язня з використанням нейронних мереж і генетичних алгоритмів. Основна увага приділялася динамічним стратегіям, які здатні змінювати свою поведінку залежно від умов гри та дій інших гравців. Інтеграція методів машинного навчання, таких як рекурентні нейронні мережі LSTM, у поєднанні з генетичними алгоритмами створює потужний інструмент для моделювання взаємодій у багатокористувацьких середовищах.

Використання Node.js та TypeScript у якості основи розробки забезпечує модульність і масштабованість програмного забезпечення, тоді як TensorFlow виступає ключовим інструментом для роботи з нейронними мережами. Запропонована структура програми дозволяє ефективно взаємодіяти між модулями, зберігаючи високу продуктивність та гнучкість у зміні параметрів

моделювання.

Генетичні алгоритми стали важливим елементом навчання агентів. Їх застосування дозволяє оптимізувати параметри стратегій, імітуючи природні процеси еволюції. Це включає такі етапи, як селекція, мутація та кросинговер, які спрямовані на створення нових стратегій і покращення адаптації до змінних умов гри. Нейронні мережі, у свою чергу, використовуються для прогнозування дій супротивників на основі історичних даних і формування рішень відповідно до обраної стратегії.

Розроблене програмне забезпечення складається з кількох основних модулів, кожен з яких виконує специфічну функцію: симуляція гри, навчання нейронної мережі, оптимізація стратегій за допомогою генетичних алгоритмів та аналіз результатів взаємодії. Інтеграція цих компонентів забезпечує цілісність системи та дозволяє моделювати складні сценарії ігрової взаємодії.

Робота також включає концептуальну базу, яка охоплює різноманітні типи стратегій, такі як класичні, динамічні, стратегії на основі підкріплювального навчання, а також спеціалізовані підходи, включаючи Hopeful, Suspicious і Sink State стратегії. Усі ці елементи об’єднані в єдину структуру, що дозволяє досліджувати різні аспекти теорії ігор.

Запропонований метод слугує основою для подальшого розвитку систем адаптивного навчання. У майбутньому можливе впровадження додаткових методів навчання та розширення функціоналу, спрямованого на аналіз складніших ігрових моделей та симуляцію багатокористувацьких сценаріїв.

ВИСНОВКИ

У процесі виконання дипломної роботи було проведено комплексне дослідження стратегій вирішення дилеми в'язня за допомогою нейронних мереж та генетичних алгоритмів. Проаналізовано існуючі підходи до моделювання поведінки в іграх із теорії ігор, виконано розробку та тестування нових динамічних стратегій на основі сучасних підходів до адаптивного навчання.

У першому розділі роботи було проведено аналіз класичних стратегій дилеми в'язня, таких як «Завжди зраджуй», «Завжди співпрацюй», «Око за око», а також складніших підходів, зокрема, стратегій із пам'яттю та адаптацією. На основі цього аналізу визначено основні недоліки статичних моделей, зокрема їх обмежену здатність адаптуватися до змінної поведінки супротивника. Це створило основу для вибору напрямків удосконалення, які спрямовані на розробку динамічних стратегій із використанням нейронних мереж та методів штучного інтелекту.

Другий розділ містить теоретичне обґрунтування математичних моделей та алгоритмів, що використовуються для створення адаптивних стратегій. Описано математичний апарат, який базується на використанні рекурентних нейронних мереж LSTM для моделювання поведінки гравців, а також на методах генетичних алгоритмів і штучного відбору для оптимізації стратегій. Зокрема, представлено детальну математичну модель навчання нейронних мереж, що дозволяє враховувати історію взаємодій між гравцями, та модель генетичного алгоритму, який забезпечує еволюцію стратегій через адаптацію параметрів нейронних мереж.

У третьому розділі було розроблено програмну реалізацію запропонованих підходів, що включає архітектуру нейронних мереж для реалізації динамічних стратегій, алгоритми їх навчання за допомогою генетичних методів та програмний інструментарій для тестування. Для реалізації було обрано сучасні фреймворки, такі як TensorFlow, які забезпечують ефективну обробку великих масивів даних і можливість налаштування складних архітектур нейронних мереж. Крім того, описано алгоритми тестування, які дозволяють оцінити ефективність стратегій у різних ігрових сценаріях.

Четвертий розділ містить опис емпіричних досліджень, які обґрунтовують, що динамічні стратегії, які використовують моделі штучного інтелекту та генетичні алгоритми, мають потенціал перевершити класичні підходи як у плані адаптації до поведінки опонента, так і в досягненні стабільного співробітництва.

Таким чином, результати дослідження демонструють, що запропонований метод має значний потенціал для вирішення дилеми в'язня шляхом адаптації стратегій у реальному часі. Попри потребу в значних обчислювальних ресурсах, ці підходи наближають моделювання до реальних сценаріїв, де поведінка агентів залежить від контексту і взаємодій.

Результати роботи мають практичну цінність, оскільки можуть бути застосовані для аналізу та моделювання складних соціальних і економічних систем, де поведінка агентів змінюється динамічно. Крім того, запропоновані моделі можуть використовуватися в системах штучного інтелекту для розробки стратегій взаємодії в багатоагентних середовищах. Подальші дослідження можуть бути спрямовані на інтеграцію цих підходів у більш складні сценарії, такі як багатоетапні ігри або задачі з великим числом гравців.

Таким чином, результати дипломної роботи підтверджують доцільність використання сучасних методів нейронних мереж та еволюційних алгоритмів для вирішення задач теорії ігор, демонструючи їх ефективність, адаптивність та широкий спектр можливих застосувань.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Nikoleta E. Glynatsi, Vincent Knight and Marc Harper. Properties of Winning Iterated Prisoner's Dilemma Strategies, 2024.
2. Domajnko, Martin. Iterated prisoner's dilemma and survival of the fittest from an ecological perspective. Proceedings of the 8th Student Computing Research Symposium (SCORES'22) 2022.
3. Ferraz, V., Pitz, T. Analyzing the Impact of Strategic Behavior in an Evolutionary Learning Model Using a Genetic Algorithm. Comput Econ 63, 437–475 (2024). https://doi.org/10.1007/s10614-022-10348-1
4. Rui Dong, Xinghong Jia, Xianjia Wang, Yonggang Chen. Optimal Tag-Based Cooperation Control for the "Prisoner's Dilemma", 2020. https://doi.org/10.1155/2020/8498613
5. Montero-Porras, E., Grujić, J., Fernández Domingos, E. *et al.* Inferring strategies from observations in long iterated Prisoner's dilemma experiments. *Sci Rep* **12**, 7589 (2022). https://doi.org/10.1038/s41598-022-11654-2
6. Lanyu Yang, Dongchun Jiang, Fuqiang Guo And Mingjian Fu. The State-Action-Reward-State-Action Algorithm in Spatial Prisoner's Dilemma Game. College of Computer and Data Science, Fuzhou University, Fuzhou, 350116, China.
7. Lin, B., Bouneffouf, D., Cecchi, G. (2022). Online Learning in Iterated Prisoner's Dilemma to Mimic Human Behavior. In: Khanna, S., Cao, J., Bai, Q., Xu, G. (eds) PRICAI 2022: Trends in Artificial Intelligence. PRICAI 2022. Lecture Notes in Computer Science, vol 13631. Springer, Cham. https://doi.org/10.1007/978-3-031-20868-3_10
8. Gansterer, M., Hartl, R.F. The Prisoners' Dilemma in collaborative carriers' request selection. Cent Eur J Oper Res 29, 73–87 (2021). https://doi.org/10.1007/s10100-020-00717-2
9. de Paulo, K.P., Estombelo-Montesco, C.A. & Tejada, J. New memory-one strategies of the Iterated Prisoner's Dilemma: a new framework to programmed human-AI interaction. Discov Psychol 4, 20 (2024). https://doi.org/10.1007/s44202-024-00133-6
10. Umberto Cerruti, Simone Dutto, Nadir Murru, A symbiosis between cellular automata and genetic algorithms, Chaos, Solitons & Fractals, Volume 134, 2020, 109719, ISSN 0960-0779, https://doi.org/10.1016/j.chaos.2020.109719.
11. Бородкіна І.Л. Web-технології та Web-дизайн : застосування мови HTML для створення електронних ресурсів/ І.Л.Бородкіна, Р. О. Бородкін: К: Ліра До, 2020. – 212 с.
12. Серверні web-технології. Навчальний посібник. К.: КПІ ім. Ігоря Сікорського, 2023. – 109 с.
13. Технології розробки WEB-ресурсів [Електронний ресурс] : навчальний посібник / В. П. Молчанов, О. К. Пандорін. – Харків : ХНЕУ ім. С. Кузнеця, 2020. – 130 с. ISBN 978-966-676-754-0.
14. Ерік Фрімен. Head First. Патерни проєктування. ТОВ «Фабула», 2020. –

672c.
15. Micheal Full (2020). How the Internet Works & the Web Development Process. p. 30. ISBN 979-8641657073.
16. Martin Fowler. Refactoring: Improving the Design of Existing Code (2nd Edition). Addison-Wesley Professional, 2019. – 424p. ISBN 978-0134757599
17. Daniel. Learn React Hooks: Build and refactor modern React.js applications using Hooks. Packt Publishing, 2019. — 426 p. — ISBN 978-1838641443, 1838641440.
18. Alls J. Clean Code in C#: Refactor your legacy C# code base and improve application performance by applying best practices. Packt, 2020. — 500 p. — ISBN 9781838982973.
19. J. Clean Code in C#: Refactor your legacy C# code base and improve application performance by applying best practices. Packt, 2020. — 500 p. — ISBN 9781838982973.
20. Lemaire M. Refactoring at Scale: Regaining Control of Your Codebase. O'Reilly Media, 2020 —245 p. — ISBN 978-1-492-07553-0.
21. McDonough J.E. Automated Unit Testing with ABAP: A Practical Approach. Apress, 2021. — 408 p. — ISBN 978-1484269503.
22. Oliver, J.M.; Esteban, M.D.; López-Gutiérrez, J.-S.; Negro, V.; Neves, M.G. Optimizing Wave Overtopping Energy Converters by ANN Modelling: Evaluating the Overtopping Rate Forecasting as the First Step. Sustainability 2021, 13, 1483.
23. Li, T.; Chan, Y.H.; Lun, D.P.K. Improved Multiple-Image-Based Reflection Removal Algorithm Using Deep Neural Networks. IEEE Trans. Image Process. 2021, 30, 68–79.
24. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative adversarial networks. Commun. ACM 2020, 63, 139–144.
25. Rani, P.; Kavita; Verma, S.; Nguyen, G.N. Mitigation of Black Hole and Gray Hole Attack Using Swarm Inspired Algorithm with Artificial Neural Network. IEEE Access 2020, 8, 121755–121764.
26. Milad, A.; Adwan, I.; Majeed, S.A.; Yusoff, N.I.M.; Al-Ansari, N.; Yaseen, Z.M. Emerging Technologies of Deep Learning Models Development for Pavement Temperature Prediction. IEEE Access 2021, 9, 23840–23849.
27. Shamshirband, S.; Mosavi, A.; Rabczuk, T.; Nabipour, N.; Chau, K. Prediction of significant wave height; comparison between nested grid numerical model, and machine learning models of artificial neural networks, extreme learning and support vector machines. Eng. Appl. Comput. Fluid Mech. 2020, 14, 805–817.
28. Cecchetti, R.; de Paulis, F.; Olivieri, C.; Orlandi, A.; Buecker, M. Effective PCB Decoupling Optimization by Combining an Iterative Genetic Algorithm and Machine Learning. Electronics 2020, 9, 1243.
29. Nabipour, N.; Dehghani, M.; Mosavi, A.; Shamshirband, S. Short-Term Hydrological Drought Forecasting Based on Different Nature-Inspired Optimization Algorithms Hybridized with Artificial Neural Networks. IEEE

