

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

**Навчально-науковий інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу**

До захисту допущено:
Завідувач кафедри
_____ Оксана ТИМОЩУК
« ____ » _____ 2026 р.

Дипломна робота
на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системний аналіз і управління»
спеціальності 124 «Системний аналіз»
на тему: «Візуальне середовище для емпіричного аналізу ігор в
нормальній формі»

Виконала:

студентка IV курсу, групи КА-22
Сорокопуд Таїсія Сергіївна _____

Керівник:

Доцент кафедри ММСА, к.т.н., доц.
Куєвда Юлія Валеріївна _____

Консультант з економічного розділу:

Доцент кафедри ЕК, к.е.н.
Рощина Надія Василівна _____

Консультант з нормоконтролю:

Доцент кафедри ММСА, к.ф.-м.н.
Статкевич Віталій Михайлович _____

Рецензент:

Доцент кафедри ІІІ, к.ф.-м.н., доцент
Пишнограєв Іван Олександрович _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.

Студентка _____

Київ – 2026 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 124 «Системний аналіз»
Освітня програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Оксана ТИМОЩУК
« ____ » _____ 2026 р.

ЗАВДАННЯ
на дипломну роботу студентці
Сорокопуд Таїсії Сергіївні

1. Тема роботи «Візуальне середовище для емпіричного аналізу ігор в нормальній формі», керівник роботи Куєвда Юлія Валеріївна, доцент кафедри ММСА, кандидат технічних наук, доцент, затверджені наказом по університету від 27.05.2026 р. № 1944-с.
2. Термін подання студентом роботи «10» червня 2026 р.
3. Вихідні дані до роботи: табличні представлення функцій виграшів ігор у нормальній формі, зокрема біматриці двоосібних ігор; набір класичних, синтетичних і прикладних ігор для проведення обчислювальних експериментів; матеріали переддипломної практики.
4. Зміст роботи: дослідження предметної області емпіричного аналізу ігор у нормальній формі; розгляд математичного апарату теорії ігор, графового та спектрального аналізу; аналіз табличного представлення функцій виграшів і біматричних моделей двоосібних ігор; розроблення програмного середовища; аналіз результатів роботи системи на прикладах класичних, синтетичних і прикладних ігор.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): презентація.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Функціонально- вартісний аналіз	Рощина Н.В., доц., к.е.н.		

7. Дата видачі завдання 27.05.2026.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Формування тематики дослідження	15.02.2026– 01.03.2026	Виконано
2	Аналіз актуальності задачі дослідження	02.03.2026– 20.03.2026	Виконано
3	Аналіз відомих підходів за темою роботи	21.03.2026– 13.04.2026	Виконано
4	Формування теми, мети, об'єкта, предмета та задач дипломної роботи	14.04.2026– 19.04.2026	Виконано
5	Ознайомлення з технологіями для реалізації програмного середовища	20.04.2026– 26.04.2026	Виконано
6	Розробка програмного середовища для аналізу двоосібних ігор у нормальній формі	27.04.2026– 09.05.2026	Виконано
7	Написання звіту з переддипломної практики та підготовка матеріалів для теоретичної частини дипломної роботи	04.05.2026– 17.05.2026	Виконано
8	Оформлення розділів дипломної роботи	18.05.2026– 24.05.2026	Виконано
9	Попередній захист дипломної роботи	06.06.2026– 08.06.2026	Виконано
10	Захист дипломної роботи	15.06.2026	Виконано

Студентка _____

Таїсія СОРОКОПУД

Керівник _____

Юлія КУЄВДА

РЕФЕРАТ

Дипломна робота: 177 с., 28 рис., 9 табл., 2 дод., 15 джерел.

ТЕОРІЯ ІГОР, ЕМПІРИЧНИЙ АНАЛІЗ ІГОР, ЛАНДШАФТ ІГОР.

Об'єкт дослідження – процеси емпіричного аналізу та порівняння структурних характеристик ігор у нормальній формі.

Предмет дослідження – методи графового, спектрального та візуального аналізу ігор у нормальній формі.

Мета роботи – дослідити та реалізувати аналітико-візуалізаційний етап емпіричного аналізу ігор у нормальній формі, що передбачає роботу з уже сформованими таблицями виграшів.

Методи дослідження – методи теорії ігор, графові методи аналізу стратегічних переходів, метод головних компонент та обчислювальні експерименти.

Актуальність роботи – зумовлена необхідністю розроблення засобів для наочного аналізу, класифікації та порівняння ігор у нормальній формі. Такий підхід є важливим для подальшого створення нових ігрових структур із заданими властивостями, які можуть використовуватися як тестові середовища для дослідження стратегічної поведінки та навчання агентів.

Результати роботи – розроблено візуальне середовище для аналізу ігор у нормальній формі та проведено обчислювальні експерименти на прикладах класичних, синтетичних і прикладних ігор.

Шляхи подальшого розвитку предмету дослідження – автоматизація формування таблиць виграшів на основі симуляцій або даних, використання ландшафту ігор як основи для процедурного створення нових ігрових структур із заданими структурними характеристиками.

ABSTRACT

Diploma thesis: 177 p., 28 fig., 9 tables, 2 appendices, 15 references.

GAME THEORY, EMPIRICAL GAME ANALYSIS, LANDSCAPE OF GAMES.

Object of research – processes of empirical analysis, visualization and comparison of structural characteristics of games in normal form.

Subject of research – methods of graph-based, spectral and visual analysis of games in normal form.

Purpose of the study – to investigate and implement the analytical and visualization stage of empirical analysis of games in normal form, which involves working with already constructed payoff tables.

Research methods – methods of game theory, graph-based methods for analysing strategic transitions, principal component analysis and computational experiments.

Relevance of the study – determined by the need to develop tools for visual analysis, classification and comparison of games in normal form. This approach is important for the further creation of new game structures with specified properties, which can be used as test environments for studying strategic behaviour and agent learning.

Results of the study – a visual environment for the analysis of games in normal form was developed, and computational experiments were conducted on examples of classical, synthetic and applied games.

Ways to further develop the subject of research – automation of payoff table construction based on simulations or data, and the use of the landscape of games as a basis for the procedural creation of new game structures with specified structural characteristics.

ЗМІСТ

ВСТУП	8
РОЗДІЛ 1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ ЗАДАЧІ	10
1.1 Актуальність аналізу стратегічних і багатоагентних взаємодій	10
1.2 Проблема аналізу ігор у нормальній формі	11
1.3 Емпіричний аналіз ігор та місце розробленої системи	13
1.4 Ландшафт ігор як інструмент порівняння та класифікації.....	15
1.5 Перспектива створення нових ігрових структур	16
1.6 Постановка задачі дипломної роботи	18
1.7 Висновки до розділу 1	19
РОЗДІЛ 2 МАТЕМАТИЧНИЙ ТА МЕТОДИЧНИЙ АПАРАТ АНАЛІЗУ ІГОР У НОРМАЛЬНІЙ ФОРМІ	21
2.1 Ігри в нормальній формі як вхідна модель аналізу	21
2.2 Найкращі відповіді, домінування стратегій та рівновага Неша.....	23
2.3 Емпіричний аналіз ігор у нормальній формі.....	26
2.4 Граф відповідей і структурні характеристики гри	28
2.5 α -Rank та ранжування стратегічних профілів.....	31
2.6 Спектральний аналіз графів і кластеризація стратегічних профілів	34
2.7 Методи зниження розмірності та побудова ландшафту ігор	37
2.8 Висновки до розділу 2	44
РОЗДІЛ 3 РОЗРОБЛЕННЯ ПРОГРАМНОГО СЕРЕДОВИЩА ТА АНАЛІЗ РЕЗУЛЬТАТІВ.....	47
3.1 Загальна характеристика розробленого програмного середовища	47
3.2 Архітектура програмної реалізації.....	52
3.3 Набір ігор для проведення обчислювальних експериментів.....	58
3.4 Аналіз окремої гри у програмному середовищі	64
3.5 Побудова ландшафту ігор за структурними характеристиками	78
3.6 Висновки до розділу 3	91

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ	93
4.1 Постановка задачі проектування.....	94
4.2 Обґрунтування функцій програмного продукту.....	95
4.3 Обґрунтування системи параметрів програмного продукту	102
4.4 Аналіз експертного оцінювання параметрів	105
4.5 Аналіз рівня якості варіантів реалізації функцій.....	109
4.6 Економічний аналіз варіантів розробки програмного продукту	112
4.7 Вибір оптимального варіанта реалізації програмного продукту	119
4.8 Висновки до розділу 4	120
ВИСНОВКИ.....	121
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ	122
ДОДАТОК А. ТЕКСТ ПРОГРАМИ	124
ДОДАТОК Б. ІЛЮСТРАТИВНИЙ МАТЕРІАЛ.....	169

ВСТУП

Сучасні дослідження стратегічних і багатоагентних взаємодій усе частіше потребують не лише аналізу окремих ігор, а й засобів для порівняння різних ігрових середовищ між собою. Ігри використовуються як тестові моделі для дослідження поведінки агентів, алгоритмів прийняття рішень і методів навчання, однак традиційний підхід, орієнтований переважно на пошук сильних стратегій або окремих рівноважних станів, не завжди дає змогу зрозуміти внутрішню структуру самої гри. У зв'язку з цим актуальним стає дослідження структурних властивостей ігор, характеру стратегічних переходів, циклічності, подібності між іграми та можливості їх класифікації за структурними ознаками.

Аналіз ландшафту ігор дозволяє перейти від розгляду окремої гри до вивчення простору ігор як сукупності об'єктів із певними структурними характеристиками [1]. Такий підхід дає змогу кількісно порівнювати ігри різного розміру та складності, виявляти групи структурно подібних ігор, відокремлювати транзитивні, циклічні та складніші змішані структури. Особливо важливим є те, що отримана інформація може використовуватися не лише для аналізу вже наявних ігор, а й для подальшого створення нових ігрових структур із заданими властивостями. Це відкриває перспективу формування тестових середовищ для дослідження стратегічної поведінки агентів і розвитку методів багатоагентного навчання.

Актуальність дипломної роботи зумовлена необхідністю розроблення програмних засобів, які дозволяють наочно досліджувати, класифікувати та порівнювати ігри у нормальній формі. У межах роботи основну увагу зосереджено на аналітико-візуалізаційному етапі емпіричного аналізу ігор, тобто на роботі з уже сформованими таблицями виграшів. Для таких ігор доцільним є автоматизоване отримання графових і числових характеристик,

побудова графів відповідей, аналіз циклів, кластеризація стратегічних профілів і формування ландшафту ігор, який відображає подібність між різними стратегічними середовищами.

Метою дипломної роботи є дослідження та реалізація аналітико-візуалізаційного етапу емпіричного аналізу ігор у нормальній формі, що передбачає роботу з уже сформованими таблицями виграшів, побудову графів відповідей, визначення структурних характеристик і порівняння ігор у вигляді ландшафту. Об'єктом дослідження є ігри в нормальній формі як математичні моделі стратегічної взаємодії між гравцями. Предметом дослідження є методи графового, спектрального та візуального аналізу ігор у нормальній формі.

У роботі використано методи теорії ігор, аналіз найкращих відповідей і рівноваг Неша, графові методи аналізу стратегічних переходів, спектральний аналіз графів, кластеризацію, метод головних компонент, методи візуалізації даних та обчислювальні експерименти. Практична цінність роботи полягає у створенні візуального середовища, яке дозволяє аналізувати ігри у нормальній формі, досліджувати їхні структурні характеристики та порівнювати різні ігри між собою. Отримані результати можуть бути використані як основа для подальшого розширення системи, зокрема для автоматизованого формування нових ігрових структур із наперед заданими структурними властивостями.

РОЗДІЛ 1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ ЗАДАЧІ

1.1 Актуальність аналізу стратегічних і багатоагентних взаємодій

Сучасні задачі дослідження стратегічної поведінки дедалі частіше пов'язані з аналізом систем, у яких результат залежить від взаємодії кількох учасників. Такі системи виникають в економіці, управлінні, соціальному моделюванні, кібербезпеці, багатоагентних середовищах та задачах навчання агентів. Ігрові моделі також використовуються в задачах прийняття рішень у складних системах, зокрема для формалізації взаємодії учасників, вибору стратегій та аналізу наслідків прийнятих рішень [2]. У цих випадках важливим є не лише визначення окремого оптимального рішення, а й дослідження того, як різні стратегії співвідносяться між собою, які форми поведінки є стійкими, які взаємодії мають циклічний характер і як одна стратегічна ситуація відрізняється від іншої.

Ігрові моделі посідають важливе місце в таких дослідженнях, оскільки дозволяють формалізувати взаємодію між учасниками та аналізувати наслідки вибору різних стратегій. Особливої актуальності ця задача набуває в контексті багатоагентних систем, де ігри можуть використовуватися як тестові середовища для оцінювання алгоритмів прийняття рішень і навчання агентів. При цьому важливим стає не лише досягнення високої якості гри агентами в окремих середовищах, а й розуміння структури самих ігор, їхніх структурних властивостей і взаємозв'язків [1].

Традиційний підхід до аналізу ігор часто зосереджується на окремій грі та її результатах. Проте для складніших стратегічних середовищ цього недостатньо, оскільки різні ігри можуть мати подібні або, навпаки, суттєво відмінні структурні властивості. Дві гри можуть бути близькими за розміром

таблиці виграшів, але відрізнятися характером стратегічних переходів, наявністю циклів, кількістю стійких станів або складністю взаємодії між стратегіями. Тому виникає потреба у методах, які дозволяють аналізувати не тільки числові значення виграшів, а й загальну структуру гри.

Одним із перспективних напрямів такого аналізу є побудова графових представлень і подальше порівняння ігор у спільному просторі структурних характеристик. У цьому підході гра розглядається як об'єкт, для якого можна обчислити набір ознак, що відображають характер стратегічної взаємодії. Такі ознаки можуть бути використані для класифікації ігор, виявлення подібних структур, розмежування транзитивних і циклічних взаємодій, а також для визначення того, які ігри можуть бути корисними як тестові середовища для агентів.

У межах цієї дипломної роботи зазначена проблема розглядається через розроблення візуального середовища для аналізу ігор у нормальній формі. Основна увага приділяється аналітико-візуалізаційному етапу роботи з уже сформованими таблицями виграшів: побудові графів відповідей, визначенню структурних характеристик, виявленню циклів і кластерів, а також порівнянню ігор у спільному просторі ознак. Такий підхід дозволяє підвищити наочність аналізу стратегічних взаємодій і створює основу для подальшого розширення системи.

1.2 Проблема аналізу ігор у нормальній формі

Ігри в нормальній формі є одним із поширених способів подання стратегічних взаємодій, оскільки дозволяють задати результати для всіх можливих комбінацій стратегій гравців. Таке подання є зручним для подальшої обчислювальної обробки, оскільки таблиця виграшів може бути

безпосередньо використана для пошуку рівноважних станів, визначення найкращих відповідей та порівняння результатів різних стратегічних профілів. Водночас із погляду дослідницького аналізу табличне представлення виграшів не завжди є достатнім для повної інтерпретації гри.

Основна проблема полягає в тому, що табличне подання фіксує числові значення виграшів, але не завжди наочно відображає структуру взаємодії між стратегіями. Зі збільшенням кількості стратегій зростає кількість стратегічних профілів, ускладнюється ручний перегляд таблиці виграшів, а виявлення закономірностей у поведінці гравців стає менш очевидним. У такій ситуації досліднику важливо не лише бачити окремі значення виграшів, а й розуміти, які переходи між профілями є можливими, які стани є стійкими, чи існують циклічні залежності та які частини гри мають подібну структурну роль.

Класичний аналіз ігор часто орієнтований на пошук рівноважних станів, однак для складніших стратегічних середовищ цього недостатньо. Дві гри можуть мати однакову кількість стратегій і подібне табличне представлення виграшів, але суттєво відрізнятися за внутрішньою структурою. Одна гра може мати виражену транзитивну структуру, інша – циклічну, третя – поєднувати кілька типів стратегічних залежностей. Тому виникає потреба в таких методах аналізу, які дозволяють досліджувати гру не лише через окремі рівноваги або значення виграшів, а через сукупність її структурних характеристик.

Перехід від аналізу табличного представлення функцій виграшів до дослідження структурних властивостей гри дає змогу розглядати гру як цілісний об'єкт із системою стратегічних переходів і взаємозв'язків між профілями. У такому підході графові представлення використовуються для виявлення властивостей, які не завжди помітні у початкових числових даних: циклічних залежностей, домінуючих напрямів переходів, груп подібних станів, складності стратегічної взаємодії та загальної структури гри [1].

Особливо важливою є проблема порівняння різних ігор між собою. Якщо кожна гра розглядається окремо, важко визначити, наскільки вона подібна до інших ігор або чим саме від них відрізняється. Для цього необхідно перейти до формування єдиного простору ознак, у якому кожна гра описується набором кількісних характеристик. Такий підхід дає можливість порівнювати ігри різного розміру, виявляти групи структурно близьких ігор та будувати їхній ландшафт.

1.3 Емпіричний аналіз ігор та місце розробленої системи

Емпіричний теоретико-ігровий аналіз є важливим напрямом дослідження стратегічних взаємодій у складних системах, де повне аналітичне задання гри може бути ускладненим або недоцільним. На відміну від суто теоретичного підходу, у якому функції виграшу часто задаються формально, емпіричний підхід орієнтований на роботу з конкретними результатами взаємодії стратегій. Такі результати можуть бути отримані на основі симуляцій, експериментів, багаторазових запусків агентів або вже підготовлених наборів даних. У підсумку формується емпірична гра, яка може бути подана через табличне представлення функцій виграшів гравців і використана для подальшого аналізу.

Повний процес емпіричного теоретико-ігрового аналізу охоплює декілька етапів. До нього належать визначення учасників взаємодії, формування множин стратегій, отримання або оцінювання виграшів для стратегічних профілів, побудова емпіричної гри та її подальше дослідження. Саме тому важливо розмежовувати побудову емпіричної гри та аналіз уже сформованої гри. Перша частина пов'язана з отриманням даних і побудовою табличного представлення функцій виграшів гравців, тоді як друга – з

інтерпретацією цього представлення, виявленням структурних властивостей і порівнянням гри з іншими стратегічними середовищами. У межах цієї дипломної роботи основну увагу зосереджено саме на аналітико-візуалізаційному етапі емпіричного аналізу. Розроблена система не виконує повний цикл формування емпіричної гри з початкових даних або симуляцій, а працює з уже заданим або попередньо сформованим табличним представленням функцій виграшів гравців. Такий підхід дозволяє чітко визначити межі роботи та зосередитися на задачі структурного аналізу гри: побудові графа відповідей, визначенні рівноважних і нестійких профілів, аналізі циклів, обчисленні графових характеристик, кластеризації стратегічних профілів і порівнянні ігор у спільному просторі ознак.

Таке розмежування є принциповим для коректного використання поняття емпіричного аналізу ігор. У літературі цей термін часто охоплює весь процес – від вибору стратегій і отримання виграшів до інтерпретації результатів. У цій роботі реалізується не весь цей фреймворк, а його окрема складова, пов'язана з дослідженням уже сформованої гри. Отже, вхідними даними програмного середовища є табличне представлення функцій виграшів гравців, а результатами роботи – набір числових і візуальних представлень, які допомагають інтерпретувати структуру стратегічної взаємодії.

Важливість такого етапу зумовлена тим, що саме табличне представлення виграшів не завжди дає достатнє уявлення про характер гри. Для дослідника важливо не лише знати значення виграшів, а й розуміти, які стратегічні профілі є стійкими, які переходи можливі між ними, чи існують циклічні структури, які групи профілів мають подібну роль та як ця гра співвідноситься з іншими іграми. Саме тому аналітико-візуалізаційний етап є необхідною складовою емпіричного дослідження, оскільки він перетворює табличне представлення виграшів на більш інформативне структурне представлення стратегічної взаємодії.

У контексті цієї роботи особливе значення має можливість порівняння ігор між собою. Якщо для кожної гри обчислити набір структурних характеристик, то надалі ці ігри можна розглядати як об'єкти у спільному просторі ознак. Це створює основу для побудови ландшафту ігор, де близьке розташування точок свідчить про подібність стратегічної структури, а віддаленість – про суттєві відмінності. Такий підхід узгоджується з сучасними дослідженнями, у яких графові характеристики використовуються для аналізу структурних властивостей ігор, їх класифікації та подальшого створення нових ігрових структур із заданими властивостями [1].

Таким чином, розроблена система займає місце між етапом побудови емпіричної гри та етапом прийняття дослідницьких висновків щодо її структури. Вона не замінює засоби збору даних або симуляцій, але доповнює їх, забезпечуючи інструменти для аналізу вже отриманого табличного представлення функцій виграшів гравців. Це дозволяє використовувати систему як окремий програмний інструмент для дослідження ігор у нормальній формі та як складову ширшого процесу емпіричного теоретико-ігрового аналізу.

1.4 Ландшафт ігор як інструмент порівняння та класифікації

Для переходу від аналізу окремої гри до дослідження множини ігор необхідно мати спосіб їхнього кількісного та візуального порівняння. Якщо кожна гра розглядається лише через табличне представлення виграшів, то складно визначити, наскільки вона подібна до інших ігор, які структурні властивості є для неї характерними та до якого типу стратегічних середовищ її можна віднести. Саме тому доцільним є підхід, у якому гра описується не тільки через значення виграшів, а через набір структурних характеристик.

Ландшафт ігор можна розглядати як спільний простір, у якому кожна гра подана окремою точкою, а її положення визначається сукупністю обчислених ознак. До таких ознак можуть належати характеристики графа відповідей, зокрема кількість стратегічних переходів, наявність циклів, показники зв'язності, розподіл вхідних і вихідних степенів, кількість стійких профілів та інші величини, що описують структурні властивості гри.

Такий підхід дозволяє порівнювати ігри різного розміру та різної природи. Ігри зі схожою структурою стратегічних взаємодій розташовуються близько одна до одної, тоді як ігри з відмінними структурними властивостями займають віддалені області ландшафту. Це дає змогу відокремлювати транзитивні, циклічні та змішані структури, виявляти групи споріднених ігор, а також визначати нетипові приклади, які суттєво відрізняються від інших за характером стратегічних переходів.

У межах цієї роботи ландшафт ігор використовується як інструмент порівняння класичних, синтетичних і прикладних ігор у нормальній формі. Його побудова дозволяє не лише проаналізувати кожну гру окремо, а й визначити її місце серед інших ігор за структурними характеристиками. Це підвищує наочність дослідження та створює основу для подальшого використання отриманого простору ознак у задачах класифікації, порівняння й інтерпретації стратегічних середовищ.

1.5 Перспектива створення нових ігрових структур

Одним із важливих напрямів розвитку аналізу ігор є перехід від опису вже наявних ігрових моделей до створення нових ігрових структур із наперед визначеними властивостями. У сучасних дослідженнях багатоагентних систем це має особливе значення, оскільки ігри можуть використовуватися як тестові

середовища для перевірки алгоритмів прийняття рішень, навчання агентів та аналізу стратегічної поведінки. Для таких задач важливо мати не лише набір відомих ігор, а й можливість цілеспрямовано формувати нові середовища, які відрізняються рівнем складності, характером стратегічних переходів, наявністю циклів або ступенем транзитивності.

Ландшафт ігор має не лише описове, а й конструктивне значення: після представлення гри через набір кількісних характеристик можна перейти до зворотної задачі – формування таких ігор, які займали б певну область простору ознак або мали б задані структурні властивості [1]. Це відкриває можливість створення нових ігрових середовищ, орієнтованих на дослідження конкретних типів стратегічної взаємодії.

Такий підхід є перспективним, оскільки різні ігри можуть бути корисними для різних дослідницьких цілей. Наприклад, ігри з вираженою транзитивною структурою можуть використовуватися для перевірки здатності алгоритму знаходити сильніші стратегії, тоді як циклічні або змішані структури є важливими для дослідження ситуацій, у яких не існує простого лінійного впорядкування стратегій. У цьому випадку ландшафт ігор може виступати не лише засобом візуалізації, а й інструментом орієнтації в просторі можливих ігрових структур.

У контексті цієї дипломної роботи безпосереднє процедурне створення нових ігор не є основною задачею реалізації. Проте розроблене візуальне середовище формує необхідну основу для такого напрямку подальших досліджень. Система дозволяє працювати з уже сформованими табличними представленнями функцій виграшів, будувати графи відповідей, обчислювати структурні характеристики та розміщувати ігри у спільному просторі ознак. Саме ці результати можуть бути використані надалі для визначення того, які властивості мають різні групи ігор і які параметри потрібно змінювати для отримання нових структур.

Отже, у межах цієї дипломної роботи створення нових ігор розглядається як перспективний напрям подальшого розвитку, а не як безпосередня функція реалізованої системи. Основний результат роботи полягає у формуванні інструментів для аналізу, візуалізації та порівняння вже заданих ігор, що надалі може бути використано для побудови нових ігрових структур із заданими структурними властивостями.

1.6 Постановка задачі дипломної роботи

З огляду на розглянуті особливості аналізу ігор у нормальній формі, у дипломній роботі ставиться задача розроблення програмного середовища, яке забезпечує аналітико-візуалізаційний етап дослідження стратегічних взаємодій. Основна увага приділяється роботі з уже сформованими табличними представленнями функцій виграшів, які можуть відповідати класичним, синтетичним або прикладним іграм. Такий підхід дозволяє зосередитися не на процесі отримання виграшів із симуляцій або даних, а на подальшому структурному аналізі гри та інтерпретації її властивостей.

Поставлена задача полягає у створенні середовища, яке дає змогу користувачеві обирати або задавати гру в нормальній формі, переглядати табличне представлення функцій виграшів, знаходити рівноважні стани, будувати граф відповідей і досліджувати систему стратегічних переходів між профілями. Важливо, щоб програмний продукт не обмежувався виведенням окремих числових результатів, а забезпечував наочне подання структури гри, зокрема дозволяв аналізувати цикли, зв'язність, степені вершин, кластери стратегічних профілів та інші характеристики, що описують структурну організацію гри.

Окремою складовою задачі є побудова ландшафту ігор. Для цього необхідно сформуванати набір структурних ознак для кожної гри, перетворити їх у спільний простір характеристик і використати методи зниження розмірності для візуального подання результатів. У такому ландшафті кожна гра має бути представлена як окрема точка, а її розташування має відображати подібність або відмінність від інших ігор за обчисленими характеристиками. Це дозволяє перейти від аналізу однієї гри до порівняльного дослідження множини стратегічних середовищ.

У межах дипломної роботи передбачається реалізація засобів для аналізу класичних ігор, синтетично згенерованих прикладів та прикладних моделей, зокрема локаційних ігор. Таке поєднання дозволяє перевірити роботу програмного середовища на різних типах стратегічних структур і показати, як графові та візуальні методи допомагають інтерпретувати результати. Особлива увага приділяється тому, щоб отримані характеристики могли бути використані не лише для опису окремих ігор, а й для їх порівняння в межах ландшафту.

1.7 Висновки до розділу 1

У першому розділі було розглянуто предметну область дипломної роботи та обґрунтовано актуальність аналізу ігор у нормальній формі. Показано, що для сучасних досліджень стратегічних і багатоагентних взаємодій важливим є не лише аналіз окремих рівноважних станів, а й дослідження структури гри, її циклів, взаємозв'язків між профілями та подібності до інших ігор.

Визначено, що розроблена система реалізує аналітико-візуалізаційний етап емпіричного аналізу, тобто працює з уже сформованими табличними

представленнями функцій виграшів. Окремо обґрунтовано значення ландшафту ігор як інструмента порівняння, класифікації та подальшого створення нових ігрових структур із заданими структурними властивостями.

Таким чином, у розділі сформовано загальне обґрунтування задачі дипломної роботи та визначено необхідність розроблення програмного середовища для графового, структурного й візуального аналізу ігор у нормальній формі.

РОЗДІЛ 2 МАТЕМАТИЧНИЙ ТА МЕТОДИЧНИЙ АПАРАТ АНАЛІЗУ ІГОР У НОРМАЛЬНІЙ ФОРМІ

У цьому розділі розглянуто методи, необхідні для розв’язання поставленої задачі та розроблення візуального середовища для аналізу ігор у нормальній формі. Основну увагу приділено математичному подання гри, аналізу найкращих відповідей і рівноваг Неша, побудові графа відповідей, а також методам структурного та візуального дослідження стратегічних профілів.

Розглянуті методи формують основу для подальшої програмної реалізації. Табличне представлення функцій виграшів використовується як вхідне подання гри, граф відповідей – як інструмент опису стратегічних переходів, а спектральний аналіз, кластеризація та методи зниження розмірності – як засоби узагальнення й візуалізації структури гри. Окрему увагу приділено ландшафту ігор, який дозволяє порівнювати різні ігри між собою за набором структурних характеристик.

2.1 Ігри в нормальній формі як вхідна модель аналізу

У межах цієї роботи базовим об’єктом дослідження є скінченна гра в нормальній формі. Таке подання є зручним для програмної реалізації, оскільки всі можливі результати стратегічної взаємодії задаються у вигляді скінченного набору значень виграшів. У класичній теорії ігор нормальна форма використовується для опису ситуацій, у яких задано множини гравців, їхні можливі стратегії та функції виграшу для кожного стратегічного профілю [3, 4]. У межах розробленого середовища така форма подання використовується

як вхідна структура даних для подальшого обчислення рівноваг, побудови графа відповідей, визначення структурних характеристик і формування ландшафту ігор.

Формально скінченну гру в нормальній формі можна подати як:

$$G = \langle N, \{S_i\}_{i \in N}, \{u_i\}_{i \in N} \rangle,$$

де N – множина гравців, S_i – множина стратегій i -го гравця, u_i – функція виграшу i -го гравця. Стратегічний профіль визначається як набір стратегій усіх гравців:

$$s = (s_1, s_2, \dots, s_n), s_i \in S_i.$$

Для кожного такого профілю функція виграшу задає результат гри для відповідного гравця:

$$u_i(s_1, s_2, \dots, s_n).$$

У практичній частині роботи основна увага приділяється двоосібним іграм у нормальній формі. Для двох гравців множини стратегій можна позначити як

$$S_1 = \{s_1^1, s_1^2, \dots, s_1^m\},$$

$$S_2 = \{s_2^1, s_2^2, \dots, s_2^n\}.$$

Тоді кожна пара стратегій (s_1^i, s_2^j) утворює стратегічний профіль. Для кожного такого профілю визначається результат гри у вигляді пари виграшів:

$$(u_1(s_1^i, s_2^j), u_2(s_1^i, s_2^j)).$$

Для зберігання двоосібної гри використовується біматричне подання, у якому виграші першого та другого гравців задаються двома матрицями однакового розміру [4]:

$$A = (a_{ij}), B = (b_{ij}),$$

де a_{ij} – виграш першого гравця при виборі i -ї стратегії першим гравцем і j -ї стратегії другим гравцем, а b_{ij} – відповідний виграш другого гравця.

Відповідно, кожна клітинка матриці виграшів має вигляд:

$$(a_{ij}, b_{ij}).$$

У розробленому програмному середовищі саме матриці A та B є основною вхідною формою подання гри. На їх основі виконуються всі подальші етапи аналізу: визначення найкращих відповідей, пошук чистих рівноваг Неша, побудова графа відповідей, обчислення структурних характеристик, спектральне подання, кластеризація стратегічних профілів і порівняння ігор у межах ландшафту.

Важливо, що гра в нормальній формі у цій роботі розглядається не лише як таблиця числових виграшів, а як початкова модель для побудови структурного представлення стратегічної взаємодії. Кожен стратегічний профіль надалі може бути інтерпретований як вершина графа, а зміна стратегії одним із гравців за умови покращення його виграшу – як напрямлений перехід між вершинами. Таким чином, табличне представлення функцій виграшів є вихідною точкою для графового, спектрального та візуального аналізу, що реалізується в програмному середовищі.

2.2 Найкращі відповіді, домінування стратегій та рівновага Неша

Після задання гри в нормальній формі необхідно визначити, які стратегічні профілі можуть вважатися стійкими з погляду односторонніх відхилень гравців. Для цього використовуються поняття найкращої відповіді, домінування стратегій та рівноваги Неша. Ці поняття є базовими для аналізу раціональної поведінки гравців і подальшої побудови графа стратегічних переходів.