Access 2020, 8, 15210–15222.
30. Nabipour, N.; Dehghani, M.; Mosavi, A.; Shamshirband, S. Short-Term Hydrological Drought Forecasting Based on Different Nature-Inspired Optimization Algorithms Hybridized with Artificial Neural Networks. IEEE Access 2020, 8, 15210–15222.

ДОДАТОК А

(обов'язковий)

ПРОГРАМНИЙ КОД ОСНОВНИХ МОДУЛІВ

А.1 Абстрактний клас стратегії

```
export abstract class Strategy {
    abstract makeMove(params: IMoveParams): Promise<EMove>
    abstract predict(history: TGameRoundsHistory):
Promise<EMove>
    abstract reset(): void
}
```

А.2 Модуль динамічних стратегій

```
export class DynamicStrategy extends Strategy {
    public readonly mlp: MLPModel;
    public readonly lstm: LSTMModel;

    constructor() {
        super();
        this.mlp = new MLPModel()
        this.lstm = new
LSTMModel(DefaultOpponentMovesPredictor.model)
    }
    reset(): void {}

    async makeMove(params: IMoveParams): Promise<EMove> {
        const prediction = await this.mlp.predict(params)
        const move = prediction >= 0.5 ? EMove.POSITIVE :
EMove.NEGATIVE
        return move
    }

    async predict(history: TGameRoundsHistory):
Promise<number> {
        if(!history.length) return 0.5
        const output = await this.lstm.predict(history)
        return output
    }

    async trainPredictor(history: TGameRoundsHistory) {
        if(history.length > 2) {
```

```
            const trainingData = []
            for(let i = 1; i < history.length; i++) {
                trainingData.push(
                    history.slice(0, i)
                )
            }
            console.log(trainingData)
            await this.lstm.trainOn(trainingData, 1)
        }
    }

    async train(trainData: [IMoveParams, EMove][]) {
        return this.mlp.train(trainData, 10)
    }
}
```

A.3 Модуль MLP моделі

```
export class MLPModel {
    public model = tf.sequential();

    constructor(model?: tf.Sequential) {
        if(model) {
            this.model = model
            return;
        }

        this.model.add(tf.layers.dense({
            inputShape: [7],
            units: 16,
            activation: 'relu'
        }));
        this.model.add(tf.layers.dense({
            units: 16,
            activation: 'relu'
        }));
        this.model.add(tf.layers.dense({
            units: 8,
            activation: 'relu'
        }));
        this.model.add(tf.layers.dense({
            units: 1,
            activation: 'sigmoid'
        }));
        this.model.compile({
```

```
            optimizer: tf.train.adam(),
            loss: 'binaryCrossentropy',
            metrics: ['accuracy']
        });
    }

    public prepTensors(trainData: [IMoveParams, EMove][]) {
        const inputSequences: TModelInput[] = [];
        const labels: number[] = [];

        trainData.forEach(([ params, move ]) => {

inputSequences.push(transformIMoveParamsToModelInput(params
));
            labels.push(move);
        });

        const inputTensor = tf.tensor2d(inputSequences);
        const labelTensor = tf.tensor1d(labels, 'int32');
        return { inputTensor, labelTensor }
    }

    public async train(trainData: [IMoveParams, EMove][],
epochs = 15) {
        const { inputTensor, labelTensor } =
this.prepTensors(trainData)
        const history = await this.model.fit(
            inputTensor,
            labelTensor,
            {
                epochs,
                batchSize: 32,
                validationSplit: 0.2,
                callbacks: [
                    tf.callbacks.earlyStopping({
                        monitor: "val_acc",
                        patience: 10,
                        minDelta: 0.01
                    }),
                ]
            }
        );
        return history
    }
```

```
    async predict(params: IMoveParams): Promise<EMove> {
        const inputTensor =
tf.tensor2d([transformIMoveParamsToModelInput(params)]);
        const predictions = this.model.predict(inputTensor)
as tf.Tensor;

        const predictedValues = predictions.dataSync();
        return Array.from(predictedValues)[0]
    }
}
```

A.4 Модуль реалізації LSTM моделі

```
export class LSTMModel {
    public model = tf.sequential();

    constructor(model?: tf.Sequential) {
        if(model) {
            this.model = model
        } else {
            this.model.add(tf.layers.masking({ maskValue: -
1, inputShape: [MAX_ROUNDS_PER_GAME, 2] }));
            this.model.add(tf.layers.lstm({ units: 50,
returnSequences: false }));
            this.model.add(tf.layers.dense({ units: 1,
activation: 'sigmoid' }));
        }

        this.model.compile({
            optimizer: 'adam',
            loss: 'binaryCrossentropy',
            metrics: ['accuracy']
        });
    }

    static padSequences(sequences: number[][][], maxLength:
number): number[][][] {
        return sequences.map(seq => {
            if((maxLength - seq.length) <= 0) {
                console.log(seq)
                console.log(maxLength)
            }
            const padding = Array(maxLength -
seq.length).fill([-1, -1]);
            return seq.concat(padding);
```

```
        });
    }

    public prepTensors(trainData: TGameRoundsHistory[]) {
        const inputSequences: number[][][] = [];
        const labels: number[] = [];

        trainData.forEach(gameHistory => {
            if(!gameHistory.length) return;
            const sequence = gameHistory.map(move =>
[move.m1, move.m2]);
            inputSequences.push(sequence);
            labels.push(gameHistory[gameHistory.length -
1].m2);  // Predict the m2
        });

        const paddedInputSequences =
LSTMModel.padSequences(inputSequences,
MAX_ROUNDS_PER_GAME);

        // Convert data to tensors
        const inputTensor =
tf.tensor3d(paddedInputSequences, [inputSequences.length,
MAX_ROUNDS_PER_GAME, 2]);
        const labelTensor = tf.tensor1d(labels, 'int32');
        return { inputTensor, labelTensor }
    }

    public async trainOn(games: TGameRoundsHistory[],
epochs = 30) {
        const { inputTensor, labelTensor } =
this.prepTensors(games);
        const history = await this.model.fit(inputTensor,
labelTensor, {
            epochs,
            batchSize: 32,
            validationSplit: 0.3,
            callbacks: [
                tf.callbacks.earlyStopping({
                    monitor: "val_acc",
                    patience: 5,
                    minDelta: 0.01
                }),
            ]
        });
```

```
        return history
    }

    async predict(history: TGameRoundsHistory):
Promise<number> {
        const { inputTensor } = this.prepTensors([history])
        const predictions = this.model.predict(inputTensor)
as tf.Tensor;

        const predictedValues = predictions.dataSync();
        return Array.from(predictedValues)[0]
    }
}
```

А.5 Модуль симуляції гри у «дилему в'язня»

```
export class PrisonersDilemmaGame {
    public roundsHistory: TGameRoundsHistory;
    public params: TGameMoveParamsHistory;

    private readonly SCORE_MATRIX = {
        [EMove.POSITIVE]: {
            [EMove.POSITIVE]: 2,
            [EMove.NEGATIVE]: 0,
        },
        [EMove.NEGATIVE]: {
            [EMove.POSITIVE]: 3,
            [EMove.NEGATIVE]: 1,
        }
    }

    private s1Score = 0;
    private s2Score = 0;

    public get lastRound() {
        return this.roundsHistory[this.roundsHistory.length
- 1]
    }

    constructor() {
        this.reset()
    }

    reset() {
        this.roundsHistory = []
```

```
        this.params = []
        this.s1Score = 0;
        this.s2Score = 0;
    }

    async simulate(
        strategy1: Strategy,
        strategy2: Strategy,
        rounds: number,
        autoReset = true,
        saveParamsHistory = false,
    ): Promise<TGameRoundsHistory> {
        const MAX_SCORE_PER_ROUND = 3;

        let cumulativeDiff: number;
        let diffCount: number;
        let avgDiff: number;

        if(
            strategy1 instanceof DynamicStrategy
            && strategy2 instanceof DynamicStrategy
        ) {
            const weights1 =
strategy1.mlp.model.getWeights()
            const weights2 =
strategy2.mlp.model.getWeights()

            cumulativeDiff = 0;
            diffCount = 0;

            for(let weightsIndex = 0; weightsIndex <
weights1.length; weightsIndex++) {
                const w1 =
weights1[weightsIndex].arraySync()
                const w2 =
weights2[weightsIndex].arraySync()

                const calcDiff = (a, b) => {
                    if(a.length !== b.length) throw new
Error("Cannot calculate diff")
                    for(let i = 0; i < a.length; i++) {
                        if(typeof a[i] === 'number' &&
typeof b[i] === 'number') {
                            cumulativeDiff += Math.abs(a[i]
- b[i]);
```

```
                            diffCount++;
                        } else {
                            calcDiff(a[i], b[i]);
                        }
                    }
                }
                calcDiff(w1, w2)
            }

            avgDiff = cumulativeDiff / diffCount;

            console.log("cumulativeDiff: ", cumulativeDiff)
            console.log("c: ", diffCount)
            console.log("avgDiff: ", avgDiff)
        }

        for (let i = 0; i < rounds; i++) {
            // console.log(`ROUND #${i}`)
            const lastRound: IGameRoundResult | null =
this.lastRound

            const s1Prediction = await
strategy1.predict(this.roundsHistory)
            const s2Prediction = await
strategy2.predict(this.roundsHistory)

            const sharedParams = {
                history: this.roundsHistory,
                roundNumber: i,
                maxRounds: rounds - 1,
                avgDiff,
                absDiff: cumulativeDiff,
                maxAbsDiff: diffCount,
                maxScore: rounds * MAX_SCORE_PER_ROUND,
            }
            const s1Params: IMoveParams = {
                ...sharedParams,
                last: lastRound
                    ? {
                        opponentMove: lastRound.m2,
                        selfMove: lastRound.m1
                    }
                    : null,
                scoreDifference: this.s1Score -
this.s2Score,
```

```
                opponentMovePredictorOutput: s1Prediction
            }
            const s2Params: IMoveParams = {
                ...sharedParams,
                last: lastRound
                    ? {
                        opponentMove: lastRound.m1,
                        selfMove: lastRound.m2
                    }
                    : null,
                scoreDifference: this.s2Score -
this.s1Score,
                opponentMovePredictorOutput: s2Prediction
            }
            const move1 = await
strategy1.makeMove(s1Params);
            const move2 = await
strategy2.makeMove(s2Params);

            this.s1Score +=
this.SCORE_MATRIX[move1][move2];
            this.s2Score +=
this.SCORE_MATRIX[move2][move1];

            if(saveParamsHistory)
                this.params.push([s1Params, s2Params])

            this.roundsHistory.push({
                m1: move1,
                m2: move2
            })
        }

        // Reset strategies for a new game
        strategy1.reset();
        strategy2.reset();

        const history = this.roundsHistory
        if(autoReset)
            this.reset()
        return history;
    }
}
```