Найкращою відповіддю гравця називається така стратегія, яка забезпечує йому максимальний виграш за фіксованих стратегій інших гравців. Для i -го гравця множину найкращих відповідей можна записати у вигляді:

$$BR_i(s_{-i}) = \{s_i \in S_i | u_i(s_i, s_{-i}) \geq u_i(s'_i, s_{-i}), \forall s'_i \in S_i\},$$

де s_{-i} означає набір стратегій усіх гравців, крім i -го, S_i – множина стратегій i -го гравця, а u_i – його функція виграшу. Отже, стратегія належить до множини найкращих відповідей, якщо жодна інша доступна стратегія гравця не забезпечує більшого виграшу за тих самих стратегій інших учасників [3, 4].

Для двоосібної гри у нормальній формі найкращі відповіді визначаються безпосередньо за матрицями виграшів. Найкраща відповідь першого гравця знаходиться шляхом порівняння значень у відповідному стовпці його матриці виграшів, оскільки стратегія другого гравця фіксована. Найкраща відповідь другого гравця, навпаки, визначається шляхом порівняння значень у відповідному рядку його матриці виграшів. Саме така процедура використовується для автоматичного визначення найкращих відповідей у програмному середовищі.

Окремим випадком аналізу стратегій є домінування. Стратегія називається домінуючою, якщо вона забезпечує гравцеві не менший виграш, ніж інші його стратегії, незалежно від вибору суперника. Якщо виграш є строго більшим для всіх можливих дій іншого гравця, стратегія називається строго домінуючою. Відповідно, домінована стратегія є такою, що поступається іншій стратегії за всіма або частиною можливих відповідей суперника. Аналіз домінування дозволяє виявляти стратегії, які мають очевидну перевагу або, навпаки, є нераціональними для вибору [3, 4].

Центральним поняттям некооперативної теорії ігор є рівновага Неша. Стратегічний профіль називається рівновагою Неша, якщо жоден гравець не може збільшити свій виграш шляхом односторонньої зміни власної стратегії за умови, що стратегії інших гравців залишаються незмінними [5, 4]. Формально стратегічний профіль

$$s^* = (s_1^*, s_2^*, \dots, s_n^*)$$

є рівновагою Неша, якщо для кожного гравця i виконується умова:

$$u_i(s_i^*, s_{-i}^*) \geq u_i(s_i, s_{-i}^*), \forall s_i \in S_i.$$

Це означає, що стратегія кожного гравця в рівноважному профілі є найкращою відповіддю на стратегії інших гравців. Для двоосібної гри умови рівноваги Неша можна записати так:

$$\begin{aligned} u_1(s_1^*, s_2^*) &\geq u_1(s_1, s_2^*), \forall s_1 \in S_1, \\ u_2(s_1^*, s_2^*) &\geq u_2(s_2, s_1^*), \forall s_2 \in S_2. \end{aligned}$$

У межах цієї роботи основна увага приділяється чистим рівновагам Неша. Чиста рівновага відповідає такому стратегічному профілю, у якому кожен гравець обирає одну конкретну стратегію без використання змішаного або ймовірнісного вибору. У біматричній грі така рівновага відповідає клітинці матриць виграшів, у якій стратегія кожного гравця є найкращою відповіддю на стратегію іншого гравця [3, 4].

У програмному середовищі пошук чистих рівноваг Неша виконується шляхом перевірки всіх стратегічних профілів гри. Для кожного профілю визначається, чи може перший або другий гравець односторонньо змінити стратегію та отримати більший виграш. Якщо жоден із гравців не має вигідного одностороннього відхилення, відповідний профіль фіксується як чиста рівновага Неша.

Важливо, що не кожна гра має рівновагу Неша в чистих стратегіях. Зокрема, у циклічних іграх можуть виникати ситуації, коли для кожного стратегічного профілю існує вигідне одностороннє відхилення. У таких випадках аналіз лише рівноважних станів не дає повної інформації про структуру гри. Саме тому в цій роботі рівноваги Неша розглядаються разом із графом відповідей, циклами, структурними характеристиками та іншими засобами аналізу стратегічних переходів.

2.3 Емпіричний аналіз ігор у нормальній формі

Емпіричний теоретико-ігровий аналіз використовується у випадках, коли стратегічну взаємодію доцільно досліджувати на основі конкретного скінченного набору стратегій і відповідних значень виграшів. На відміну від повністю аналітичного опису гри, де функції виграшу задаються формально, емпіричний підхід орієнтований на побудову моделі за результатами взаємодії обраних стратегій або політик. Такі результати можуть бути отримані шляхом симуляцій, експериментів, багаторазових запусків агентів або попередньої обробки даних [6, 7].

У загальному випадку емпірична гра є скінченим наближенням складнішої стратегічної системи. Якщо початкове середовище має великий простір можливих дій, то для аналізу обирається обмежена множина стратегій, яка вважається репрезентативною для досліджуваної взаємодії. Після цього для кожної комбінації стратегій визначаються відповідні значення виграшів, на основі яких формується емпірична таблиця виграшів. Саме це табличне представлення надалі використовується для застосування методів теорії ігор та обчислювального аналізу.

У контексті емпіричного аналізу важливим є не лише табличне подання, а й спосіб формування значень виграшів. Якщо у теоретично заданій грі виграші визначаються безпосередньо через аналітичні функції, то в емпіричній грі вони можуть бути результатом спостережень, обчислювальних експериментів або моделювання поведінки агентів. Через це емпірична таблиця виграшів відображає не всю можливу стратегічну систему, а її скінченне представлення, побудоване на основі доступних або обраних стратегій.

У межах цієї дипломної роботи емпірична гра розглядається на етапі, коли таблиця виграшів уже сформована. Процес отримання виграшів із

симуляцій, експериментів або зовнішніх наборів даних не є основною задачею розробленої системи. Вхідними даними програмного середовища є вже задані таблиці виграшів, а подальший аналіз спрямований на їхню інтерпретацію, побудову графа відповідей, визначення структурних характеристик і порівняння гри з іншими іграми.

Такий підхід відповідає аналітико-візуалізаційному етапу емпіричного теоретико-ігрового аналізу. Після формування емпіричної гри виникає задача дослідити її властивості: визначити найкращі відповіді, знайти рівноважні профілі, виявити можливі стратегічні переходи, цикли, зв'язність та інші структурні ознаки. У цьому випадку табличне представлення функцій виграшів виступає не кінцевим результатом, а початковою структурою для подальшого графового, спектрального та візуального аналізу.

Важливо враховувати, що результати емпіричного аналізу залежать від вибору стратегій, включених до моделі, та способу отримання виграшів. Якщо набір стратегій є неповним або не відображає суттєвих варіантів поведінки, сформована емпірична гра може описувати лише частину початкової стратегічної ситуації. Тому під час інтерпретації результатів необхідно враховувати, що емпіричне табличне представлення функцій виграшів є скінченим представленням взаємодії, а не обов'язково повним описом усієї системи [6, 7].

У цій роботі зазначене обмеження враховується через чітке визначення меж програмної реалізації. Розроблене середовище призначене для аналізу вже сформованих ігор у нормальній формі та не виконує автоматичного відновлення функцій виграшу з початкових даних. Його призначення полягає в тому, щоб на основі заданих таблиць виграшів надати користувачеві інструменти для структурного аналізу, візуалізації та порівняння ігор у спільному просторі ознак.

2.4 Граф відповідей і структурні характеристики гри

Після задання табличного представлення функцій виграшів та визначення найкращих відповідей виникає задача подання гри як системи стратегічних переходів. Для цього використовується граф відповідей, який дозволяє перейти від табличного подання виграшів до аналізу структурних властивостей гри. У такому поданні вершини відповідають стратегічним профілям, а напрямлені ребра відображають можливі односторонні переходи гравців, що приводять до покращення їхнього виграшу.

Для двоосібної гри в нормальній формі множина вершин графа відповідей визначається як множина всіх стратегічних профілів:

$$V = S_1 \times S_2.$$

Кожна вершина відповідає парі стратегій (s_1^i, s_2^j) . Напрявлене ребро між двома вершинами існує тоді, коли відповідні стратегічні профілі відрізняються стратегією лише одного гравця, а зміна цієї стратегії збільшує його виграш. Формально для профілів s та s' ребро (s, s') належить множині E , якщо існує гравець i , для якого виконується:

$$\begin{aligned} s_{-i} &= s'_{-i}, \\ u_i(s'_i) &> u_i(s). \end{aligned}$$

Отже, граф відповідей можна подати у вигляді:

$$G = (V, E),$$

де V – множина вершин, що відповідає множині чистих стратегічних профілів, а E – множина напрямлених ребер, що відповідають одностороннім покращувальним переходам. Такий підхід узгоджується з графовим аналізом ігор, у якому структура взаємодії між стратегіями досліджується через вершини, ребра та властивості відповідного графа [1, 8].

У межах цієї роботи граф відповідей будується саме як граф односторонніх покращувальних переходів між стратегічними профілями. Це важливо відрізнити від ширшого підходу, пов'язаного з α -Rank response graph, де ребра можуть інтерпретуватися через імовірності переходів у межах еволюційної моделі. У цьому підпункті розглядається базовий граф відповідей, а α -Rank та пов'язане з ним ранжування профілів розглядаються окремо.

Граф відповідей дозволяє інтерпретувати рівноважні та нестійкі профілі з погляду можливих переходів. Якщо з вершини не виходить жодного ребра, то жоден гравець не може односторонньо змінити свою стратегію та покращити виграш. Така вершина відповідає чистій рівновазі Неша. Якщо ж із вершини виходять ребра, відповідний профіль не є стійким, оскільки принаймні один із гравців має вигідне одностороннє відхилення.

Окрім виявлення рівноважних профілів, граф відповідей дає змогу обчислювати структурні характеристики гри. До базових характеристик належать кількість вершин, кількість ребер, щільність графа, вхідні та вихідні степені вершин, кількість циклів і кількість сильно зв'язаних компонент. Кількість вершин визначається кількістю стратегічних профілів гри. Для двоосібної гри з m стратегіями першого гравця та n стратегіями другого гравця вона дорівнює:

$$|V| = m \cdot n.$$

Кількість ребер $|E|$ показує, скільки покращувальних переходів існує між стратегічними профілями. Щільність напрямленого графа (або орієнтованого графа) без петель можна визначити як:

$$D = \frac{|E|}{|V|(|V|-1)}.$$

Цей показник характеризує насиченість графа переходами. Висока щільність може свідчити про велику кількість можливих односторонніх

покращень, тоді як низька щільність вказує на більш обмежену структуру переходів.

Вхідний степінь вершини показує, скільки переходів веде до відповідного стратегічного профілю, а вихідний степінь – скільки покращувальних переходів можливі з нього. Профілі з високим вхідним ступенем можуть розглядатися як такі, до яких часто ведуть покращення інших профілів. Профілі з високим вихідним ступенем, навпаки, мають багато можливостей для вигідного відхилення, тому є менш стійкими з погляду локальної стратегічної поведінки.

Важливою характеристикою графа відповідей є наявність циклів. Цикл означає, що послідовність односторонніх покращень може повернути гру до вже пройденого стратегічного профілю. Це свідчить про нетранзитивний характер взаємодії, коли локальні покращення не обов'язково приводять до остаточного стійкого стану. У сучасних дослідженнях циклічність графа відповідей розглядається як одна з ознак складності стратегічної структури гри [1].

Ще однією важливою характеристикою є сильно зв'язані компоненти. Сильно зв'язана компонента в напрямленому графі – це така підмножина вершин, у якій кожна вершина досяжна з будь-якої іншої вершини цієї підмножини. У контексті графа відповідей такі компоненти можуть відображати групи стратегічних профілів, між якими існує взаємна досяжність через послідовності покращувальних переходів. Це дає змогу виявляти структурні області гри, що мають внутрішню циклічність або складні взаємозв'язки.

У розробленому програмному середовищі граф відповідей використовується не лише для візуалізації окремої гри, а й для формування набору ознак, необхідних для побудови ландшафту ігор. До таких ознак належать кількість стратегій, кількість стратегічних профілів, кількість ребер, щільність графа, середні та максимальні значення вхідних і вихідних ступенів,

кількість рівноваг Неша, кількість циклів, кількість сильно зв'язаних компонент та нормалізовані структурні показники. Таким чином, граф відповідей є проміжною ланкою між таблицею виграшів і подальшим порівнянням ігор у спільному просторі структурних характеристик.

2.5 α -Rank та ранжування стратегічних профілів

Після побудови графа відповідей виникає задача не лише виявити рівноважні профілі або цикли, а й оцінити відносну важливість окремих стратегій чи стратегічних профілів у загальній структурі гри. Для цього можуть використовуватися методи ранжування, які дозволяють визначити, які стани мають більшу вагу з погляду довгострокової динаміки переходів. Одним із таких підходів є α -Rank – метод багатоагентного оцінювання, що ґрунтується на еволюційній інтерпретації взаємодії між стратегіями [1, 9].

На відміну від рівноваги Неша, яка визначає локально стійкий стратегічний профіль з погляду односторонніх відхилень, α -Rank дозволяє оцінювати стратегії або профілі за їхньою роллю у ширшій системі переходів. Метод розглядає гру як динамічну структуру, у якій можливі переходи між станами залежать від порівняння виграшів. Якщо певний стан частіше досягається в межах такої динаміки або має більшу довгострокову вагу, він отримує вищий ранг.

Основою α -Rank є побудова марковської моделі переходів між стратегічними станами. Нехай множина станів гри складається зі стратегічних профілів або стратегій, для яких необхідно визначити ранги. Для цієї множини стратегічних профілів формується матриця переходів:

$$P = (p_{ij}),$$

де p_{ij} – імовірність переходу від стратегічного профілю i до стратегічного профілю j . Значення таких імовірностей залежать від того, наскільки вигідним є відповідний перехід для гравця або популяції агентів. Якщо перехід веде до стану з кращим результатом, його ймовірність зростає; якщо ж такий перехід не покращує результат, його вага є меншою.

Ключовим результатом такого підходу є стаціонарний розподіл марковського процесу:

$$\pi P = \pi,$$

де π – вектор стаціонарних імовірностей. Компонента π_i показує довгострокову значущість відповідного стратегічного профілю. Чим більшою є ця величина, тим важливішим вважається відповідний стратегічний профіль або стратегія у структурі гри. Таким чином, α -Rank дає змогу перейти від простого визначення стійких станів до кількісного оцінювання ролі різних елементів стратегічного простору [9].

Параметр α у методі α -Rank пов'язаний із силою відбору, тобто з тим, наскільки суттєво різниця у виграшах впливає на ймовірності переходів. За малих значень α відмінності у виграшах мають слабший вплив, тоді як за великих значень переходи до більш вигідних стратегій стають значно ймовірнішими. У граничному випадку метод сильніше відображає напрямки стратегічного домінування та покращення виграшів [9].

У контексті цієї дипломної роботи α -Rank розглядається як методологічна основа для ранжування стратегічних профілів і аналізу їхньої відносної значущості. Такий підхід доповнює класичний пошук рівноваг Неша. Рівновага Неша показує, чи є профіль локально стійким щодо односторонніх відхилень, тоді як ранжування дозволяє оцінити, наскільки важливим є профіль у загальній структурі переходів гри. Це особливо корисно для ігор, у яких чисті рівноваги відсутні або не дають достатньо повного уявлення про характер стратегічної взаємодії.

Водночас важливо розрізнити повну реалізацію α -Rank і використання пов'язаних із ним ідей для аналізу структури гри. Повний метод α -Rank передбачає побудову спеціальної еволюційної моделі стратегічних переходів, обчислення відповідної матриці переходів і знаходження стаціонарного розподілу [9].

У реалізованому програмному середовищі наближена оцінка α -Rank обчислюється не через повну еволюційну модель, а на основі графа відповідей, побудованого для стратегічних профілів. Для цього до напрямленого графа переходів застосовується алгоритм PageRank з коефіцієнтом загасання 0,85: профіль отримує вищу оцінку, якщо до нього ведуть переходи з інших значущих профілів. Отримані значення нормуються так, щоб їхня сума дорівнювала одиниці, тому вони інтерпретуються як відносний розподіл значущості стратегічних профілів у структурі гри. Така оцінка зберігає ідею ранжування станів за структурою переходів, але не є повною реалізацією класичного α -Rank, оскільки не передбачає побудови саме α -Rank еволюційної моделі переходів.

У межах розробленого програмного середовища таке ранжування використовується як один з інструментів оцінювання відносної важливості стратегічних профілів у графовому поданні гри. Для візуального аналізу результати ранжування можуть бути подані у вигляді таблиці, діаграми розподілу або додаткового виділення вершин у графі відповідей. Профілі з вищим рангом можуть інтерпретуватися як більш значущі у структурі стратегічної взаємодії, тоді як профілі з нижчим рангом мають меншу відносну вагу. Таке подання дозволяє користувачеві не лише побачити рівноважні стани, а й оцінити роль інших профілів, які можуть бути важливими для розуміння загальної динаміки гри.

2.6 Спектральний аналіз графів і кластеризація стратегічних профілів

Після побудови графа відповідей виникає задача узагальнення його структури та виявлення груп стратегічних профілів зі схожою роллю у грі. Якщо кількість профілів є невеликою, граф можна аналізувати безпосередньо за вершинами, ребрами та циклами. Однак зі збільшенням кількості стратегій візуальне подання графа ускладнюється, а інтерпретація всіх переходів між профілями стає менш наочною. Для розв'язання цієї задачі доцільно використовувати спектральний аналіз графів і кластеризацію, які дозволяють виявляти приховану структуру зв'язків між вершинами [8, 10, 11].

Основою спектрального аналізу є подання графа за допомогою матриць. Для графа відповідей може бути побудована матриця суміжності:

$$A = (a_{ij}),$$

де a_{ij} показує наявність або вагу ребра між вершинами i та j . Оскільки граф відповідей є орієнтованим, така матриця загалом не є симетричною. Водночас для побудови спектрального подання часто використовується симетризоване представлення графа, яке дозволяє застосувати класичні методи спектрального аналізу. Наприклад, може бути використана матриця:

$$A^{sym} = \frac{A + A^T}{2},$$

яка враховує наявність зв'язків між вершинами без збереження повної інформації про напрям переходу.

На основі матриці суміжності формується матриця степенів:

$$D = \text{diag}(d_1, d_2, \dots, d_n),$$

де d_i – степінь відповідної вершини. Для симетризованого графа степінь вершини може визначатися як сума елементів відповідного рядка матриці суміжності:

$$d_i = \sum_j a_{ij}^{sym}.$$

Далі будується лапласіан графа:

$$L = D - A^{sym},$$

де L – матриця лапласіана, D – матриця степенів, A^{sym} – матриця суміжності. Лапласіан є одним із центральних об'єктів спектральної теорії графів, оскільки його власні значення та власні вектори відображають важливі властивості структури графа, зокрема зв'язність, розділення на групи та характер взаємозв'язків між вершинами [10].

Спектральне подання ґрунтується на розв'язанні задачі на власні значення:

$$Lx = \lambda x,$$

де λ – власне значення, а x – відповідний власний вектор. Власні вектори лапласіана можуть використовуватися як нові координати вершин графа. У результаті кожен стратегічний профіль отримує положення у просторі меншої розмірності, де близькість між вершинами відображає подібність їхнього положення у структурі графа.

У контексті аналізу ігор це означає, що стратегічні профілі, які мають схожу роль у системі переходів, можуть бути розташовані близько один до одного у спектральному просторі. Наприклад, профілі, що належать до однієї області циклічних переходів або мають подібні зв'язки з іншими профілями, можуть утворювати компактні групи. Це робить спектральне подання корисним не лише для візуалізації, а й для подальшої кластеризації стратегічних профілів.

Кластеризація стратегічних профілів полягає у поділі множини вершин графа на групи таким чином, щоб профілі всередині однієї групи були більш подібними між собою за структурою зв'язків, ніж профілі з різних груп. Для цього можуть використовуватися координати, отримані в результаті спектрального аналізу. Після побудови спектрального простору до вершин застосовується алгоритм кластеризації, унаслідок чого кожному стратегічному профілю ставиться у відповідність номер кластера.

Оскільки граф відповідей є орієнтованим, під час кластеризації важливо враховувати, що напрям переходів містить додаткову інформацію про структуру гри. Для певних алгоритмів може використовуватися симетризоване представлення графа, однак інтерпретація результатів має враховувати початкову напрямленість стратегічних переходів. У дослідженнях, присвячених кластеризації орієнтованих мереж, підкреслюється, що напрям зв'язків є важливим чинником для аналізу спільнот і структурних груп у таких графах [12].

Результатом кластеризації є подання гри на більш узагальненому рівні. Замість аналізу кожного стратегічного профілю окремо можна розглядати групи профілів, які мають подібну роль у графі відповідей. Наприклад, один кластер може містити профілі, з яких переважно виходять покращувальні переходи, інший – профілі, до яких веде багато переходів, а ще інший – профілі, що беруть участь у циклічних структурах. Така інтерпретація допомагає зрозуміти загальну організацію стратегічного простору гри.

На основі кластеризації також може бути побудований скорочений граф кластерів. У такому графі кожен кластер розглядається як узагальнена вершина, а ребра між кластерами відображають наявність переходів між стратегічними профілями, що належать до різних груп. Якщо між двома кластерами існує багато переходів, відповідне ребро може мати більшу вагу. Це дозволяє спростити візуальне подання складної гри та перейти від детального графа профілів до узагальненої схеми взаємодії між групами станів.

У розробленому програмному середовищі спектральний аналіз і кластеризація використовуються як інструменти візуального та структурного узагальнення графа відповідей. Вони дозволяють не лише побачити окремі переходи між профілями, а й виявити більші структурні області гри. Отримані результати можуть бути використані для побудови спектрального графа,

кластеризованого графа, скороченого графа кластерів, а також як додаткові ознаки під час порівняння ігор у межах ландшафту.

2.7 Методи зниження розмірності та побудова ландшафту ігор

Після аналізу окремої гри виникає задача порівняння різних ігор між собою. Якщо для однієї гри основними об'єктами аналізу є таблиця виграшів, граф відповідей, рівноважні профілі, цикли та структурні характеристики, то для множини ігор необхідно перейти до єдиного способу їх кількісного опису. Для цього кожна гра подається у вигляді вектора ознак, який узагальнює її основні властивості. Надалі такі вектори можуть бути використані для побудови ландшафту ігор – візуального представлення множини ігор у спільному просторі структурних характеристик.

Основна ідея такого підходу полягає в тому, що гра розглядається не лише через табличне представлення функцій виграшів, а як об'єкт, якому можна поставити у відповідність набір числових характеристик. Ці характеристики можуть описувати як розмір гри, так і її внутрішню структуру. До них належать кількість стратегій гравців, кількість стратегічних профілів, кількість ребер у графі відповідей, щільність графа, середні та максимальні значення вхідних і вихідних степенів, кількість чистих рівноваг Неша, кількість циклів, кількість сильно зв'язаних компонент, ентропія розподілу рангів та інші показники, що характеризують структуру стратегічних переходів. Саме такий підхід використовується в дослідженнях ландшафту ігор, де графові характеристики застосовуються для порівняння ігор різного розміру та структури [1].

Формально для набору ігор

$$G_1, G_2, \dots, G_m$$

кожній грі можна поставити у відповідність вектор характеристик:

$$x_i = (x_{i1}, x_{i2}, \dots, x_{ik}),$$

де x_i – вектор ознак i -ї гри, а k – кількість обчислених характеристик. Наприклад, одна координата такого вектора може відповідати кількості вершин графа відповідей, інша – кількості ребер, ще інша – кількості циклів або значенню щільності графа. У результаті кожна гра описується не початковою таблицею виграшів, а набором структурних властивостей, придатних для порівняння з іншими іграми.

Якщо розглядається множина з m ігор, то всі вектори ознак можна об'єднати в матрицю характеристик:

$$X = \begin{pmatrix} x_{11} & \cdots & x_{1k} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mk} \end{pmatrix}$$

У цій матриці кожен рядок відповідає окремій грі, а кожен стовпець – певній структурній характеристиці. Така матриця є основою для подальшого аналізу множини ігор. Вона дозволяє перейти від якісного порівняння ігор до кількісного аналізу їхньої подібності.

Однак безпосередньо візуалізувати простір ознак складно, якщо кількість характеристик p є більшою за дві або три. У практичних задачах кількість ознак зазвичай є значно більшою, тому кожна гра фактично є точкою у багатовимірному просторі. Для того щоб подати ці точки у зручному для інтерпретації вигляді, необхідно застосувати методи зниження розмірності. Такі методи дозволяють перейти від багатовимірного простору характеристик до двовимірного або тривимірного простору, зберігши при цьому найбільш важливу інформацію про відмінності між об'єктами.

Перед застосуванням методів зниження розмірності необхідно виконати попередню обробку ознак. Це пов'язано з тим, що різні характеристики можуть мати різні масштаби. Наприклад, кількість вершин або ребер може набувати значно більших числових значень, ніж щільність графа або

нормалізована кількість циклів. Якщо не виконати масштабування, ознаки з більшими абсолютними значеннями можуть мати непропорційно великий вплив на результат. Тому доцільно застосовувати стандартизацію ознак:

$$z_{kj} = \frac{x_{kj} - \mu_j}{\sigma_j},$$

де x_{kj} – початкове значення j -ї ознаки для k -ї гри, μ_j – середнє значення цієї ознаки за всіма іграми, σ_j – стандартне відхилення, а z_{kj} – стандартизоване значення. Після такої обробки кожна ознака має нульове середнє значення та одиничне стандартне відхилення. Це дозволяє порівнювати ознаки між собою та зменшує ризик того, що одна характеристика буде домінувати лише через масштаб вимірювання.

Основним методом зниження розмірності в цій роботі є метод головних компонент. Його ідея полягає в тому, щоб замінити початковий набір взаємопов'язаних ознак новими координатами, які називаються головними компонентами. Головні компоненти є лінійними комбінаціями початкових ознак і будуються так, щоб перша компонента пояснювала найбільшу частину варіації даних, друга – найбільшу частину залишкової варіації за умови ортогональності до першої, і так далі. Завдяки цьому можна зменшити розмірність даних, зберігши найбільш інформативні напрями мінливості.

Нехай Z – стандартизована матриця ознак. Метод головних компонент передбачає обчислення коваріаційної матриці

$$C = \frac{1}{m-1} Z^T Z,$$

де m – кількість ігор. Далі для матриці C розв'язується задача на власні значення:

$$C w_i = \lambda_i w_i,$$

де w_i – власний вектор, що задає напрям i -ї головної компоненти, а λ_i – відповідне власне значення. Власне значення показує, яку частину варіації даних пояснює відповідна компонента. Якщо власні значення впорядкувати за

спаданням, то перші компоненти міститимуть найбільшу кількість інформації про відмінності між іграми.

Проекція початкових даних на простір головних компонент виконується за формулою

$$Y = ZW,$$

де W – матриця, стовпцями якої є вибрані власні вектори, а Y – матриця координат ігор у просторі головних компонент. Для двовимірного ландшафту використовується матриця W , що містить перші два власні вектори, а для тривимірного – перші три. У такому поданні кожна гра отримує координати (PC_1, PC_2) або (PC_1, PC_2, PC_3) , які використовуються для побудови графічної візуалізації.

Важливою характеристикою методу головних компонент є частка поясненої дисперсії. Для i -ї компоненти вона визначається як

$$r_i = \frac{\lambda_i}{\sum_{j=1}^p \lambda_j}.$$

Цей показник демонструє, яку частину загальної мінливості даних пояснює відповідна головна компонента. Якщо перші дві або три компоненти пояснюють значну частину дисперсії, то побудований ландшафт добре відображає основні відмінності між іграми. Якщо ж частка поясненої дисперсії є малою, то двовимірна або тривимірна проекція може втрачати частину важливої інформації, і результати потрібно інтерпретувати обережніше.

У контексті ландшафту ігор перша головна компонента може відображати один узагальнений напрям відмінностей між іграми, а друга – інший. Наприклад, якщо найбільші коефіцієнти першої компоненти відповідають кількості циклів, щільності графа і середньому вихідному степеню, то цю компоненту можна інтерпретувати як напрям, пов'язаний із насиченістю та циклічністю стратегічних переходів. Якщо друга компонента має значний зв'язок із кількістю рівноваг, кількістю сильно зв'язаних компонент або ентропією рангів, то вона може відображати інший аспект

структури гри. Тобто метод головних компонент дозволяє не лише побудувати візуалізацію, а й частково пояснити, які саме характеристики впливають на розташування ігор у ландшафті.

Коефіцієнти головних компонент мають важливе інтерпретаційне значення. Вони показують, з якою вагою початкові ознаки входять до складу кожної компоненти. Якщо певна ознака має великий за модулем коефіцієнт, це означає, що вона суттєво впливає на координату гри вздовж відповідної осі. Додатний або від'ємний знак коефіцієнта показує напрям впливу. Завдяки цьому можна визначити, які саме структурні властивості найбільше формують розміщення ігор на ландшафті.

Побудований ландшафт інтерпретується через взаємне розташування точок. Якщо дві гри розміщені близько одна до одної, це означає, що їхні вектори структурних характеристик є подібними. Така подібність не обов'язково означає однакові таблиці виграшів, але свідчить про близькість структурної організації стратегічних переходів. Наприклад, дві ігри можуть мати різні значення виграшів, але подібну кількість циклів, близьку щільність графа відповідей і схожу кількість рівноважних профілів. У такому випадку вони можуть опинитися поруч у просторі головних компонент.

Навпаки, якщо гра розташована далеко від інших, це може свідчити про специфічну структуру. Така гра може мати нетипово велику кількість циклів, відсутність рівноваг, високу щільність графа або складну структуру сильно зв'язаних компонент. У практичному аналізі це дає змогу виявляти ігри, які потребують окремої інтерпретації, а також групи ігор, що мають спільні структурні риси. Саме тому ландшафт ігор є зручним інструментом не лише для візуалізації, а й для попередньої класифікації стратегічних середовищ.

У межах цієї роботи ландшафт будується для множини класичних, синтетичних і прикладних ігор. Класичні ігри використовуються як базові орієнтири, оскільки їхня структура є відомою та інтерпретованою. Синтетичні ігри дозволяють перевірити роботу методів на ширшому наборі штучно

сформованих прикладів. Прикладні ігри дають змогу оцінити, як реалістичніші або спеціально побудовані моделі розташовуються відносно класичних структур. У результаті користувач може бачити не лише окремий аналіз гри, а й її місце серед інших ігор.

Крім методу головних компонент, для побудови ландшафту можуть застосовуватися нелінійні методи зниження розмірності, зокрема UMAP. Метод UMAP призначений для побудови низьковимірною представлення даних із намаганням зберегти локальну структуру багатовимірною простору [13]. На відміну від PCA, який є лінійним методом, UMAP може краще відображати складні нелінійні залежності між об'єктами. Це може бути корисним тоді, коли ігри утворюють групи, які не можна добре відокремити за допомогою лінійної проєкції.

Загальна ідея UMAP полягає в тому, що у початковому просторі ознак будується граф сусідства між об'єктами, після чого шукається таке низьковимірне подання, у якому локальні відношення між сусідніми точками зберігаються якомога краще. У контексті ландшафту ігор це означає, що метод намагається зберегти близькість між іграми, які мають подібні структурні характеристики. Якщо PCA дає більш інтерпретовані координати через лінійні комбінації ознак, то UMAP може краще показувати локальні групи й кластери в даних.

Водночас результати UMAP потрібно інтерпретувати обережно. По-перше, цей метод залежить від параметрів, зокрема кількості сусідів і мінімальної відстані між точками у низьковимірному просторі. По-друге, координати UMAP не мають такої прямої інтерпретації через початкові ознаки, як головні компоненти в PCA. Тому в цій роботі PCA використовується як основний метод побудови ландшафту, оскільки він дозволяє аналізувати внесок окремих характеристик, а UMAP розглядається як додатковий інструмент для альтернативного відображення структури множини ігор.

Процес побудови ландшафту ігор у розробленому програмному середовищі можна описати як послідовність етапів. Спочатку для кожної гри формується таблиця виграшів і будується граф відповідей. Далі для цього графа обчислюються структурні характеристики, які разом утворюють вектор ознак гри. Після цього формується спільна таблиця ознак для всіх ігор, що беруть участь у порівнянні. На наступному етапі ознаки масштабуються, після чого застосовується метод зниження розмірності. Отримані координати використовуються для побудови двовимірного або тривимірного графічного представлення.

У візуальному середовищі ландшафт ігор має бути не лише статичним зображенням, а й засобом інтерпретації результатів. Для цього кожній точці може відповідати назва гри або її числовий ідентифікатор. Додатково може виводитися легенда, таблиця ознак, значення поясненої дисперсії для головних компонент, а також коефіцієнти впливу початкових характеристик на компоненти. Це дозволяє користувачеві не лише бачити, які ігри розташовані поруч, а й розуміти, за рахунок яких структурних властивостей формується таке розташування.