А.6 Приклад реалізації статичної стратегії в форму класу StaticStrategy

```
export class WinStayLoseShiftStaticStrategy extends
StaticStrategy {
    private lastMove: EMove | null = null;  // Track the
player's own last move

    async makeMove(params: IMoveParams): Promise<EMove> {
        if (this.lastMove === null) {
            this.lastMove = EMove.POSITIVE;  // Cooperate
on the first move
            return EMove.POSITIVE;
        }

        if (params.last?.opponentMove === this.lastMove) {
            return this.lastMove;  // If the outcome was
good, repeat the last move
        }

        // If the outcome was bad, switch strategy
        this.lastMove = this.lastMove === EMove.POSITIVE ?
EMove.NEGATIVE : EMove.POSITIVE;
        return this.lastMove;
    }

    async predict(history): Promise<EMove> {
        return null;
    }
    reset(): void {
        this.lastMove = null;
    }
}
```

А.7 Приклад навчання динамічних стратегій на основі статичних

```
import { MAX_ROUNDS_PER_GAME } from "./constants";
import { randomIntFromInterval } from
"./helpers/rand.helper";
import { randomPairs } from "./helpers/random-pairs-from-
array.helper";
import { LSTMModel } from "./models/lstm";
import { PrisonersDilemmaGame } from "./simulation/game";
import { DynamicStrategy } from
"./strategies/dynamic/dynamic.stractegy";
import { AlwaysCooperateStaticStrategy } from
"./strategies/static/always-cooperate-static.strategy";
```

```
import { AlwaysDefectStaticStrategy } from
"./strategies/static/always-deflect-static.strategy";
import { GrimTriggerStaticStrategy } from
"./strategies/static/grim-trigger-static.strategy";
import { RandomStaticStrategy } from
"./strategies/static/random.strategy";
import { TitForTatWithForgivenessStaticStrategy } from
"./strategies/static/tit-for-tat-forgiveness-
static.strategy";
import { GenerousTitForTatStaticStrategy } from
"./strategies/static/tit-for-tat-generous-static.strategy";
import { TitForTatStaticStrategy } from
"./strategies/static/tit-for-tat-static.strategy";
import { WinStayLoseShiftStaticStrategy } from
"./strategies/static/win-stay-lose-shift-static.stretegy";
import { Strategy } from "./strategies/strategy";
import { TGameRoundsHistory } from "./types/game-
move.interface";
import { EMove } from "./types/move.enum";
import * as tf from '@tensorflow/tfjs-node';
import { IMoveParams } from "./types/move.params";

export class DefaultOpponentMovesPredictor {
    private static readonly VERSION = 11
    private static readonly NAME = `lstm-default-opponent-
move-predictor-${this.VERSION}/model.json`
    public static model: tf.Sequential = null;
    public static async load() {
        this.model = await
tf.loadLayersModel(`file://./${this.NAME}`) as
tf.Sequential
    }

    public static async trainOn(strategies: Strategy[]) {

        const lstm = new LSTMModel()
        const game = new PrisonersDilemmaGame()

        const trainStrategiesPairs =
randomPairs(strategies, 5000)
        const trainData: TGameRoundsHistory[] = []
        for(let [s1, s2] of trainStrategiesPairs) {
            const history = await game.simulate(
                s1,
                s2,
```

```
                randomIntFromInterval(1,
MAX_ROUNDS_PER_GAME)
            )
            trainData.push(history)
        }

        await lstm.trainOn(trainData)

        const testStrategiesPairs = randomPairs(strategies,
100)

        const testData: TGameRoundsHistory[] = []
        for(let [s1, s2] of testStrategiesPairs) {
            const history = await game.simulate(
                s1,
                s2,
                randomIntFromInterval(1,
MAX_ROUNDS_PER_GAME)
            )
            testData.push(history)
        }

        const { inputTensor, labelTensor } =
lstm.prepTensors(testData)
        const labels = labelTensor.arraySync()

        const predictions = lstm.model.predict(inputTensor)
as tf.Tensor;

        const predictedValues = predictions.dataSync();
        const predictedMoves =
Array.from(predictedValues).map(val => (val > 0.5 ?
EMove.POSITIVE : EMove.NEGATIVE)); // Convert probabilities
to moves

        // Evaluate the predictions against the actual
labels
        let correctPredictions = 0;
        labels.forEach((label, index) => {
            if (label === predictedMoves[index]) {
                correctPredictions++;
            }
        });

        const accuracy = correctPredictions /
```

```
labels.length;
        console.log(`Test Accuracy: ${accuracy * 100}%`);

        await lstm.model.save(`file://./lstm-${(new
Date()).toISOString().split('.').shift().replace(' ', '-
')}`)
    }
}

const main = async () => {
    await DefaultOpponentMovesPredictor.trainOn([
        new AlwaysCooperateStaticStrategy(),
        new AlwaysDefectStaticStrategy(),
        new TitForTatStaticStrategy(),
        new GrimTriggerStaticStrategy(),
        new TitForTatWithForgivenessStaticStrategy(),
        new WinStayLoseShiftStaticStrategy(),
        new GenerousTitForTatStaticStrategy()
    ])

    await DefaultOpponentMovesPredictor.load()
    console.log("DefaultMovePredictor loaded:")

console.log(DefaultOpponentMovesPredictor.model.summary())

    const exampleStrategies = [
        new AlwaysCooperateStaticStrategy(),
        new RandomStaticStrategy(),
        new AlwaysDefectStaticStrategy(),
        new TitForTatStaticStrategy(),
        new GrimTriggerStaticStrategy(),
        new TitForTatWithForgivenessStaticStrategy(),
        new WinStayLoseShiftStaticStrategy(),
        new GenerousTitForTatStaticStrategy()
    ]

    const trainData: [IMoveParams, EMove][][] = new
Array(exampleStrategies.length).fill([])

    const game = new PrisonersDilemmaGame()
    for(let strategyIndex = 0; strategyIndex <
exampleStrategies.length; strategyIndex++) {
        for(let opponentStrategy of exampleStrategies) {
            game.reset()
            const moves = await game.simulate(
```

```
                exampleStrategies[strategyIndex],
                opponentStrategy,
                MAX_ROUNDS_PER_GAME,
                false,
                true
            )
            const params = game.params
            for(let i = 0; i < MAX_ROUNDS_PER_GAME; i++) {

trainData[strategyIndex].push([params[i][0], moves[i].m1])
            }
        }
    }

    const dynamicStrategies: DynamicStrategy[] = []
    for(let strategyIndex = 0; strategyIndex <
exampleStrategies.length; strategyIndex++) {
        const dynamicStrategy = new DynamicStrategy()
        await
dynamicStrategy.train(trainData[strategyIndex])
        dynamicStrategies[strategyIndex] = dynamicStrategy
    }

    console.log("DynamicStrategies successfully
pretrained")
    const [
        AlwaysCooperateDynamicStrategy,
        RandomDynamicStrategy,
        AlwaysDefectDynamicStrategy,
        TitForTatDynamicStrategy,
        GrimTriggerDynamicStrategy,
        TitForTatWithForgivenessDynamicStrategy,
        WinStayLoseShiftDynamicStrategy,
        GenerousTitForTatDynamicStrategy,
    ] = dynamicStrategies

    console.log(await
AlwaysCooperateDynamicStrategy.makeMove({
        last: {
            selfMove: EMove.POSITIVE,
            opponentMove: EMove.NEGATIVE
        },
        history: [
            { m1: EMove.POSITIVE, m2: EMove.POSITIVE },
            { m1: EMove.POSITIVE, m2: EMove.NEGATIVE },
```

```
            { m1: EMove.POSITIVE, m2: EMove.NEGATIVE },
            { m1: EMove.POSITIVE, m2: EMove.NEGATIVE },
        ],
        roundNumber: 3,
        maxRounds: MAX_ROUNDS_PER_GAME,
        scoreDifference: -4,
        maxScore: MAX_ROUNDS_PER_GAME * 3,
        opponentMovePredictorOutput: EMove.NEGATIVE,
    }))

    console.log(await
AlwaysDefectDynamicStrategy.makeMove({
        last: {
            selfMove: EMove.NEGATIVE,
            opponentMove: EMove.POSITIVE
        },
        history: [
            { m1: EMove.NEGATIVE, m2: EMove.POSITIVE },
            { m1: EMove.NEGATIVE, m2: EMove.NEGATIVE },
            { m1: EMove.NEGATIVE, m2: EMove.NEGATIVE },
            { m1: EMove.NEGATIVE, m2: EMove.POSITIVE },
        ],
        roundNumber: 3,
        maxRounds: MAX_ROUNDS_PER_GAME,
        scoreDifference: 6,
        maxScore: MAX_ROUNDS_PER_GAME * 3,
        opponentMovePredictorOutput: 0.6,
    }))

    console.log(await TitForTatDynamicStrategy.makeMove({
        last: {
            selfMove: EMove.NEGATIVE,
            opponentMove: EMove.NEGATIVE
        },
        history: [
            { m1: EMove.POSITIVE, m2: EMove.NEGATIVE },
            { m1: EMove.NEGATIVE, m2: EMove.POSITIVE },
            { m1: EMove.POSITIVE, m2: EMove.NEGATIVE },
            { m1: EMove.NEGATIVE, m2: EMove.POSITIVE },
        ],
        roundNumber: 3,
        maxRounds: MAX_ROUNDS_PER_GAME,
        scoreDifference: 0,
        maxScore: MAX_ROUNDS_PER_GAME * 3,
        opponentMovePredictorOutput: 0.4,
```

```
    }))

    const history = await game.simulate(
        AlwaysCooperateDynamicStrategy,
        AlwaysDefectDynamicStrategy,
        MAX_ROUNDS_PER_GAME,
        true,
        false
    )
    console.log(history)
}
main()
    .catch(e => { throw e })
```

ДОДАТОК Б

(обов’язковий)

КОПІЇ НАУКОВИХ ПУБЛІКАЦІЙ

https://doi.org/10.31891/2219-9365-2024-80-29
УДК 004

ФОРКУН Юрій
Хмельницький національний університет
https://orcid.org/0000-0002-7906-4191
forkun@ridne.net
БАДЬОРА Ярослав
Хмельницький національний університет
y.badyora@gmail.com

МЕТОД РОЗРОБКИ ПРОГРАМНИХ ЗАСТОСУНКІВ ДЛЯ СИМУЛЯЦІЇ ТА УТВОРЕННЯ СТРАТЕГІЙ «ДИЛЕМИ В'ЯЗНЯ» НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ

Робота присвячена розробці методу розробки програмних застосунків для симуляції та утворення стратегій «дилеми в'язня» на основі нейронних методом динамічних стратегій вирішення «дилеми в'язня» за допомогою нейронних мереж і генетичних алгоритмів.

Ключові слова: метод динамічних стратегій, «дилема в'язня», симуляція, нейронна мережа, нейрон, стратегія, генетичний алгоритм.

FORKUN Yurii, BADYORA Yaroslav
Khmelnytskyi National University

THE METHOD OF DEVELOPING SOFTWARE APPLICATIONS FOR THE SIMULATION AND CREATION OF "PRISONER'S DILEMMA" STRATEGIES BASED ON NEURAL NETWORKS

The research examines strategies for resolving the Prisoner's Dilemma through neural networks and genetic algorithms. In particular, dynamic strategies as opposed to traditional static strategies — allow for behavioral adaptation based on a variety of input data and can be developed using neural networks.

Through the use of genetic algorithms, the neural network architecture and learning parameters are optimized by simulating the evolution of strategies within a competitive dynamic. This study shows how these kinds of dynamic strategies can be used to better understand the behavior of individual agents operating in economies and societies where people's selfish interests frequently collide with the well-being of the group as a whole.

By taking this approach, the Prisoner's Dilemma can be modeled to a new level and more realistic behavioral strategies can be constructed in repeated dilemma scenarios. It also makes it easier to investigate the different factors that influence the evolution of strategies.

Keywords: prisoner's dilemma, game theory, simulation, neural network, neuron, strategy, genetic algorithm.

ПОСТАНОВКА ПРОБЛЕМИ У ЗАГАЛЬНОМУ ВИГЛЯДІ ТА ЇЇ ЗВ'ЯЗОК ІЗ ВАЖЛИВИМИ НАУКОВИМИ ЧИ ПРАКТИЧНИМИ ЗАВДАННЯМИ

«Дилема в'язня» є однією з класичних моделей теорії ігор, яка підкреслює боротьбу між самомаксимізацією та добробутом спільноти. У цій дилемі гравці стикаються з вибором: співпрацювати або втекти, де втеча приносить найбільшу винагороду, незалежно від вибору іншого гравця. Однак, якби обидва гравці навіть вирішили втекти, результат для обох був би меншим, ніж результат співпраці. Цей парадокс описує міркування про явище прийняття рішень у короткостроковій перспективі часто через відсутність раціональності, яка переважає серед членів групи в довгостроковій перспективі, дуже поширене явище в більшості соціально-економічних і біологічних систем.

Класичні розв'язання дилеми «в'язня» свідчать про прийняття статичних стратегій, таких як «око за око», або «ніколи не співпрацювати, завжди відмовлятися». Однак вони демонструють свою застосовність у статичних середовищах, без урахування плинності середовища та можливих моделей поведінки гравців з часом [1]. Це робить звичайні стратегії непридатними для моделювання ситуацій реального світу, де стратегії гравців є міжчасовими та розвиваються у відповідь на дії інших. Наприклад, дослідження [3] показують, що в складних системах, які включають довгострокові взаємодії, статичні стратегії з самого початку не обіцяють бажаних рішень, отже, причина, чому існує потреба в нових підходах до стратегічного моделювання.

Застосування нейронних мереж у модифікації стратегій «дилеми в'язня» є новим підходом, оскільки воно формує такі типи стратегій, які можуть навчатися та змінювати свою поведінку відповідно до попередніх взаємодій. Вони також можуть з'ясувати складні та складні нелінійні зв'язки між рішеннями,

прийнятими гравцями, та результатами таких рішень – отже, краща поведінка, яка буде використовуватися під час навчання в змагальному середовищі [1]. Подібним чином, результати дослідження в статті [4] демонструють, що нейронні мережі здатні досягти більшої гнучкості та адаптивних реакцій у поведінкових стратегіях гравців, ніж звичайні методи, що є вирішальним в умовах високої динаміки та високої невизначеності системи.

Інтеграція нейронних мереж і генетичних алгоритмів видається ефективною з точки зору оптимізації стратегії в дилемі ув'язненого. Генетичні алгоритми дозволяють шукати оптимальні структури нейронних мереж і параметри їх навчання, а також розвивати стратегії в конкурентному середовищі. Наприклад, у дослідженні [4] наголошується на застосуванні генетичних алгоритмів для еволюції кооперативних стратегій у повторюваній дилемі: «всі або жодна». Це дозволяє знайти більш надійні стратегії, які можуть пристосовуватися до змін середовища або інших гравців. Вона дає також гарну можливість вивчити еволюцію довгострокової стратегії, особливо вплив на неї оточуючих та інших агентів.

Ще однією важливою перевагою використання нейронних мереж є їх здатність моделювати складні соціальні та економічні взаємодії, де поведінка агентів залежить не лише від поточних дій, але й від минулого досвіду та очікуваних майбутніх змін. Наприклад, [7] вивчає алгоритми онлайн-навчання, які дозволяють моделям адаптуватися до мінливих умов під час гри, змінюючи свої стратегії у відповідь на дії інших агентів і зміни середовища. Це дозволяє створювати точніші моделі взаємодії в соціальних системах, де гравці можуть змінювати свої стратегії в залежності від умов гри.

Дослідження також підкреслюють важливість адаптивних стратегій моделювання задачі «дилеми в'язня», особливо в ситуаціях, коли правила гри змінюються, або коли інформація про дії інших гравців обмежена [6]. Нейронні мережі можна навчити та адаптувати до нових умов, враховуючи не лише поточні, але й майбутні події, що дозволяє їм приймати більш оптимальні рішення у складних та невизначених середовищах.

У практичному контексті використання нейронних мереж для моделювання адаптивної поведінки в «дилемі в'язня» може бути корисним для вирішення проблем колективного управління ресурсами, моделювання економічного співробітництва та прийняття рішень в умовах невизначеності. Дослідження [5] вказує на те, що адаптивні стратегії, засновані на нейронних мережах, можуть підвищити ефективність колективного прийняття рішень за таких обставин, при цьому враховуючи зміни в моделях поведінки та міжагентній взаємодії. Це дозволяє розробляти нові способи вирішення задач у складних системах, у яких зміна поведінки гравців може критично вплинути на результат взаємодії.

Важливим аспектом цього дослідження є розуміння різних факторів, таких як соціальні норми чи винагороди, які можуть вплинути на еволюційні стратегії в динамічних середовищах. Дослідження [2] показує, що моделювання адаптивної поведінки в таких системах може допомогти зрозуміти, як різні стратегії співпраці та зради розвиваються в умовах конкуренції; що є важливим для розробки ефективних моделей управління в складних соціально-економічних системах. Зокрема, врахування змін у винагороді за співпрацю чи відмову дозволяє точніше моделювати поведінку агента та ідентифікувати, які стратегії є найбільш надійними в довгостроковій перспективі.

Важливим аспектом розробки систем штучного інтелекту (ШІ) для вирішення повторюваної дилеми в'язня є розробка нових стратегій, які враховують минулі взаємодії. Наприклад, концепція пам'яті-один, яка використовується при моделюванні взаємодії людини з комп'ютером, дозволяє агентам враховувати минулі кроки під час прийняття рішень. Це дає їм змогу будувати системи, які передбачають поведінку опонента на основі минулого досвіду, тим самим підвищуючи ефективність кооперативних стратегій у довгостроковій перспективі. Такий підхід демонструє значні переваги порівняно зі звичайними моделями без урахування історії попередніх дій [9].

Поєднання нейронних мереж з еволюційними алгоритмами, такими як генетичні, є ще одним перспективним підходом. Наприклад, дослідження показали, що генетичні алгоритми разом з нейронною мережею дозволяють розробляти адаптивні стратегії в мінливому середовищі, а також здатні розвиватися, що обов'язково важливо для стабільної співпраці між агентами [3]. Крім того, такий підхід дозволяє агентам швидко знаходити оптимальні стратегії в нових умовах, реагуючи на зміни в поведінці опонента.

Адаптивність стратегій також забезпечується завдяки онлайн-навчанню, що дозволяє агентам змінювати свої дії в реальному часі. Використання підкріплювального навчання у поєднанні з нейронними мережами відкриває нові можливості для створення агентів, які можуть не лише ефективно імітувати людську поведінку, але й адаптуватися до її змін у динамічних умовах гри [7]. Це особливо важливо у ситуаціях, де поведінка інших гравців може бути непередбачуваною, що вимагає швидкої реакції та корекції власної стратегії.

Online-навчання забезпечує адаптацію стратегій, що дозволяє агентам змінювати свої дії в режимі реального часу. Використання навчання з підкріпленням у поєднанні з нейронними мережами відкриває

нові шляхи для створення агентів, які можуть не тільки ефективно моделювати поведінку людини, але й адаптуватися до змін у динамічному ігровому середовищі [7]. Це особливо важливо в ситуаціях, коли поведінка інших гравців непередбачувана і вимагає швидкої реакції та коригування власної стратегії.