Окрему роль відіграє таблиця ознак, на основі якої будується ландшафт. Вона дозволяє перевірити числові значення характеристик для кожної гри та зіставити їх із положенням точки на графіку. Наприклад, якщо певна гра розташована далеко від інших, можна звернутися до таблиці ознак і визначити, які саме показники відрізняють її від решти. Це підвищує прозорість аналізу та зменшує ризик неправильної інтерпретації візуалізації.

Під час побудови ландшафту важливо також враховувати вплив розміру гри. Ігри з більшою кількістю стратегій природно можуть мати більше стратегічних профілів, більше ребер і потенційно більше циклів. Якщо використовувати лише абсолютні значення характеристик, великі ігри можуть відокремлюватися від малих не через відмінність у структурі, а через масштаб. Тому доцільно додавати нормалізовані ознаки, наприклад нормалізовану

кількість ребер, нормалізовану кількість циклів, відносну кількість рівноваг або щільність графа. Такі ознаки дозволяють порівнювати ігри різного розміру більш коректно.

Ще одним важливим аспектом є добір ознак. Якщо включити до аналізу надто багато взаємозалежних характеристик, деякі властивості можуть бути представлені надмірно. Наприклад, кількість вершин, кількість стратегій і кількість профілів можуть бути тісно пов'язаними між собою. Тому під час аналізу результатів необхідно враховувати, які саме ознаки включені до таблиці, чи не дублюють вони одна одну та які з них найбільше впливають на головні компоненти. У цьому допомагає аналіз коефіцієнтів PCA.

Ландшафт ігор не є остаточним доказом еквівалентності або відмінності ігор, але є ефективним інструментом попереднього порівняльного аналізу. Близьке розташування ігор свідчить про подібність за обраними ознаками, але не гарантує повної стратегічної еквівалентності. Водночас віддалене розташування вказує на відмінність у межах використаного набору характеристик, але також залежить від способу масштабування, вибору ознак і методу зниження розмірності. Тому результати ландшафту мають інтерпретуватися разом із детальним аналізом окремих ігор, їхніх таблиць виграшів, графів відповідей і структурних характеристик.

2.8 Висновки до розділу 2

У другому розділі було розглянуто методи аналізу ігор у нормальній формі, які є основою для розроблення програмного середовища. Визначено, що початковою моделлю дослідження є скінченна двоосібна гра, подана за допомогою матриць виграшів. Таке подання дозволяє формалізувати

стратегічні профілі, обчислювати виграші гравців і використовувати ці матриці як вхідну структуру для подальшого аналізу.

Розглянуто поняття найкращої відповіді, домінування стратегій та рівноваги Неша. Ці поняття дають змогу визначати стійкі стратегічні профілі та аналізувати можливість односторонніх відхилень гравців. Окремо зазначено, що пошук лише рівноважних станів не завжди є достатнім для повної інтерпретації гри, тому його доцільно доповнювати графовими та структурними методами.

Також було розглянуто емпіричну гру як скінченне представлення стратегічної взаємодії, побудоване на основі заданого набору стратегій і відповідних значень виграшів. У межах цієї роботи подальший аналіз виконується для вже сформованих біматричних представлень виграшів, що відповідає аналітико-візуалізаційному етапу емпіричного теоретико-ігрового аналізу.

Важливу роль у методичному апараті роботи відіграє граф відповідей, який дозволяє подати гру як систему односторонніх покращувальних переходів між стратегічними профілями. На його основі можуть бути визначені структурні характеристики гри: кількість вершин і ребер, щільність графа, вхідні та вихідні степені, цикли, сильно зв'язані компоненти та інші показники. Саме ці характеристики надалі використовуються для порівняння різних ігор між собою.

Окремо розглянуто α -Rank як методологічну основу для ранжування стратегічних профілів, а також спектральний аналіз і кластеризацію як засоби узагальнення структури графа відповідей. Ці методи дозволяють не лише досліджувати окремі переходи між профілями, а й виявляти групи профілів зі схожою структурною роллю у грі.

Завершальним елементом методичного апарату є побудова ландшафту ігор. Для цього кожна гра подається у вигляді вектора структурних характеристик, після чого застосовуються методи зниження розмірності,

зокрема метод головних компонент та UMAP. Це дає змогу перейти від аналізу окремої гри до порівняння множини ігор у спільному просторі ознак.

РОЗДІЛ 3 РОЗРОБЛЕННЯ ПРОГРАМНОГО СЕРЕДОВИЩА ТА АНАЛІЗ РЕЗУЛЬТАТІВ

У цьому розділі розглянуто розроблене програмне середовище для аналізу ігор у нормальній формі, його функціональні можливості, архітектуру та результати застосування до різних типів ігор. Основну увагу приділено реалізації аналітико-візуалізаційного етапу емпіричного аналізу, тобто роботі з уже сформованими таблицями виграшів, побудові графів відповідей, обчисленню структурних характеристик і формуванню ландшафту ігор.

Розроблене середовище дозволяє аналізувати як класичні, так і синтетичні та прикладні ігри. У межах розділу описано основні модулі програмної реалізації, набір ігор для обчислювальних експериментів, результати аналізу окремої гри та побудову ландшафту для порівняння ігор за структурними характеристиками.

3.1 Загальна характеристика розробленого програмного середовища

Розроблене програмне середовище призначене для аналізу ігор у нормальній формі на основі вже сформованих таблиць виграшів. Його основне призначення полягає у підтримці аналітико-візуалізаційного етапу емпіричного аналізу: поданні гри у зручній формі, автоматизованому обчисленні її основних характеристик, побудові графових представлень і формуванні візуальних результатів, які спрощують інтерпретацію стратегічної структури.

У межах реалізованого програмного середовища основна увага приділяється двоосібним іграм. Кожна така гра задається двома матрицями виграшів однакового розміру: перша матриця відповідає виграшам першого гравця, друга – виграшам другого гравця. Рядки матриць відповідають стратегіям першого гравця, а стовпці – стратегіям другого гравця. Таке подання узгоджується з теоретичною моделлю біматричної гри, розглянутою в розділі 2, і є зручним для подальшої автоматизованої обробки.

Програмне середовище реалізовано мовою Python. Для створення веб-інтерфейсу використано бібліотеку Streamlit, що дозволяє запускати застосунок у браузері та взаємодіяти з результатами аналізу через елементи інтерфейсу. Для обробки матриць, таблиць і числових результатів використано бібліотеки NumPy та pandas. Побудова й аналіз графів виконуються за допомогою NetworkX, а візуалізація результатів – за допомогою Matplotlib і Plotly. Для методів зниження розмірності та кластеризації використовуються засоби scikit-learn, зокрема PCA, StandardScaler і KMeans, а також бібліотека umap-learn для побудови UMAP-проекцій.

На рисунку 3.1 наведено робоче вікно розробленого програмного середовища в режимі аналізу окремої гри. У лівій частині інтерфейсу розташована панель налаштувань, за допомогою якої користувач може вибрати режим роботи, гру для аналізу, кількість кластерів і додаткові параметри відображення результатів. У центральній частині вікна відображається назва застосунку, вибрана гра, довідкова інформація про неї, а також вкладки з результатами аналізу.

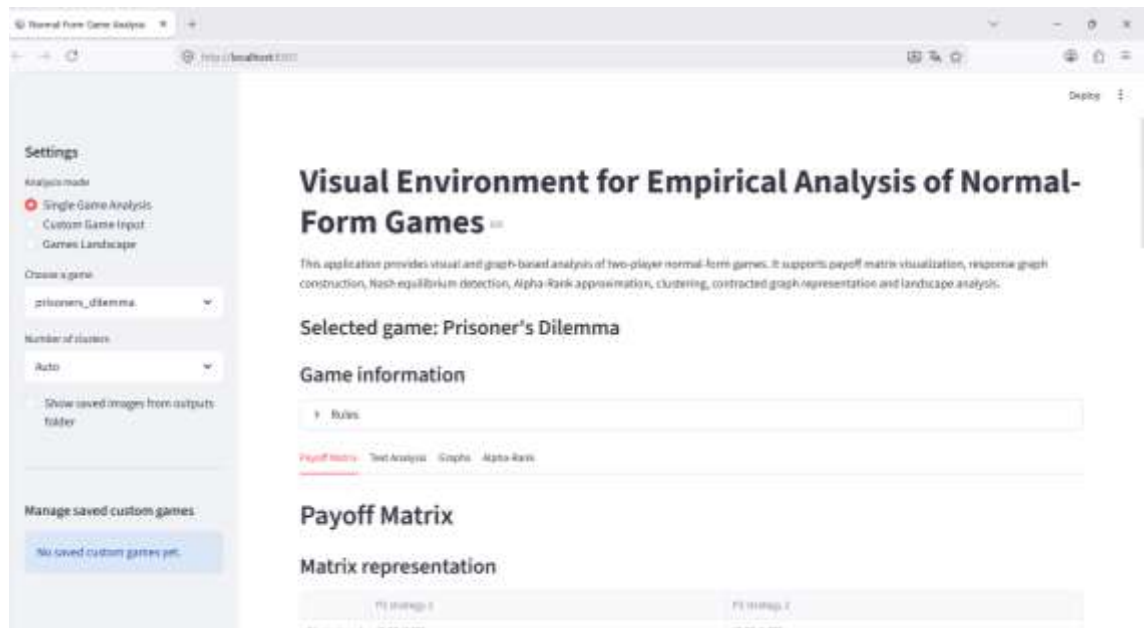


Рисунок 3.1 Інтерфейс програмного середовища в режимі Single Game Analysis на прикладі гри “Дилема в’язня”

У програмному середовищі передбачено три основні режими роботи: Single Game Analysis, Custom Game Input і Games Landscape. Перший режим призначений для детального аналізу однієї вибраної гри. У цьому режимі користувач може обрати гру із запропонованого набору, переглянути її опис, правила та матричне подання. Після вибору гри застосунок автоматично формує матриці виграшів, теплові карти виграшів для обох гравців, таблицю найкращих відповідей, список чистих рівноваг Неша, статистики графа відповідей і відповідні графічні візуалізації.

Результати аналізу окремої гри поділено на кілька вкладок: Payoff Matrix, Text Analysis, Graphs і Alpha-Rank. Така структура інтерфейсу дозволяє послідовно переходити від початкових даних до результатів їх структурного аналізу. У вкладці Payoff Matrix відображаються матриці виграшів обох гравців і теплові карти, які дозволяють візуально оцінити розподіл значень у грі. Це спрощує попередній аналіз, оскільки користувач може одразу побачити, які стратегічні профілі дають більші або менші виграші.

У вкладці Text Analysis подано числові й табличні результати аналізу. Зокрема, виводяться найкращі відповіді гравців, чисті рівноваги Неша,

кількість вершин і ребер графа відповідей, вхідні та вихідні степені вершин, сильно зв'язані компоненти й цикли. Якщо кількість циклів невелика, вони можуть бути показані безпосередньо; якщо циклів багато, застосунок повідомляє їхню загальну кількість, щоб не перевантажувати інтерфейс.

У вкладці Graphs реалізовано побудову кількох типів графічних представлень. Користувач може обрати граф відповідей, спектральний граф, кластеризований граф, скорочений граф кластерів або розподіл циклів. Це дозволяє аналізувати гру на різних рівнях деталізації: від окремих переходів між стратегічними профілями до узагальненого подання груп профілів.

У вкладці Alpha-Rank подано наближене ранжування стратегічних профілів. У програмі воно використовується як додатковий інструмент оцінювання відносної важливості профілів у структурі гри. Результати подаються у вигляді таблиці та діаграми розподілу значень. У межах роботи це ранжування розглядається як наближена оцінка, методологічно пов'язана з ідеями α -Rank, а не як повна реалізація класичного методу [9].

Режим Custom Game Input, представлений на рисунку 3.2, призначений для введення власної гри користувачем. У цьому режимі можна задати назву гри, кількість стратегій першого та другого гравців, а також заповнити окремі матриці виграшів для кожного гравця. Після введення значень користувачка гра може бути проаналізована так само, як і попередньо визначені приклади. Також передбачено можливість збереження користувацької гри та додавання її до загального ландшафту ігор.

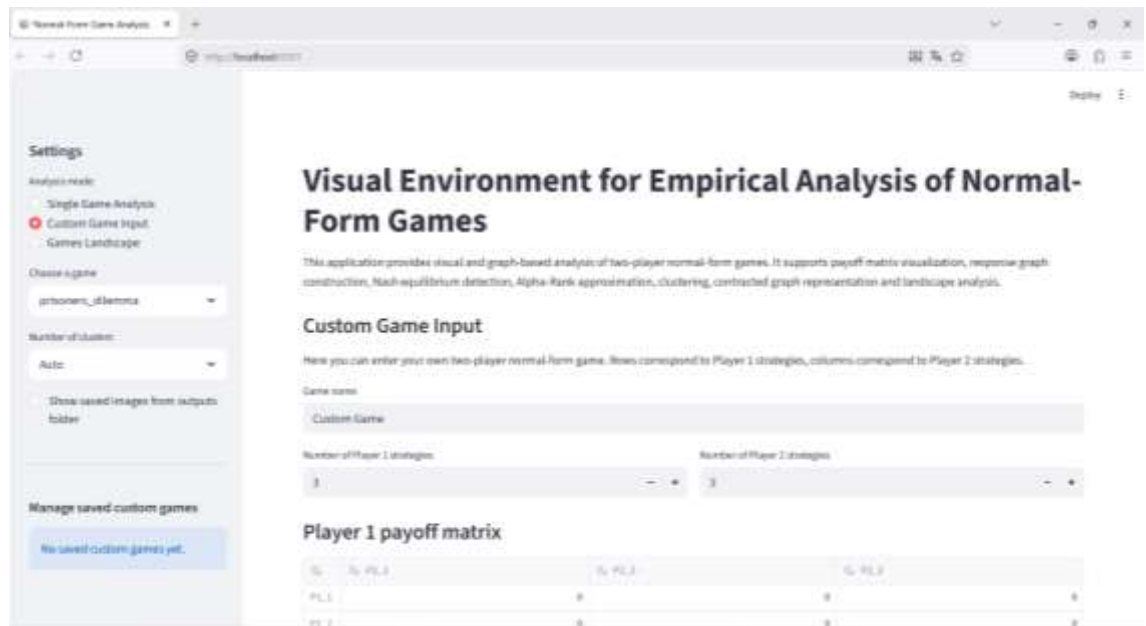


Рисунок 3.2 Інтерфейс програмного середовища в режимі Custom Game Input

Режим Games Landscape (рисунок 3.3) призначений для порівняння кількох ігор між собою. У цьому режимі кожна гра перетворюється на набір числових характеристик, після чого ці характеристики використовуються для побудови двовимірного або тривимірного ландшафту. Користувач може обрати набір ігор, метод зниження розмірності – PCA або UMAP, тип візуалізації – 2D або 3D, а також кількість кластерів. У результаті формується простір, у якому ігри зі схожими структурними характеристиками розташовуються ближче одна до одної. Такий підхід відповідає ідеї ландшафту ігор, у межах якої ігри порівнюються за характеристиками їхньої стратегічної структури [1].