Використовуючи нейронні мережі для моделювання їхніх стратегій, можна вивести приховані стимули, що стоять за цими діями, а також дозволити їм швидко адаптуватися в новому середовищі. Ці моделі здатні використовувати великі історичні набори даних взаємодії завдяки методам штучного інтелекту машинного навчання, що потенційно дає змогу прогнозувати майбутні дії опонентів на основі попередніх взаємодій між кількома агентами [1]. Це допомагає вдосконалювати розроблені стратегії, які можуть витримати будь-які мінливі обставини гри.

Таким чином, останні дослідження показують, що нейронні мережі, інтегровані з еволюційними методами, такими як генетичні алгоритми та клітинні автомати, забезпечують потужні засоби вирішення дилеми ув'язненого. Вони створюють адаптивні моделі, які ефективно реагують на зміни в поведінці інших акторів і забезпечують стабільну співпрацю в динамічних середовищах[7][9]. Цей підхід відкриває нові горизонти для досліджень теорії ігор і ШІ, спрямованих на розробку надійних і гнучких стратегій для вирішення складних проблем у багатоагентних системах.

ФОРМУЛЮВАННЯ ЦІЛЕЙ СТАТТІ

Мета роботи – удосконалення досліджень методів вирішення "дилеми в'язня", шляхом приближення методології симуляції стратегій до поведінки у реальному світі.

ВИКЛАД ОСНОВНОГО МАТЕРІАЛУ

Більшість стратегій для розв'язання "дилеми в'язня", які наразі використовуються, ґрунтуються на статичних стратегічних підходах, згідно з аналізом цих підходів. Стабільні умови сприяють ефективності цих стратегій. Однак умови взаємодії агентів у реальному світі варіюються через різноманітні фактори, особливо в соціальних та економічних системах, що робить статичні стратегії більш обмеженими та менш успішними. Необхідні нові методи стратегічного моделювання через динамічні зміни в поведінці агентів, викликані непередбачуваними зовнішніми умовами. Необхідно впровадити техніки, які можуть реагувати на змінні умови навколишнього середовища та відповідно коригувати стратегії, щоб вирішити цю проблему.

Проаналізувавши існуючі методи формування стратегій вирішення дилеми, можна зрозуміти що існуючі стратегії максимально ефективні тільки в сталих, та малозмінних середовищах. Такі математично прості стратегії важко відобразити аналогією з реального світу, а значить, важко знайти прикладне рішення для їх використання.

Щоб вирішити цю проблему, на основі існуючих досліджень, нами було запропоновано метод, що дозволить зробити генерацію та навчання таких стратегій більш гнучкою, а стратегії що вийдуть у результаті навчання, будуть розумними та поводити себе аналогічно мінливим змінам свого середовища.

Таким чином, ми пропонуємо змоделювати стратегію з вирішення "дилеми в'язня", за допомогою нейронних мереж, які дозволяють агентам мати складну модель поведінки, таким чином ефективно коригуючи свої дії відповідно до різноманітних варіацій середовища. Ці нейронні мережі допоможуть створювати оптимальні стратегії у великих наборах даних і залучати попередні історії взаємодій, таким чином сприяючи стабільній співпраці гравців. Такий підхід назвемо методом динамічних стратегій.

Цей метод використовує інформацію про стратегію суперника та поточний стан гри як вхідні данні нейронної мережі, що репрезентує стратегію гри. Ці методи дозволяють агентам адаптуватися до змін у поведінці супротивника протягом тривалих ігрових сесій, що є критично важливим у динамічних середовищах, де взаємодії агентів непередбачувані. Ці моделі можуть навчатися в реальному часі через зворотне поширення помилки, що робить цей метод особливо ефективним для швидкої адаптації під час гри [7]. На відміну від традиційних статичних стратегій, які не змінюються в залежності від поведінки супротивників, нейронні мережі дозволяють адаптуватися. Вони мають здатність оцінювати попередні дії, робити з них висновки та миттєво коригувати свій підхід.

Нейронні мережі були обрані для розробки динамічних стратегій через ряд факторів. По-перше, складні, нелінійні зв'язки між вхідними даними та виходами можуть бути змодельовані нейронними мережами. Це особливо важливо в контексті «дилеми в'язня», де гравці повинні постійно адаптуватися, оскільки рішення, які вони приймають, часто мають непередбачувані наслідки. Нейронні мережі дуже точно прогнозують поведінку супротивника, оскільки вони обробляють великі обсяги даних і навчаються на основі взаємодій. По-друге, вони дозволяють розробляти гнучкі стратегії, які можна коригувати під час гри. Для повторюваних взаємодій у грі, наприклад, здатність мереж «пам'ятати» минулі стани та дії можуть забезпечуватися використанням моделей LSTM та GRU, що і є одним з ключових елементів нашого методу.

Важливо врахувати ряд факторів, пов'язаних з грою, при розробці нейронної мережі, яка може ефективно моделювати стратегії «дилеми в'язня». Оскільки взаємодії між агентами є вирішальними для прийняття рішень, структура мережі повинна враховувати як поточний стан гри, так і їхні минулі взаємодії.

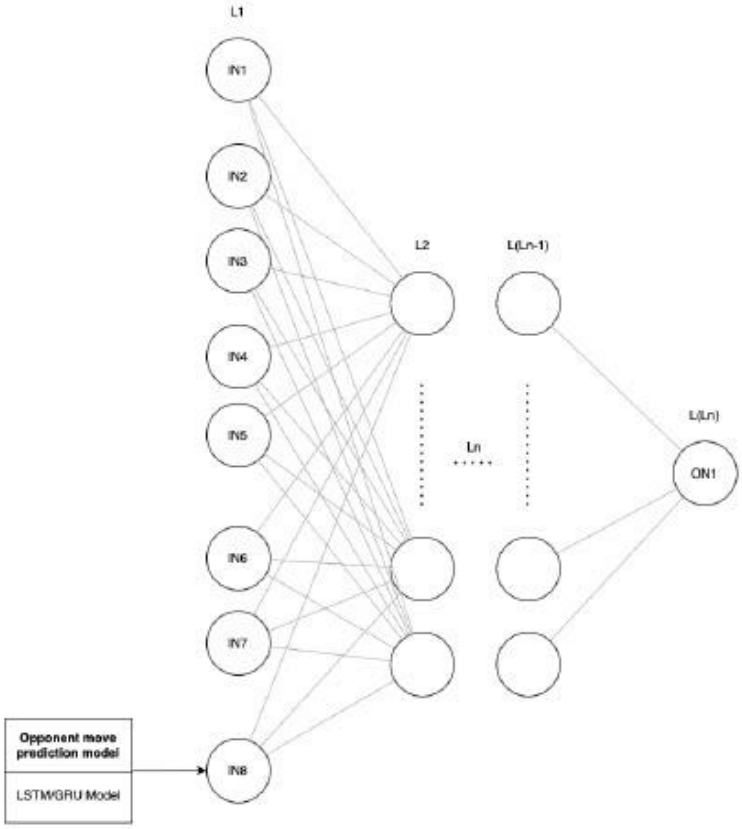


Рис. 1. Схема нейронної мережі стратегії

Щоб забезпечити найточнішу оцінку поведінки супротивника, пропонується дотримуватись наступної схеми побудови нейронної мережі агента (Рисунок 1):

Нейрон IN1 повинен виконувати функцію лічильника ходів, допомагаючи агенту визначити, чи знаходиться він на ранній, середній чи пізній стадії гри. Це важливо, оскільки поведінка агентів може змінюватися в залежності від етапу гри: гравці можуть бути більш співпрацюючими на початку, але пізніше можуть виявитися більш агресивними. В результаті модель може змінювати свою поведінку у відповідь на зміни стратегії суперника завдяки цій інформації.

Нейрони IN2 та IN3 повинні контролювати бали агента та його суперника. Це надасть агенту можливість оцінити стан гри на той момент і вирішити, чи залишитися на кооперативних тактиках, чи перейти до більш конкурентних. Більш точну оцінку поточного стану гри можна зробити, якщо нормалізація балів неможлива під час симуляції. Це можна досягти, представляючи різницю між оцінками агентів за допомогою одного нейрона.

Нейрони IN4 та IN5 запроектовані таким чином, щоб аналізувати минулі дії як супротивника, так і агента, що дозволяє нейронній мережі вчитися на попередніх взаємодіях і коригувати свою стратегію на основі цієї інформації. Це особливо важливо для розробки адаптивних стратегій, які можуть ефективно реагувати на зміни в поведінці супротивника під час гри. Дослідження, що використовують стратегії, основані на пам'яті, продемонстрували, що врахування минулих дій супротивників може суттєво підвищити ефективність стратегії.

Проектування нейронів IN6 та IN7, які оцінюють, наскільки подібні стратегії агентів одна до одної, є ще однією важливою особливістю цієї моделі. Агенти можуть використовувати цю інформацію, щоб визначити, які гравці найімовірніше співпрацюватимуть, а які можуть цього не зробити. Окрім сприяння співпраці між подібними стратегіями, цей підхід також допомагає запобігти конфлікту з тими, чиї стратегії несумісні. Ця здатність ідентифікувати подібні стратегії покращує як кооперативну, так і конкурентну динаміку в складних багатокористувацьких сценаріях.

Нейрон IN8 повинен бути використаний для захоплення виходу моделі LSTM/GRU під час повторних взаємодій з одним і тим же супротивником. Нейрон IN8 потрібно використовувати для захоплення виходу моделі LSTM/GRU під час повторних взаємодій з одним і тим же супротивником. Протягом гри цей нейрон допомагає виявляти поведінкові патерни суперника. Ця здатність не лише дозволяє прогнозувати наступний хід супротивника, але й полегшує внесення змін у власну стратегію ефективним чином на основі зібраної інформації.

Ця структура репрезентації стратегії особливо добре працює під час тривалих ігрових сесій, коли стратегія супротивника може змінюватися у відповідь на нову інформацію. Додаткові дослідження підтвердили ефективність цієї методології в моделюванні адаптивної поведінки, коли агенти ефективно підвищували свій рівень співпраці, прогнозуючи дії супротивників за допомогою обмежених тактик пам'яті. Такі моделі можна навчати в реальному часі за допомогою зворотного поширення, що підвищує адаптивність і гнучкість стратегії.

Рис. 2. Діаграма алгоритму процесу навчання динамічних стратегій

Данна структура зберігає алгоритм дій стратегії у різних ситуаціях в достатньо компактному режимі. Проте, навчання даних нейронних мереж для їх подальшого використання для вирішення «дилеми в'язня», може зайняти багато ресурсів, бо навчання методом генетичних потребує багато ресурсів для роботи. Щоб уникнути зайвих витрат, та прискорити навчання, нами сформовано поетапний алгоритм для підготовки та навчання стратегій. Цей процес включає в себе такі етапи (Рисунок 2):

1. *Програмна реалізація ефективних статичних стратегій*. На цьому етапі впроваджуються стратегії, які вже довели свою ефективність у попередніх дослідженнях, такі як «око за око» або «безумовна співпраця». Цей крок є вирішальним, оскільки закладає основу для порівняння з динамічними стратегіями. Впровадження встановлених стратегій забезпечує базу, що дозволяє чіткіше оцінити ефективність динамічних стратегій, які будуть розроблені пізніше.
2. *Розробка симуляційного середовища*. Створено симуляційне середовище для Дилеми ув'язненого з змінними параметрами, що дозволяє гнучко налаштовувати умови гри та тестувати різні стратегії. Ця гнучкість є необхідною для дослідження того, як різні стратегії працюють у різних сценаріях.

Маніпулюючи екологічними змінними, ми можемо краще зрозуміти динаміку взаємодій агентів та умови, які сприяють або співпраці, або конкуренції.

3. *Програмна реалізація динамічних стратегій*. Цей етап передбачає інтеграцію моделей нейронних мереж з динамічним оновленням стратегій у реальному часі. Впровадження динамічних стратегій є необхідним для того, щоб агенти могли навчатися та адаптуватися під час гри. Ця здатність є життєво важливою в умовах, де супротивники можуть змінювати свої стратегії, оскільки вона дозволяє агентам ефективно реагувати на нові виклики та підвищувати свої шанси на успіх.

4. *Генерація агентів*. На основі випадкових початкових значень ваг нейронних мереж генеруються динамічні стратегії для змагання зі статичними агентами. Цей процес генерації є вирішальним, оскільки він вводить різноманітність у поведінці агентів, що призводить до більш надійних симуляцій. Створюючи різноманітних агентів з різними початковими стратегіями, ми можемо спостерігати, як ці агенти взаємодіють і еволюціонують з часом, що надає цінні відомості про ефективність різних стратегічних підходів.

5. *Наближення динамічних стратегій до роботи статичних*. Використання глибокого навчання дозволяє динамічним стратегіям тісно узгоджуватися з поведінкою статичних стратегій, що полегшує початкову фазу навчання та оптимізації. Ця узгодженість є важливою, оскільки вона дозволяє динамічним стратегіям використовувати переваги перевірених статичних стратегій, зберігаючи при цьому гнучкість для адаптації до змін. Цей гібридний підхід дозволить покращити ефективність проходження наступних етапів навчання.

6. *Симуляція ігор та навчання методом штучного відбору*. Під час симуляцій використовується метод клітинних автоматів для організації взаємодій агентів, моделюючи складні середовища з численними учасниками. Цей метод є важливим для оцінки того, як агенти з подібними або різними стратегіями взаємодіють у різних ситуаціях і як їхні стратегії еволюціонують з часом. Процес моделювання дозволяє досліджувати нові поведінкові патерни, які виникають внаслідок взаємодії агентів, що сприяє глибшому розумінню співпраці та конкуренції в багатоагентних системах. Змінюючи покоління, методом штучного відбору ми можем вивести цілі сімейства динамічних стратегій, що за допомогою відповідних нейронів зможуть відрізняти своїх від чужих, навіть без аналізу поведінки.

7. *Збір та аналіз результатів*. Останній етап передбачає аналіз отриманих результатів симуляцій для визначення найбільш ефективних динамічних стратегій для різних ігрових умов. Цей аналіз є критично важливим для валідації запропонованих методів та розуміння, які стратегії дають найкращі результати в конкретних ситуаціях. Оцінюючи ефективність різних стратегій, ми можемо вдосконалити наші моделі та визначити напрямки майбутніх досліджень.

Таким чином, запропонований метод динамічних стратегій, заснований на нейронних мережах та генетичних алгоритмах, суттєво підвищує адаптивність стратегій у порівнянні зі статичними підходами. Це створює основу для розробки більш ефективних моделей поведінки в динамічних середовищах, роблячи цей підхід цінним для вирішення широкого спектра викликів у багатоагентних системах. Оскільки ця галузь продовжує розвиватися, інтеграція цих технік може призвести до більш складних симуляцій, які краще відображають реальні взаємодії, що в кінцевому підсумку покращить наше розуміння співпраці та конкуренції в різних контекстах.

ВИСНОВКИ З ДАНОГО ДОСЛІДЖЕННЯ І ПЕРСПЕКТИВИ ПОДАЛЬШИХ РОЗВІДОК У ДАНОМУ НАПРЯМІ

У цьому дослідженні подано метод розробки програмних застосунків для симуляції та утворення стратегій «дилеми в'язня» на основі нейронних мереж та описано реалізацію динамічних стратегій, застосованих через нейронні мережі, для вирішення «дилеми в'язня». Головною інновацією є використання нейронних мереж, таких як LSTM та GRU, в комбінації з вхідними показниками, що дозволяє агентам змінювати свої стратегії в режимі реального часу відповідно до змінюваних умов гри та попередніх взаємодій. Протягом тривалих сесій цей динамічний підхід покращує реакцію та гнучкість агентів, дозволяючи їм точно передбачати та протидіяти поведінці супротивників.

Ці нейронні мережі додатково оптимізуються шляхом використання генетичних алгоритмів, що підвищує здатність агентів адаптуватися та покращувати свої тактики. Використовуючи гібридний підхід, створюється потужна, гнучка система, яка може як адаптуватися до конкурентних умов, так і підтримувати співпрацю.

Подальші дослідження можуть зосередитися на оптимізації цих адаптивних нейронних тактик, вивченні дедалі складніших ігрових ситуацій та розширенні структур нейронних мереж для підвищення гнучкості. Поєднання нейронних мереж та динамічних стратегій відкриває нові перспективи для розвитку адаптивної поведінки в багатагентних системах.

References

1. GRASP like algorithm. José Barahona da Fonseca // Computer Aided Chemical Engineering, Volume 24, 2017, Pages 279-284. https://doi.org/10.1016/S1570-7946(07)80070-8
2. Optimization of computer programming based on mathematical models of artificial intelligence algorithms. Yuhui Zheng//
3. Computers and Electrical Engineering. Volume 110, September 2023, 108834. https://doi.org/10.1016/j.compeleceng.2023.108834 Nikoleta E. Glynatsi, Vincent Knight and Marc Harper. Properties of Winning Iterated Prisoner's Dilemma Strategies, 2024.
4. Domajnko, Martin. Iterated prisoner's dilemma and survival of the fittest from an ecological perspective. Proceedings of the 18th Student Computing Research Symposium (SCORES'22) 2022.
5. Ferraz, V., Pitz, T. Analyzing the Impact of Strategic Behavior in an Evolutionary Learning Model Using a Genetic Algorithm. Comput Econ 63, 437–475 (2024). https://doi.org/10.1007/s10614-022-10348-1
6. Rui Dong, Xinghong Jia, Xianjia Wang, Yonggang Chen. Optimal Tag-Based Cooperation Control for the "Prisoner's Dilemma", 2020. https://doi.org/10.1155/2020/8498613
7. Montero-Porras, E., Grujić, J., Fernández Domingos, E. et al. Inferring strategies from observations in long iterated Prisoner's dilemma experiments. Sci Rep 12, 7589 (2022). https://doi.org/10.1038/s41598-022-11654-2
8. Lin, B., Bouneffouf, D., Cecchi, G. (2022). Online Learning in Iterated Prisoner's Dilemma to Mimic Human Behavior. In: Khanna, S., Cao, J., Bai, Q., Xu, G. (eds) PRICAI 2022: Trends in Artificial Intelligence. PRICAI 2022. Lecture Notes in Computer Science, vol 13631. Springer, Cham. https://doi.org/10.1007/978-3-031-20868-3_10
9. de Paulo, K.P., Estombelo-Montesco, C.A. & Tejada, J. New memory-one strategies of the Iterated Prisoner's Dilemma: a new framework to programmed human-AI interaction. Discov Psychol 4, 20 (2024). https://doi.org/10.1007/s44202-024-00133-6
10. Umberto Cerruti, Simone Dutto, Nadir Murru, A symbiosis between cellular automata and genetic algorithms, Chaos, Solitons & Fractals, Volume 134, 2020, 109719, ISSN 0960-0779, https://doi.org/10.1016/j.chaos.2020.109719

ДОДАТОК В

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

МЕТОД РОЗРОБКИ ПРОГРАМНИХ ЗАСТОСУНКІВ ДЛЯ СИМУЛЯЦІЇ ТА СТВОРЕННЯ СТРАТЕГІЙ “ДИЛЕМИ В’ЯЗНЯ” НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ

Розробив: студент Бадьора Ярослав Михайлович
Керівник: канд. тех. н., доцент Форкун Юрій Вікторович

Актуальність теми

1. **Наявність наукових публікацій з обраної теми**:
   З поширенням використання "штучного" інтелекту, концепт "дилеми в'язня" почав знову популяризуватись у науковій спільноті. Це підтверджується зростаючою кількістю робіт, присвячених використанню "дилеми вязня" у сучасних сферах та завданнях, для моделювання взаємодій у складних системах взяємодії між агентами в тому числі.

2. **Виявлення прогалин у дослідженнях**:
   Існуючі підходи до вирішення "дилеми в'язня" здебільшого базуються на використанні статичних стратегій, які не враховують динаміку взаємодій між агентами та зміни у поведінці гравців. Відсутність моделей, що дозволяють агентам навчатися і адаптуватися в реальному часі, створює прогалину, яку потрібно заповнити.

3. **Зв'язок із сучасними тенденціями**:
   Сучасні технології, такі як машинне навчання та нейронні мережі, дозволяють моделювати складні нелінійні залежності та динамічну поведінку. Це робить проблему адаптивних стратегій актуальною для багатогранних застосувань, від економіки до штучного інтелекту.

4. **Практична та наукова значущість**:
   Розробка методів для симуляції стратегій "дилеми в'язня" з використанням нейронних мереж дозволяє вирішувати прикладні задачі у сфері соціальної взаємодії, аналізу конфліктів та управління ресурсами. Крім того, ці дослідження сприяють розвитку теорії ігор та багатогранних систем.