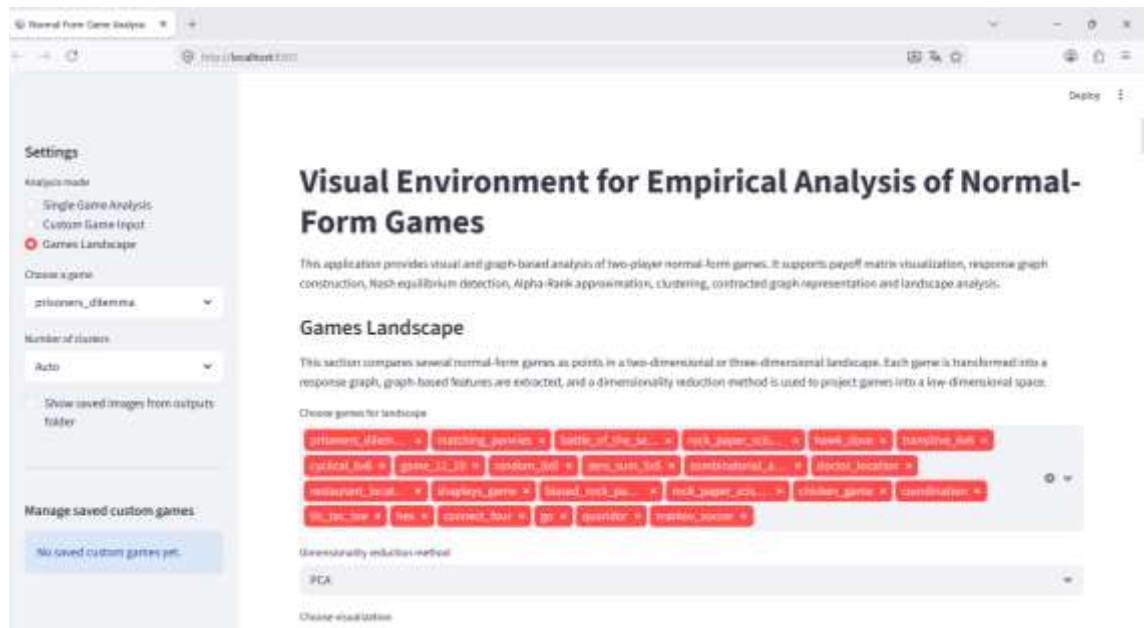


Рисунок 3.3 Інтерфейс програмного середовища в режимі Games Landscape

3.2 Архітектура програмної реалізації

Архітектура програмної реалізації побудована за модульним принципом. Це означає, що окремі частини програми відповідають за різні етапи роботи програмного середовища: подання гри, створення набору прикладів, виконання аналітичних обчислень, побудову графів, формування візуалізацій, побудову ландшафту ігор та відображення результатів у вебінтерфейсі. Такий підхід дозволяє відокремити логіку обробки даних від логіки візуалізації та взаємодії з користувачем. Завдяки цьому програмний продукт є більш зручним для підтримки, перевірки окремих частин і подальшого розширення.

Модульна структура є доцільною для цієї роботи, оскільки аналіз ігор у нормальній формі складається з кількох послідовних етапів. Спочатку гра має бути подана у вигляді матриць виграшів, далі для неї обчислюються найкращі

відповіді, рівноваги Неша, граф відповідей і структурні характеристики. Після цього результати можуть бути подані у вигляді таблиць, графів, спектральних представлень, кластеризованих графів або використані для побудови ландшафту ігор. Винесення цих етапів в окремі модулі дозволяє змінювати або доповнювати один компонент без повного переписування всієї системи.

Загальну структуру проєкту наведено в таблиці 3.1.

Таблиця 3.1. Загальна структура проєкту

Компонент	Призначення
app.py	основний файл веб-застосунку на Streamlit
main.py	запуск повного аналізу ігор у консольному режимі
src/game_model.py	модель гри в нормальній формі
src/games.py	створення набору попередньо визначених ігор
src/analysis.py	обчислення найкращих відповідей, рівноваг Неша, графа відповідей і статистик
src/visualization.py	побудова графічних представлень результатів
src/landscape.py	формування ознак ігор і побудова ландшафту
src/reporting.py	формування текстових і табличних результатів
src/pipeline.py	об'єднання основних етапів аналізу в один сценарій
src/config.py	збереження спільних налаштувань проєкту
outputs/	папка для збережених графіків, таблиць і звітів

Основним файлом застосунку є app.py. Він відповідає за створення вебінтерфейсу, вибір режиму роботи, обробку дій користувача та виведення результатів аналізу. Саме в цьому файлі реалізовано логіку перемикання між основними режимами роботи: аналізом окремої гри, введенням користувацької гри та побудовою ландшафту ігор. Через app.py користувач взаємодіє з усіма функціональними можливостями системи: обирає гру, задає

параметри аналізу, переглядає матриці виграшів, таблиці характеристик, графи відповідей, результати кластеризації та візуалізації ландшафту.

Файл `app.py` не містить усієї логіки обчислень безпосередньо, а викликає функції з інших модулів. Такий підхід дозволяє зберегти структуру застосунку зрозумілою: інтерфейс відповідає за взаємодію з користувачем, а аналітичні модулі – за обчислення. Наприклад, після вибору гри у вебінтерфейсі її дані передаються до модулів аналізу, де обчислюються рівноваги, граф відповідей і структурні характеристики. Після цього отримані результати повертаються до інтерфейсу й відображаються у відповідних вкладках.

Файл `main.py` використовується для запуску аналізу в консольному режимі. Його наявність є корисною під час тестування програмної логіки, оскільки дозволяє перевіряти роботу основних функцій без запуску вебінтерфейсу. Такий режим також може використовуватися для пакетної обробки ігор, збереження результатів у файли або перевірки коректності обчислень під час розроблення нових модулів. Отже, `main.py` виконує допоміжну роль і забезпечує альтернативний спосіб запуску аналітичної частини системи.

Центральною структурою даних у програмі є модель гри, реалізована в модулі `src/game_model.py`. Цей модуль відповідає за подання гри в нормальній формі та збереження її основних параметрів. До таких параметрів належать матриці виграшів гравців, назви стратегій, назва гри, кількість стратегій кожного гравця та інші допоміжні властивості, необхідні для подальшого аналізу. Завдяки використанню єдиної моделі гри всі типи прикладів – класичні, синтетичні, прикладні та користувацькі – можуть оброблятися однаковим способом.

Модуль `src/game_model.py` також забезпечує узгодженість вхідних даних. Для аналізу двоосібної гри необхідно, щоб матриці виграшів першого та другого гравців мали однаковий розмір, оскільки кожна клітинка відповідає одному стратегічному профілю. Якщо ці розміри не збігаються, подальший

аналіз гри стає некоректним. Тому модель гри виконує роль базового рівня перевірки та впорядкування даних перед тим, як вони передаються до аналітичних функцій.

Модуль `src/games.py` містить набір попередньо визначених ігор, які використовуються для демонстрації роботи середовища та проведення обчислювальних експериментів. У ньому задаються класичні ігри, синтетично згенеровані приклади та прикладні моделі. Наявність окремого модуля для ігор є зручною, оскільки дозволяє розширювати набір прикладів без зміни основних алгоритмів аналізу. Якщо потрібно додати нову гру, достатньо описати її матриці виграшів і метадані в цьому модулі, після чого вона може бути використана в інтерфейсі та в ландшафті ігор.

Особливе значення має модуль `src/analysis.py`, у якому зосереджено основні аналітичні обчислення. Він відповідає за визначення найкращих відповідей гравців, пошук чистих рівноваг Неша, побудову графа відповідей і обчислення структурних характеристик гри. Саме цей модуль реалізує перехід від матриці виграшів до формального опису стратегічних переходів. На його основі визначається, які стратегічні профілі є стійкими, які мають вигідні односторонні відхилення, які переходи формують граф відповідей і які властивості має отриманий граф.

До структурних характеристик, що обчислюються в межах аналітичного модуля, належать кількість стратегічних профілів, кількість ребер графа відповідей, щільність графа, вхідні та вихідні степені вершин, кількість циклів, кількість сильно зв'язаних компонент і кількість чистих рівноваг Неша. Ці характеристики використовуються не лише для аналізу окремої гри, а й як ознаки для побудови ландшафту. Таким чином, `src/analysis.py` є одним із ключових модулів системи, оскільки саме він формує числову основу для подальшої візуалізації та порівняння ігор.

Модуль `src/visualization.py` відповідає за побудову графічних представлень результатів аналізу. У ньому формуються візуалізації матриць

виграшів, графів відповідей, спектральних подань, кластеризованих графів, скорочених графів кластерів і розподілів циклів. Винесення функцій візуалізації в окремий модуль дозволяє відокремити обчислювальну частину від графічного подання результатів. Це важливо, оскільки одна й та сама аналітична інформація може бути подана різними способами залежно від режиму роботи застосунку або потреб користувача.

Побудова графа відповідей у візуальному модулі має не лише ілюстративне, а й аналітичне значення. За допомогою графічного подання користувач може побачити, які профілі мають вихідні покращувальні переходи, до яких профілів веде багато ребер, чи є у грі циклічні структури та як розташовані рівноважні стани. Для складніших ігор простий граф відповідей може бути перевантаженим, тому в системі передбачено додаткові способи подання, зокрема спектральний граф і кластеризований граф.

Модуль `src/landscape.py` відповідає за побудову ландшафту ігор. Його призначення полягає у формуванні таблиці структурних ознак для множини ігор, попередній обробці цих ознак, застосуванні методів зниження розмірності та отриманні координат ігор у спільному просторі. У цьому модулі реалізується перехід від аналізу окремої гри до порівняння множини ігор. Для кожної гри формується вектор характеристик, після чого всі такі вектори об'єднуються в єдину таблицю ознак.

На основі таблиці ознак у модулі `src/landscape.py` можуть застосовуватися методи PCA або UMAP. Метод PCA використовується для побудови інтерпретованого простору головних компонент, а UMAP – як альтернативний метод низьковимірного подання, що може краще зберігати локальну структуру даних. Результатом роботи модуля є набір координат, за якими ігри відображаються як точки на двовимірному або тривимірному графіку. Крім того, цей модуль може формувати додаткові дані для інтерпретації: пояснену дисперсію PCA, внесок ознак у компоненти, кластери ігор і таблицю характеристик.

Модуль `src/reporting.py` призначений для формування текстових і табличних результатів. Він використовується для подання обчислених характеристик у зручному для користувача вигляді. Наприклад, результати пошуку рівноваг, статистики графа відповідей, списки циклів, таблиці ознак або пояснення до ландшафту можуть бути сформовані саме через цей модуль. Такий підхід дозволяє відокремити обчислення від оформлення результатів і забезпечує єдиний стиль подання інформації в різних частинах застосунку.

Модуль `src/pipeline.py` виконує роль зв'язувальної ланки між окремими етапами аналізу. Він об'єднує послідовність дій в один сценарій: отримання гри, побудову матриць виграшів, обчислення рівноваг, формування графа відповідей, обчислення структурних характеристик, побудову візуалізацій і підготовку результатів для виведення. Такий модуль є корисним для повторюваних сценаріїв роботи, коли одна й та сама послідовність обчислень має виконуватися для різних ігор.

Модуль `src/config.py` містить спільні налаштування проєкту. До таких налаштувань можуть належати параметри збереження результатів, стандартні значення для візуалізацій, налаштування кількості кластерів, параметри побудови графів, шляхи до папок із результатами та інші службові змінні. Винесення таких параметрів в окремий файл робить систему більш гнучкою, оскільки дозволяє змінювати налаштування без редагування основної логіки аналізу.

Папка `outputs/` використовується для збереження результатів роботи програмного середовища. До неї можуть записуватися побудовані графіки, таблиці, текстові звіти й допоміжні файли, отримані під час аналізу ігор. Наявність окремої папки для результатів є важливою з погляду організації проєкту, оскільки дозволяє відокремити вихідні файли програми від згенерованих результатів.

Загальна логіка обробки даних у програмному середовищі є послідовною. Спочатку користувач обирає одну з попередньо визначених ігор

або задає власну гру через інтерфейс. Далі створюється об'єкт гри в нормальній формі, який містить матриці виграшів і пов'язані з ними метадані. Після цього дані передаються до аналітичного модуля, де виконуються основні обчислення: визначення найкращих відповідей, пошук чистих рівноваг Неша, побудова графа відповідей і розрахунок структурних характеристик.

Після аналітичного етапу результати передаються до модулів візуалізації та звітності. Якщо користувач аналізує окрему гру, система формує матриці виграшів, текстові результати, граф відповідей, спектральне подання, кластеризований граф і результати ранжування профілів. Якщо користувач працює з режимом ландшафту, структурні характеристики кількох ігор передаються до модуля `src/landscape.py`, де формується спільний простір ознак і виконується зниження розмірності.

3.3 Набір ігор для проведення обчислювальних експериментів

Для перевірки роботи розробленого програмного середовища було сформовано набір ігор у нормальній формі, який охоплює класичні, синтетичні, прикладні та завантажені через OpenSpiel приклади. Такий набір є необхідним для того, щоб перевірити роботу реалізованих методів не лише на простих навчальних іграх, а й на іграх із різною кількістю стратегій, різною структурою виграшів, наявністю або відсутністю чистих рівноваг Неша, циклічними переходами та різними типами стратегічної взаємодії.

Добір ігор здійснювався з урахуванням того, що програмне середовище має забезпечувати не тільки аналіз однієї окремої гри, а й порівняння множини ігор між собою. Тому до набору включено ігри різних типів: класичні двоосібні ігри, які мають добре відому структуру; синтетичні ігри більшої

розмірності, що дозволяють перевірити роботу алгоритмів на більш складних матрицях; прикладні ігри, які мають змістовну інтерпретацію; а також ігри, отримані або представлені за допомогою бібліотеки OpenSpiel.

Класичні ігри використовуються як базові контрольні приклади. Їхня структура добре описана в літературі, тому за ними зручно перевіряти коректність пошуку найкращих відповідей, чистих рівноваг Неша, побудови графа відповідей і визначення циклів. Наприклад, дилема в'язня дає змогу перевірити ситуацію з домінуючими стратегіями та єдиною рівновагою, гра «співпадіння монет» демонструє відсутність чистої рівноваги, а гра «камінь-ножиці-папір» є типовим прикладом циклічної нетранзитивної структури.

Синтетичні ігри використовуються для перевірки масштабованості та стійкості реалізованих алгоритмів. Ігри розмірності 5×5 , 6×6 , 7×7 або 10×10 мають значно більшу кількість стратегічних профілів, тому ручний аналіз таких прикладів є менш зручним. Саме на таких іграх добре проявляється необхідність автоматизованого аналізу: побудови графа відповідей, підрахунку кількості ребер, циклів, сильно зв'язаних компонент, кластеризації стратегічних профілів та побудови ландшафту.

Прикладні ігри включено до набору для демонстрації того, що розроблене середовище може працювати не лише з абстрактними теоретичними прикладами, а й з моделями, які мають змістовну інтерпретацію. До таких прикладів належать локаційна гра лікарів і локаційна гра ресторанів. У цих іграх стратегії відповідають вибору певної локації, а виграші формуються на основі розподілу користувачів або клієнтів між гравцями. Такі приклади є корисними для демонстрації зв'язку між матричним поданням гри та прикладною задачею вибору розташування.

Окрему групу становлять ігри, отримані або представлені через OpenSpiel. OpenSpiel є фреймворком для дослідження ігор і багатоагентного навчання, тому його використання дозволяє розширити набір тестових прикладів і показати можливість інтеграції з уже наявними ігровими

середовищами [14]. У межах цієї роботи такі ігри використовуються як додаткові приклади для аналізу та порівняння в ландшафті.

Набір ігор, використаний для проведення обчислювальних експериментів, наведено в таблиці 3.2.

Таблиця 3.2. Набір ігор для проведення обчислювальних експериментів

№	Назва гри	Тип подання	Тип гри і призначення
1	Дилема в'язня	2×2	класична некооперативна гра, перевірка домінування та єдиної рівноваги
2	Гра «співпадіння монет»	2×2	антагоністична гра, приклад відсутності чистої рівноваги
3	Битва статей	2×2	координаційна гра з кількома рівновагами
4	Камінь-ножиці-папір	3×3	циклічна нетранзитивна гра
5	Гра яструб-голуб	2×2	конфліктно-координаційна модель поведінки
6	Гра «курка»	2×2	конфліктна гра з ризиком взаємно не вигідного результату
7	Координаційна гра	2×2	гра узгодження дій між гравцями
8	Гра Шеплі	матрична OpenSpiel-гра	приклад циклічної структури стратегічних переходів
9	Зміщена гра «камінь-ножиці-папір»	матрична OpenSpiel-гра	модифікація циклічної гри з нерівномірними виграшами

Продовження таблиці 3.2

10	Камінь-ножиці-папір-колодязь	матрична OpenSpiel-гра	розширення класичної циклічної гри
11	Транзитивна гра	6×6	синтетична гра з упорядкованою структурою стратегій
12	Циклічна гра	6×6	синтетична гра з циклічною структурою переваг
13	Гра 11–20	10×10	гра стратегічного мислення та рівневої раціональності
14	Випадкова гра	5×5	синтетична гра з випадково згенерованими виграшами
15	Гра з нульовою сумою	5×5	антагоністична гра, де виграш одного гравця є програшем іншого
16	Гра комбінаторного аукціону	7×7	прикладна гра вибору наборів об'єктів
17	Локаційна гра лікарів	7×7	прикладна гра просторового розміщення
18	Локаційна гра ресторанів	7×7	прикладна гра просторової конкуренції
19	Тіс-Тас-Тое	емпіричне подання OpenSpiel	гра з послідовною структурою ходів, використана через обмежений набір стратегій
20	Нех	емпіричне подання OpenSpiel	настільна гра з просторовою структурою та конкурентним заповненням поля

Кінець таблиці 3.2

21	Connect Four	емпіричне подання OpenSpiel	гра з послідовними ходами та комбінаційною структурою
22	Go	емпіричне подання OpenSpiel	стратегічна настільна гра з великою кількістю станів
23	Quoridor	емпіричне подання OpenSpiel	гра з просторовим переміщенням і блокуванням ходів
24	Markov Soccer	емпіричне подання OpenSpiel	стохастична багатоагентна гра з динамікою станів

Наведений набір ігор дозволяє перевірити різні аспекти роботи програмного середовища. На класичних іграх перевіряється коректність базових алгоритмів: визначення найкращих відповідей, пошук рівноваг Неша та побудова графа відповідей. На синтетичних іграх більшої розмірності перевіряється здатність системи обробляти більшу кількість стратегічних профілів і формувати структурні характеристики для складніших графів. На прикладних іграх перевіряється можливість застосування середовища до моделей із практичною інтерпретацією.