Мета і завдання дослідження

Мета – удосконалення досліджень методів вирішення "дилеми в'язня", шляхом приближення методології симуляції стратегій до поведінки у реальному світі.

Завдання:

- Аналіз сучасних підходів до вирішення дилеми в'язня.
- Розробка архітектури нейронної мережі для моделювання адаптивних стратегій.
- Впровадження симуляційного середовища для навчання стратегій методом генетичних алгоритмів.
- Оцінка ефективності запропонованого підходу через порівняння динамічних стратегій зі статичними.

Об’єкт і предмет дослідження

- **Об’єкт**: розробка та впровадження програмного забезпечення для дослідження стратегій вирішення “дилеми в’язня”.
- **Предмет**: адаптивні стратегії для "дилеми в’язня", побудовані на основі нейронних мереж із використанням методів машинного навчання та генетичних алгоритмів.

Методи дослідження

Для реалізації поставленої мети було використано сучасні методи математичного моделювання, машинного навчання та симуляції. Основні концепти дослідження включають:

1. **Формалізація задачі**:
   Дилема в’язня представлена як повторювана гра між агентами, що взаємодіють у динамічному середовищі. Це дозволяє враховувати вплив попередніх дій на майбутню поведінку учасників.

2. **Моделі нейронних мереж**:
   Використано нейронні мережі, які здатні аналізувати нелінійні залежності, враховувати історію взаємодій та адаптувати стратегії в реальному часі. Для реалізації адаптивних стратегій обрано архітектури LSTM, що забезпечують збереження інформації про попередні стани гри.

3. **Генетичні алгоритми**:
   Для оптимізації параметрів навчання нейронних мереж та пошуку ефективних стратегій використано генетичні алгоритми. Цей підхід дозволяє натурально еволюційно покращувати динамічні стратегії через адаптацію до різних умов гри.

4. **Симуляційне середовище**:
   Створено симуляційне середовище для тестування стратегій з можливістю налаштування параметрів гри. Це середовище забезпечує варіативність сценаріїв і дозволяє оцінювати ефективність стратегій у різних контекстах, включаючи кооперацію та конкуренцію.

5. **Аналіз даних**:
   Результати симуляцій аналізувалися для виявлення найефективніших стратегій у різних ігрових умовах. Оцінка ефективності базувалася на рівні співпраці, стабільності стратегій та здатності до адаптації.

Наукова новизна отриманих результатів

Вперше використано динамічні параметри для роботи стратегії у формі нейронної мережі, одночасно з моделлю LSTM для передбачення ходів опонента, що дозволяє стратегії донавчатись прямо під час сесій симуляцій гри та змінювати свою поведіку нелінійно

Дослідження предметної області та постановка задачі

1. **Аналіз предметної області**:
   Дилема в'язня є однією з фундаментальних моделей у теорії ігор. Вона демонструє конфлікт між індивідуальною вигодою та колективною співпрацею.

2. **Основні обмеження традиційних стратегій**:
   - Статичні стратегії, як-от "око за око", не враховують динамічних змін у поведінці супротивника.
   - Відсутність можливості до адаптації у реальному часі знижує ефективність підходів у складних середовищах.
   - Недостатнє врахування минулих взаємодій призводить до втрати інформації, важливої для прогнозування поведінки.

3. **Вимоги до сучасних стратегій**:
   - Адаптивність до змін у поведінці гравців.
   - Можливість навчатися на основі історії взаємодій.
   - Ефективне використання технологій, таких як генетичні алгоритми, для навчання стратегій

4. **Постановка задачі**:
   Необхідно розробити метод створення та симуляції адаптивних стратегій для дилеми в'язня, який:
   - Використовує нейронні мережі для моделювання поведінки агентів.
   - Застосовує генетичні алгоритми для оптимізації моделей.
   - Забезпечує ефективну адаптацію у динамічних середовищах.

5. **Цільова модель**:
   - Агенти повинні аналізувати минулі дії супротивників та адаптувати свої стратегії у реальному часі.
   - Модель має забезпечувати прогнозування поведінки супротивників на основі їхніх дій та стану гри.
   - Реалізація повинна враховувати етапи гри, що впливають на зміну тактики гравців.

6. **Важливість задачі**:
   - Створення інструментів для симуляції стратегій є ключовим для розв'язання практичних проблем, пов'язаних із соціальною співпрацею, аналізом конфліктів і прийняттям рішень у багатокористувацьких системах.

Метод

1. **Основна концепція**:
   Використання нейронних мереж для адаптивного управління стратегіями у повторюваній дилемі в'язня. Ключовою ідеєю є динамічна адаптація стратегій через навчання на основі взаємодій агентів та поточного стану гри.

2. **Архітектура нейронної мережі**:
   - Використання моделей LSTM/GRU для обробки послідовностей даних (історії взаємодій).
   - Вхідні дані включають:
     - Стан гри (результати попередніх раундів).
     - Дії супротивників у минулих раундах.
     - Поточний раунд гри для врахування її етапу.

3. **Прогнозування дій супротивника**:
   Нейронні мережі дозволяють агентам аналізувати поведінкові патерни супротивників та прогнозувати їхні дії, що є основою для корекції власної стратегії.

4. **Генетичні алгоритми для оптимізації**:
   - Еволюційний підхід для знаходження оптимальних параметрів навчання нейронних мереж.
   - Генетичні алгоритми використовуються для відбору найбільш ефективних стратегій у процесі симуляцій.
   - Відбір базується на рівні кооперації, стабільності стратегії та здатності адаптуватися до змін.

5. **Методика розробки стратегій**:
   - Початкове створення базових стратегій (наприклад, "око за око").
   - Впровадження динамічних стратегій через навчання нейронних мереж.
   - Тестування стратегій у симуляційних середовищах для аналізу їхньої ефективності.

6. **Симуляційне середовище**:
   Гнучка платформа для тестування стратегій у різних умовах, що дозволяє варіювати параметри гри та оцінювати взаємодію агентів.

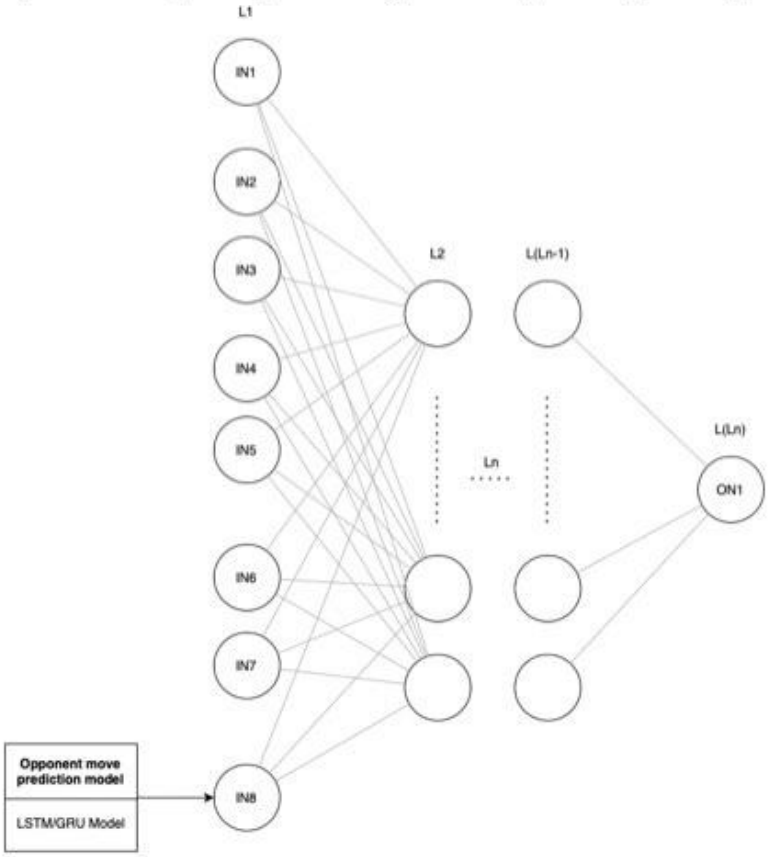
Реалізація методу

1. **Етапи реалізації методу**:
   Розробка методу симуляції стратегій включала кілька послідовних кроків:
   - Створення ефективних статичних стратегій для базового тестування.
   - Розробка симуляційного середовища з можливістю динамічної взаємодії агентів.
   - Впровадження динамічних стратегій на основі нейронних мереж із підтримкою адаптації у реальному часі.
   - Інтеграція генетичних алгоритмів для оптимізації стратегій.

2. **Архітектура програмного забезпечення**:
   - **Модуль нейронної мережі**: Моделі LSTM/GRU для обробки вхідних даних, що включають поточний стан гри, результати попередніх раундів та дії супротивників.
   - **Модуль симуляції**: Відповідає за проведення ігор між агентами у різних сценаріях. Дозволяє тестувати різні стратегії у динамічних умовах.
   - **Модуль оптимізації**: Використовує генетичні алгоритми для пошуку ефективних стратегій шляхом еволюційного відбору.

3. **Обробка даних у нейронних мережах**:
   - Вхідні нейрони відповідають за збір інформації про стан гри, дії агентів та етап гри.
   - Схема побудови нейронної мережі враховує як короткострокові, так і довгострокові взаємодії, дозволяючи агентам враховувати зміни у стратегіях супротивників.

Репрезентація стратегії у вигляді нейро-мережі



4. **Симуляційне середовище**:
   - Забезпечує змінюваність умов гри, включаючи зміну винагород за співпрацю або зраду.
   - Дозволяє тестувати вплив різних параметрів на стратегії, таких як довжина гри, кількість агентів або рівень "пам'яті" нейронних мереж.
5. **Впровадження генетичних алгоритмів**:
   - Кожне покоління агентів формується на основі еволюційного підходу.
   - Використовується механізм штучного відбору: ефективні стратегії переходять до наступного покоління, тоді як менш ефективні замінюються.
   - Змінюваність параметрів забезпечує адаптацію стратегій до складних сценаріїв гри.
6. **Інтеграція динамічних стратегій**:
   - Використання зворотного поширення помилки для навчання нейронних мереж у реальному часі.
   - Поетапний перехід від статичних до динамічних стратегій, що дозволяє ефективно комбінувати переваги обох підходів.

Опис методу орагнізацію процесу навчання данамічних стратегій

Реалізація та аналіз програмного засобу

1. **Реалізація програмного забезпечення**:
   - Програмний комплекс розроблено з використанням TypeScript, Python та бібліотек TensorFlow/PyTorch для нейронних мереж.
   - Інтерфейс симуляційного середовища дозволяє налаштовувати параметри гри, візуалізувати процес взаємодії агентів та аналізувати результати.
   - Застосовано модульний підхід, який забезпечує розширюваність та можливість інтеграції нових функцій.

2. **Результати аналізу**:
   - Було проведено симуляції для різних сценаріїв, включаючи ігри між статичними та динамічними агентами, а також між агентами з різними параметрами нейронних мереж.
   - Агенти з нейронними мережами на основі моделей LSTM демонстрували здатність до адаптації та прогнозування дій супротивника

Висновки

1. **Досягнення мети дослідження**:
   Розроблено метод створення адаптивних стратегій для "дилеми в'язня" на основі нейронних мереж і генетичних алгоритмів.

2. **Виконання завдань**:
   - Створено архітектуру нейронної мережі для моделювання адаптивних стратегій.
   - Інтегровано генетичні алгоритми для оптимізації стратегій.
   - Проведено тестування у симуляційному середовищі, яке підтвердило ефективність підходу.

3. **Ключові результати**:

   Розроблено метод формування гнучких адаптивних стратегій дій у дилемі вязня.

4. **Практична значущість**:
   Результати можуть бути використані у моделюванні взаємодій у складних системах, таких як управління ресурсами чи соціальні конфлікти.

References

1. GRASP like algorithm. José Barahona da Fonseca // Computer Aided Chemical Engineering, Volume 24, 2017, Pages 279-284. https://doi.org/10.1016/S1570-7946(07)80070-8
2. Optimization of computer programming based on mathematical models of artificial intelligence algorithms. Yuhui Zheng //
3. Computers and Electrical Engineering. Volume 110, September 2023, 108834. https://doi.org/10.1016/j.compeleceng.2023.108834 Nikoleta E. Glynatsi, Vincent Knight and Marc Harper. Properties of Winning Iterated Prisoner's Dilemma Strategies, 2024.
4. Domajnko, Martin. Iterated prisoner's dilemma and survival of the fittest from an ecological perspective. Proceedings of the
8th Student Computing Research Symposium (SCORES'22) 2022.
5. Ferraz, V., Pitz, T. Analyzing the Impact of Strategic Behavior in an Evolutionary Learning Model Using a Genetic Algorithm. Comput Econ 63, 437–475 (2024). https://doi.org/10.1007/s10614-022-10348-1
6. Rui Dong, Xinghong Jia, Xianjia Wang, Yonggang Chen. Optimal Tag-Based Cooperation Control for the "Prisoner's Dilemma", 2020. https://doi.org/10.1155/2020/8498613
7. Montero-Porras, E., Grujić, J., Fernández Domingos, E. et al. Inferring strategies from observations in long iterated Prisoner's dilemma experiments. Sci Rep 12, 7589 (2022). https://doi.org/10.1038/s41598-022-11654-2
8. Lin, B., Bouneffouf, D., Cecchi, G. (2022). Online Learning in Iterated Prisoner's Dilemma to Mimic Human Behavior. In: Khanna, S., Cao, J., Bai, Q., Xu, G. (eds) PRICAI 2022: Trends in Artificial Intelligence. PRICAI 2022. Lecture Notes in Computer Science, vol 13631. Springer, Cham. https://doi.org/10.1007/978-3-031-20868-3_10
9. de Paulo, K.P., Estombelo-Montesco, C.A. & Tejada, J. New memory-one strategies of the Iterated Prisoner's Dilemma: a new framework to programmed human-AI interaction. Discov Psychol 4, 20 (2024). https://doi.org/10.1007/s44202-024-00133-6
10. Umberto Cerruti, Simone Dutto, Nadir Murru, A symbiosis between cellular automata and genetic algorithms, Chaos, Solitons & Fractals, Volume 134, 2020, 109719, ISSN 0960-0779, https://doi.org/10.1016/j.chaos.2020.109719.

Завідувачу кафедри інженерії програмного забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Ярослава БАДЬОРИ
факультет ІТ, 2 курс, група ІПЗм-23-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (StrikePlagiarism та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

02.09.2024.
дата


підпис

Завідувачу кафедри
інженерії програмного забезпечення
проф. Леоніду БЕДРАТЮКУ
студента групи ІПЗм-23-1
Ярослава БАДЬОРИ
Ім'я, ПРІЗВИЩЕ

ЗАЯВА

Прошу закріпити за мною тему кваліфікаційної роботи освітнього ступеня «магістр» за спеціальністю 121 «Інженерія програмного забезпечення»:
Метод розробки програмних застосунків для симуляції та створення стратегій «дилеми в'язня» на основі нейронних мереж.

(керівник кваліфікаційної роботи – Юрій ФОРКУН)
Ім'я, ПРІЗВИЩЕ

02.09.2024.
Дата

Підпис здобувача

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів подібності щодо роботи, продукованими програмно-технічним засобом(ами) перевірки текстів на плагіат.
Назва: «Метод розробки програмних застосунків для симуляції та створення стратегій «дилеми в'язня» на основі нейронних мереж.»
Автор: Бадьора Ярослав Михайлович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Форкун Юрій Вікторович кандидат технічних наук, доцент.
Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є плагіатом, розміщені у розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за два дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої й електронної версії роботи.	
3	Виявлені запозичення не є плагіатом, але частково розміщені у розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того, як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:
Запозичення, виявлені у роботі, є законними і не є плагіатом, оскільки:
1) у тексті кваліфікаційної роботи системою перевірки на плагіат StrikePlagiarism виявлено схожість з деякими документами у частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, бланк завдання), у структурі змісту, у назвах розділів/підрозділів, у назвах публікацій переліку джерел посилання тощо;
2) в якості запозичень системою StrikePlagiarism було зафіксовано деякі послідовності вихідного коду і посиланання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;
3) запозичення, виявлені в тексті роботи, є фрагментарними або мають належним чином оформленні посилання;
4) виявлені модифікації тексту не впливають на відсоток схожості.
Максимальний обсяг запозичень, визначений системою Anti-Plagiarism, складає 2% За системою StrikePlagiarism коефіцієнт подібності 1 складає 0,4, коефіцієнт подібності 2 складає 0 %

Дата 01.12.2024

Завідувач кафедри ІПЗ _____ Леонід БЕДРАТЮК
Гарант освітньої програми _____ Оксана ЯШИНА
Керівник кваліфікаційної роботи _____ Юрій ФОРКУН

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 2.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 11%**

	Документ		Сумарний збіг по Базі Даних	
ID: 153409 Назва: КП_Метод розробки програмних застосунків для симуляції та утворення стратегій "дилеми в`язня" на основі нейронних мереж Додано в БД: 2024-12-02 Автора: Ярослав БАДЬОРА Керівники: кандидат тех. наук, доцент Юрій ФОРКУН Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	109819	865	2759 (3%)	34 (4%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Ярослав БАДЬОРА

Співавтор:

Назва: Метод розробки програмних застосунків для симуляції та створення стратегій "дилеми в'язня" на основі нейронних мереж

Науковий керівник: кандидат тех. наук, доцент Юрій ФОРКУН

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:0.4%

Коефіцієнт подібності 2:0%

Мікропробіли: 72

Заміна букв: 39

Інтервали: 0

Білі знаки: 16

Дата створення звіту: 2024-12-03 01:00:42.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

2.12.2024
Дата

(Форкун Ю.В.)
експерт

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Магістр»

Дипломник Бадьора Ярослав Михайлович

Тема Метод розробки програмних застосунків для симуляції та створення стратегій «дилеми вязня» на основі нейронних мереж

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 0; кількість сторінок записки 115

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі було досліджено предметну область та проведено аналіз існуючих актуальни х рішень в сфері теорії ігор та дилеми в'язня. Визначено недосліджені частини у сфері, проаналізовано як покращити існуючі рішення. У рамках роботи було сформовано технічне завдання, розробленно метод створення динамічних стратегій вирішення дилеми в'язня, що дозволить приблизити стратегії поведінки агентів до реального світу, та дозволить використання їх в симуляціях реальних подій у сфері соцілогії.

2. Висновок про відповідність проекту поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу проекту, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі доведено актуальність теми, визначено мету та завдання кваліфікаційної роботи. У першому розділі проведено аналіз предметної області, розглянуто існуючі рішення та визначені функціональні і нефункціональні вимоги до розроблюваного веб-застосунку. У другому розділі проведено аналіз сучасних концепцій методів, розглянуто їх переваги і недоліки. У третьому розділі було детально описано проектування. У четвертому розділі підготовлено всі залежності для написання коду та виконано практичну розробку програмних модулів і описано їх особливості, було виконано тестування системи та проведено її у відповідності до функціональних вимог, в результаті чого було підтверджено доцільність використання методу.

4. Позитивні сторони проекту Тематика дипломного проекту є актуальною, що було підтверджено великою кількістю сучаних досліджень в даній сфері. Під час проектування було досліджено актуальні дослідження проаналізовано математичну частину та наведено приклад реалізації.

5. Негативні сторони проекту Відсутні

6. Оцінка графічного оформлення та пояснювальної записки проекту Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки. Викладення матеріалу пояснювальної записки є структурованим, послідовним та чітким, що дозволяє зрозуміти викладений матеріал у рамках тематики кваліфікаційного проекту. Графічний матеріал надає можливість наочно побачити деталі проектування методу.

8. Інші зауваження ____________________

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «відмінно».

РЕЦЕНЗЕНТ Лисенко Сергій Миколайович, д.т.н., професор, професор кафедри комп'ютерної інженерії та інформаційних систем, заступник декана факультету інформаційних технологій з наукової та інноваційної роботи.

"04" грудня 2024 р. __________ (підпис)

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ДЕКЛАРАЦІЯ УЧАСНИКА ОСВІТНЬОГО ПРОЦЕСУ

щодо дотримання академічної доброчесності

Цією декларацією я, Бардюра Ярослав Михайлович
Прізвище, ім'я, по батькові

здобувач вищої освіти (шифр та назва спец-ті, курс, академічна група)/ науково-педагогічний працівник (назва кафедри)

назва факультету

підтверджую, що ознайомився (- лась) з Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті та Кодексом академічної доброчесності і **зобов'язуюсь** дотримуватися їх вимог під час освітнього процесу, проведення наукової діяльності, виконання організаційно-адміністративних функцій тощо.

Усвідомлюю, що у разі порушення мною принципів академічної доброчесності нестиму відповідальність перед академічною спільнотою ХНУ згідно з нормами, визначеними Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, законодавства України.

«02» лютого 2023 р.

Підпис