Гра 11–20 включена до набору як приклад гри стратегічного мислення та рівневої раціональності. Вона відрізняється від стандартних матричних прикладів тим, що пов'язана з міркуваннями гравців щодо очікуваної поведінки суперника. Такий тип гри є корисним для перевірки того, як програмне середовище працює з матрицями більшої розмірності та складнішою структурою виграшів [15].

Ігри Tic-Tac-Toe, Hex, Connect Four, Go, Quoridor та Markov Soccer мають послідовну або динамічну структуру, тому їх повне подання не є звичайною біматричною грою у нормальній формі невеликого розміру. У межах цієї роботи вони використовуються через емпіричне подання, тобто через обмежений набір стратегій або політик, для яких може бути сформована таблиця виграшів. Такий підхід узгоджується із загальною логікою емпіричного аналізу ігор: дослідник працює не з усім простором можливих станів, а з певним скінченним набором стратегічних варіантів, придатних для подальшого аналізу.

Для кожної гри виконується однакова послідовність аналізу. Спочатку формується або зчитується таблиця виграшів, після чого визначаються найкращі відповіді гравців і чисті рівноваги Неша. Далі будується граф відповідей, у якому вершини відповідають стратегічним профілям, а ребра – одностороннім покращувальним переходам. На основі цього графа обчислюються кількість вершин, кількість ребер, щільність, вхідні та вихідні степені, кількість циклів, кількість сильно зв'язаних компонент та інші характеристики.

Отримані характеристики використовуються не лише для аналізу окремої гри, а й для побудови ландшафту ігор. Завдяки цьому кожна гра може бути представлена як точка у спільному просторі ознак. Ігри зі схожими структурними характеристиками розташовуються ближче одна до одної, а ігри з відмінною структурою – далі. Саме тому набір ігор має бути достатньо різноманітним: він повинен містити і прості контрольні приклади, і складніші ігри, які дозволяють перевірити роботу методів порівняння.

Важливо, що обрані ігри відрізняються не лише розмірністю, а й характером стратегічної взаємодії. Частина з них має одну або кілька чистих рівноваг Неша, частина демонструє відсутність чистих рівноваг, частина містить циклічні стратегічні переходи, а частина має прикладну інтерпретацію. Це дозволяє перевірити, чи здатне програмне середовище

виявляти різні типи стратегічної структури та відображати ці відмінності у графах відповідей, таблицях характеристик і ландшафті ігор.

Таким чином, сформований набір ігор є основою для подальших обчислювальних експериментів у межах розробленого програмного середовища. Він забезпечує можливість перевірити роботу системи в різних сценаріях: аналіз окремої гри, порівняння класичних і синтетичних прикладів, дослідження прикладних моделей та побудова ландшафту ігор за структурними характеристиками.

3.4 Аналіз окремої гри у програмному середовищі

Одним із основних режимів розробленого програмного середовища є режим Single Game Analysis, призначений для детального аналізу окремої гри в нормальній формі. У цьому режимі користувач може обрати гру із попередньо сформованого набору, переглянути її опис і правила, дослідити матрицю виграшів, визначити рівноважні стани, побудувати граф відповідей, проаналізувати його структурні характеристики, виконати спектральне подання, кластеризацію та наближене ранжування стратегічних профілів.

Для демонстрації роботи цього режиму було обрано локаційну гру лікарів. Цей приклад є показовим для дипломної роботи, оскільки поєднує формальне подання гри у вигляді матриці виграшів із прикладною інтерпретацією задачі просторового розміщення. На відміну від простих ігор розмірності 2×2 , локаційна гра лікарів має сім стратегій для кожного гравця, тобто 49 стратегічних профілів. Це робить її достатньо складною для ручного аналізу та водночас зручною для демонстрації переваг автоматизованого графового й візуального аналізу.

У локаційній грі лікарів два гравці обирають місце розташування на заданій мережі з семи локацій. Кожна локація може інтерпретуватися як населений пункт або район, у якому проживає певна кількість мешканців. Після вибору позицій кожним із лікарів мешканці умовної мережі розподіляються між ними залежно від відстані до обраних точок. Якщо мешканець розташований ближче до першого лікаря, він зараховується до виграшу першого гравця; якщо ближче до другого – до виграшу другого гравця. Якщо відстань до обох лікарів однакова, відповідний мешканець розподіляється між ними порівну.

Просторова структура гри подана у вигляді графа, вершини якого відповідають можливим локаціям, а ребра – зв'язкам між ними. Саме ця структура визначає відстані між локаціями та, відповідно, впливає на формування виграшів гравців. У програмному середовищі для локаційних ігор передбачено виведення такого довідкового графа, що полегшує інтерпретацію матриці виграшів і подальших результатів аналізу.

На рисунку 3.4 видно, що локації розташовані нерівномірно: деякі вершини мають більше сусідів і займають більш центральне положення в мережі, тоді як інші є периферійними. Це має безпосередній вплив на стратегічну цінність відповідних позицій. Локація, що має вигідне положення відносно інших вершин, може забезпечувати лікарю більшу кількість потенційних мешканців, а отже, більший виграш. Тому вже на рівні просторової структури можна очікувати, що не всі стратегії будуть однаково вигідними.

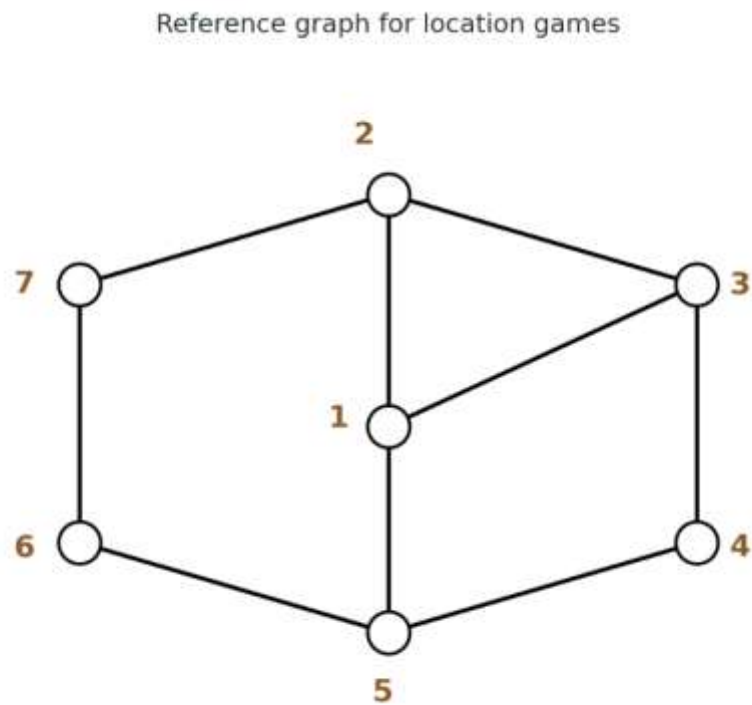


Рисунок 3.4 Просторова структура локаційної гри лікарів

Першим етапом аналізу є перегляд матриці виграшів. Оскільки кожен із двох гравців має по сім стратегій, матриця має розмір 7×7 . Кожна її клітинка відповідає певному стратегічному профілю, тобто парі локацій, обраних першим і другим лікарем. У клітинці записується пара значень: перше значення відповідає виграшу першого гравця, друге – виграшу другого гравця.

Матриця виграшів на рисунку 3.5 показує, як змінюється результат кожного гравця залежно від вибору локацій. Якщо обидва лікарі обирають одну й ту саму локацію, частина мешканців розподіляється між ними симетрично. Якщо ж лікарі обирають різні локації, виграш кожного залежить від того, які вершини мережі розташовані ближче до його позиції. Оскільки в цій грі кожен мешканець або належить одному з лікарів, або ділиться між ними порівну, сума виграшів двох гравців для кожного стратегічного профілю дорівнює загальній кількості локацій, тобто семи. Це означає, що гра має конкурентний характер: збільшення виграшу одного гравця супроводжується зменшенням виграшу іншого.

Matrix representation

	Beth: town 1	Beth: town 2	Beth: town 3	Beth: town 4	Beth: town 5	Beth: town 6	Beth: town 7
Ann: town 1	(3.50, 3.50)	(3.50, 3.50)	(4.00, 3.00)	(4.50, 2.50)	(3.50, 3.50)	(4.00, 3.00)	(4.50, 2.50)
Ann: town 2	(3.50, 3.50)	(3.50, 3.50)	(4.00, 3.00)	(4.00, 3.00)	(3.50, 3.50)	(4.00, 3.00)	(4.50, 2.50)
Ann: town 3	(3.00, 4.00)	(3.00, 4.00)	(3.50, 3.50)	(4.00, 3.00)	(3.50, 3.50)	(4.00, 3.00)	(4.00, 3.00)
Ann: town 4	(2.50, 4.50)	(3.00, 4.00)	(3.00, 4.00)	(3.50, 3.50)	(2.50, 4.50)	(3.50, 3.50)	(3.50, 3.50)
Ann: town 5	(3.50, 3.50)	(3.50, 3.50)	(3.50, 3.50)	(4.50, 2.50)	(3.50, 3.50)	(4.50, 2.50)	(4.00, 3.00)
Ann: town 6	(3.00, 4.00)	(3.00, 4.00)	(3.00, 4.00)	(3.50, 3.50)	(2.50, 4.50)	(3.50, 3.50)	(3.50, 3.50)
Ann: town 7	(2.50, 4.50)	(2.50, 4.50)	(3.00, 4.00)	(3.50, 3.50)	(3.00, 4.00)	(3.50, 3.50)	(3.50, 3.50)

Рисунок 3.5 Матриця виграшів локаційної гри лікарів

Для кращої інтерпретації матриці виграшів у програмному середовищі будуються теплові карти окремо для першого та другого гравця. Таке подання є зручним, оскільки у великих матрицях числові значення складніше аналізувати безпосередньо. Теплова карта дозволяє швидко побачити області з більшими та меншими виграшами й оцінити, які локації є вигіднішими для кожного гравця залежно від вибору суперника.

На рисунках 3.6 і 3.7 видно, що структура виграшів двох гравців є взаємопов'язаною. Це пояснюється симетричністю постановки задачі: обидва лікарі мають однаковий набір стратегій і діють за однаковими правилами. Водночас конкретні значення виграшів залежать від поєднання виборів обох гравців. Теплові карти дозволяють побачити, що окремі локації частіше приводять до вищих виграшів, тоді як інші мають менш вигідне положення в мережі.

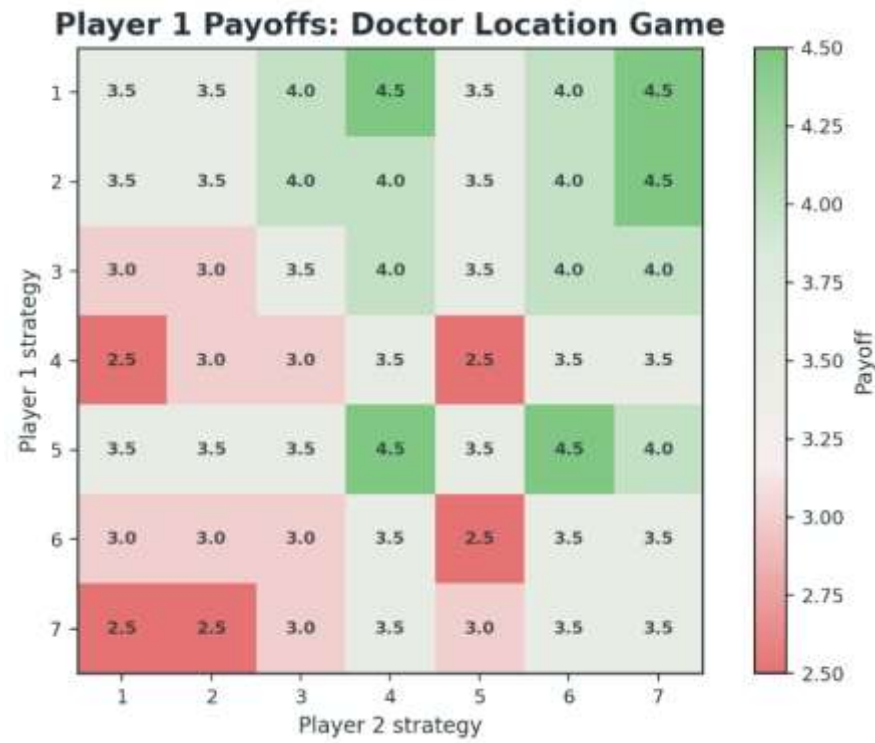


Рисунок 3.6 Теплова карта виграшів першого гравця для локаційної гри лікарів

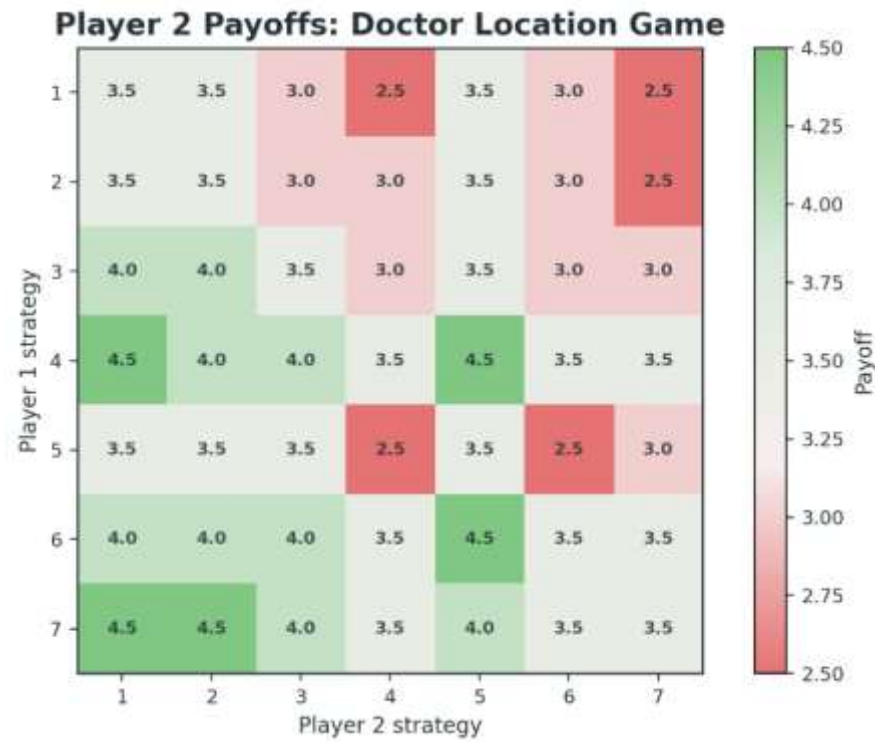


Рисунок 3.7 Теплова карта виграшів другого гравця для локаційної гри лікарів

Наступним етапом є визначення найкращих відповідей гравців. Для першого гравця програмне середовище аналізує кожен фіксований вибір другого гравця та визначає, яка локація забезпечує першому гравцеві найбільший виграш. Аналогічно для другого гравця визначається найкраща відповідь на кожну можливу локацію першого гравця. Результати цього етапу наведено на рисунку 3.8.

Best Responses

	Player	Opponent strategy	Best responses
0	Ann	Beth town 1	{1, 2, 5}
1	Ann	Beth town 2	{1, 2, 5}
2	Ann	Beth town 3	{1, 2}
3	Ann	Beth town 4	{1, 5}
4	Ann	Beth town 5	{1, 2, 3, 5}
5	Ann	Beth town 6	{5}
6	Ann	Beth town 7	{1, 2}
7	Beth	Ann town 1	{1, 2, 5}
8	Beth	Ann town 2	{1, 2, 5}
9	Beth	Ann town 3	{1, 2}

Рисунок 3.8 Таблиця найкращих відповідей для локаційної гри лікарів

Таблиця найкращих відповідей дозволяє перейти від простого перегляду виграшів до аналізу раціональної поведінки гравців. Якщо для певної локації суперника існує кілька однаково вигідних відповідей, усі вони можуть бути зафіксовані як найкращі. У локаційній грі це є змістовним, оскільки декілька різних позицій можуть забезпечувати однакову або близьку кількість мешканців залежно від просторової структури графа.

На основі найкращих відповідей визначаються чисті рівноваги Неша, що показано на рисунку 3.9. Чистою рівновагою в цьому випадку є така пара локацій, за якої жоден із лікарів не може покращити свій виграш шляхом односторонньої зміни власної позиції. Іншими словами, якщо один лікар залишається на своїй локації, другий не має вигіднішого переходу до іншої вершини, і навпаки.

Pure Nash Equilibria

	Pure Nash equilibrium
0	(1, 1)
1	(1, 2)
2	(1, 5)
3	(2, 1)
4	(2, 2)
5	(2, 5)
6	(5, 1)
7	(5, 2)
8	(5, 5)

Рисунок 3.9 Чисті рівноваги Неша для локаційної гри лікарів

Для локаційної гри лікарів програмне середовище визначило дев'ять чистих рівноваг Неша:

$(1,1), (1,2), (1,5), (2,1), (2,2), (2,5), (5,1), (5,2), (5,5)$.

Усі знайдені рівноваги містять стратегії з індексами 1, 2 або 5. Це свідчить про те, що саме ці локації мають найбільше значення в межах заданої просторової структури. Вони можуть бути інтерпретовані як стратегічно вигідні точки розміщення, оскільки за їх вибору жоден із гравців не може отримати кращий результат шляхом одностороннього переходу до іншої локації.

Наявність кількох чистих рівноваг Неша є важливою властивістю локаційної гри лікарів. На відміну від ігор з єдиним стійким профілем, тут виникає множина можливих стабільних результатів. Це означає, що стратегічна структура гри допускає кілька варіантів розміщення лікарів, які є стійкими з погляду індивідуальних відхилень. Такий результат є змістовним для прикладної інтерпретації, оскільки в задачах просторового розміщення часто існує не одна, а декілька близьких за ефективністю конфігурацій.

Після визначення рівноважних характеристик будується граф відповідей (рисунок 3.10). У цьому графі вершини відповідають стратегічним профілям гри, а напрямлені ребра показують можливі односторонні покращувальні переходи гравців. Якщо один із лікарів може змінити свою локацію за фіксованої позиції іншого лікаря та отримати більший виграш, між відповідними профілями проводиться напрямлене ребро.

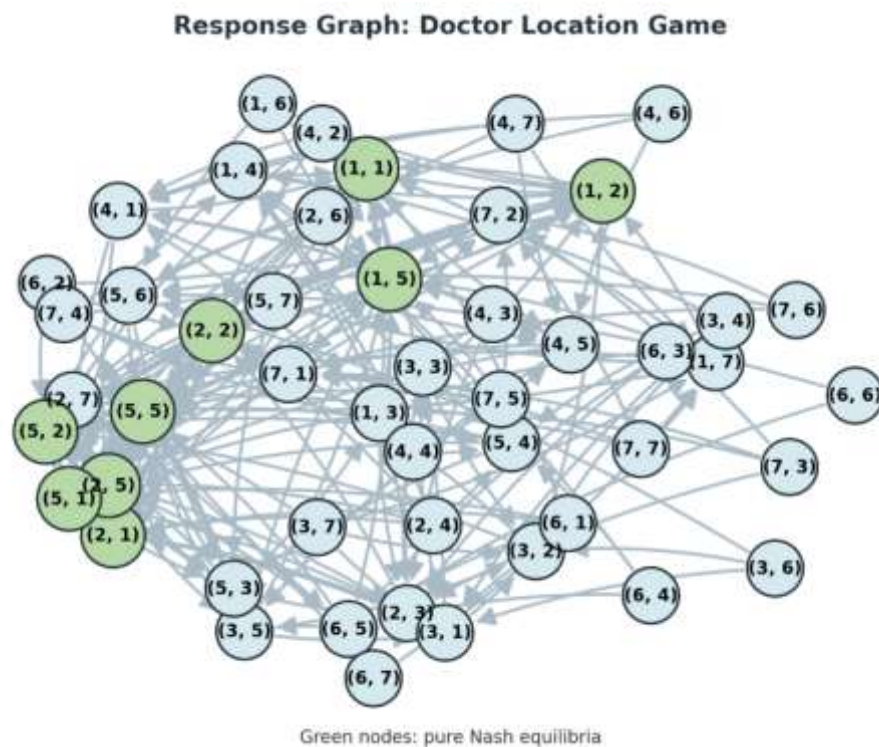


Рисунок 3.10 Граф відповідей для локаційної гри лікарів

Для локаційної гри лікарів граф відповідей містить 49 вершин і 204 ребра. Кількість вершин відповідає кількості стратегічних профілів, оскільки кожен із двох гравців має по сім стратегій. Кількість ребер показує, скільки односторонніх покращувальних переходів існує між профілями. Значення 204 свідчить про достатньо насичену структуру стратегічних переходів: з багатьох профілів один або обидва гравці можуть змінити своє місце розташування та отримати кращий результат.

Отриманий граф відповідей демонструє, що гра не зводиться до простого спрямованого руху до одного стійкого профілю. Попри наявність дев'яти чистих рівноваг Неша, у графі існує значна кількість переходів між

нестійкими станами. Це означає, що локальна динаміка покращень у грі є складною: різні стратегічні профілі можуть вести до різних областей графа, а не лише до одного центру тяжіння.

Основні характеристики графа відповідей наведено на рисунку 3.11.

Response Graph Statistics

	Metric	Value
0	Number of nodes	49
1	Number of edges	204
2	Strongly connected components	31
3	Cycles	50914

Рисунок 3.11 Основні характеристики графа відповідей для локаційної гри лікарів

За результатами аналізу граф відповідей має 31 сильно зв'язану компоненту. Це означає, що граф складається з кількох структурних частин, між якими переходи організовані нерівномірно. Частина профілів утворює взаємопов'язані області, у яких можливі переходи між станами, тоді як інші профілі мають більш ізольовану або периферійну роль у загальній структурі графа.

Особливо важливою характеристикою є велика кількість циклів. Для локаційної гри лікарів програмне середовище визначило 50914 циклів. Такий результат свідчить про те, що послідовність односторонніх покращень у грі не завжди приводить до остаточного стійкого профілю. У частині випадків гравці можуть послідовно змінювати локації, покращуючи власний виграш, але при цьому система може повертатися до вже пройдених станів. Це є ознакою нетранзитивності та складної структури стратегічної взаємодії.

Розподіл циклів подано на рисунку 3.12.

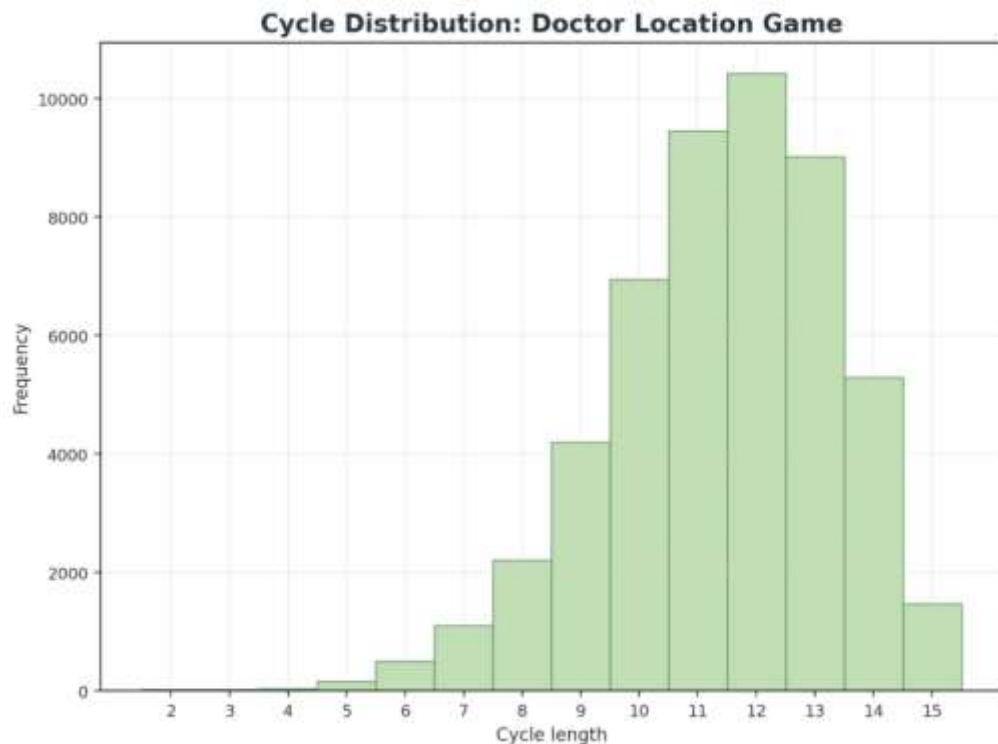


Рисунок 3.12 Розподіл циклів у графі відповідей локаційної гри лікарів

Розподіл циклів дозволяє оцінити не лише сам факт наявності циклічності, а й характер цієї циклічності. Якщо у графі переважають короткі цикли, це може свідчити про локальні взаємні покращення між невеликими групами профілів. Якщо ж є цикли більшої довжини, це вказує на складніші траєкторії переходів у просторі стратегічних профілів. Для локаційної гри лікарів велика кількість циклів є змістовною, оскільки зміна місця розташування одного лікаря може створювати стимули для зміни позиції іншого гравця, а потім знову змінювати вигідність попередніх локацій.

Наступним етапом аналізу є спектральне подання графа відповідей. Метою цього подання є розміщення стратегічних профілів у просторі меншої розмірності з урахуванням структури зв'язків між ними. На відміну від звичайного графа відповідей, спектральне подання дозволяє краще побачити групування профілів і відокремлення периферійних станів.

На рисунку 3.13 видно, що більшість стратегічних профілів зосереджена в одній щільній області. Саме в цій частині розташовано багато взаємопов'язаних профілів, між якими існують численні переходи. Водночас

окремі профілі, зокрема профіль (6,6), розміщені помітно віддалено від основної групи. Це свідчить про те, що деякі варіанти розташування лікарів мають периферійну роль у структурі гри та відрізняються від більшості інших профілів за характером зв'язків.

Зеленим кольором на спектральному графі позначено чисті рівноваги Неша. Їх розташування в центральній частині графа показує, що рівноважні профілі пов'язані зі стратегічно важливими локаціями. Це узгоджується зі змістовною інтерпретацією гри: локації 1, 2 і 5 формують кілька стійких конфігурацій, які є вигідними для обох гравців з погляду односторонніх відхилень.

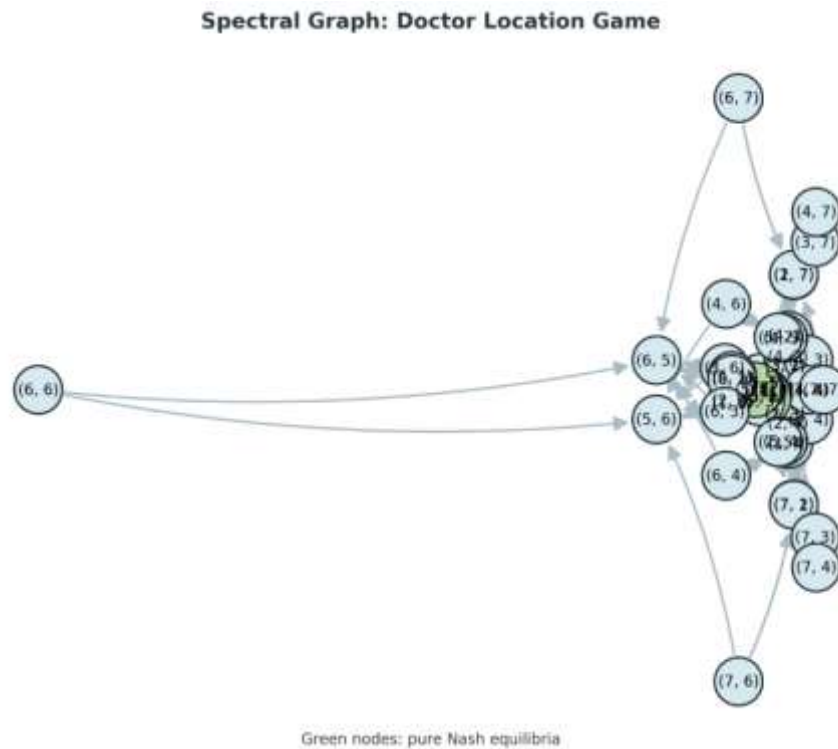


Рисунок 3.13 Спектральне подання графа відповідей локаційної гри лікарів

Для подальшого узагальнення структури гри було виконано кластеризацію стратегічних профілів. Кластеризація дозволяє розділити множину профілів на групи, які мають подібну роль у графі відповідей. Це особливо корисно для ігор із більшою кількістю стратегій, де безпосередній аналіз усіх вершин і ребер може бути складним.

На рисунку 3.14 кольори відповідають різним кластерам, а більший розмір вершин використано для позначення чистих рівноваг Неша. Видно, що значна частина профілів належить до великого центрального кластера, у якому зосереджено основну кількість переходів. Окремі профілі утворюють менші групи, розташовані на периферії графа. Зокрема, профіль (6,6) утворює відокремлену область, що додатково підтверджує його особливе положення у структурі гри.

Кластеризоване подання дозволяє побачити, що стратегічний простір гри не є однорідним. Частина профілів формує щільну центральну зону, пов'язану з основними рівноважними та близькими до них станами. Інші профілі розташовані на периферії та мають менше зв'язків із центральною частиною. Така структура є змістовною для локаційної гри, оскільки положення стратегій у просторовій мережі безпосередньо впливає на виграші гравців.

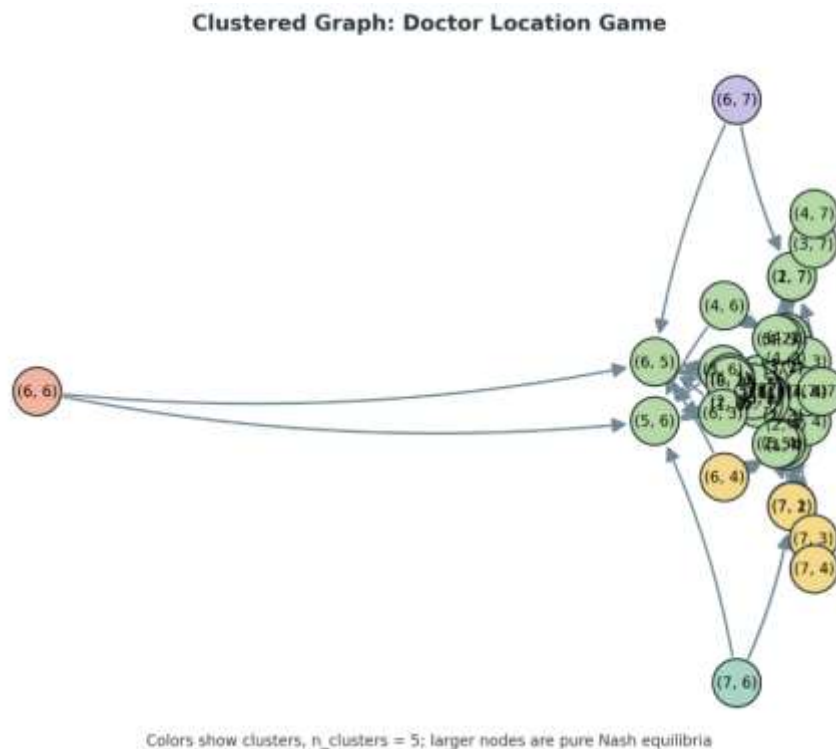


Рисунок 3.14 Кластеризований граф стратегічних профілів локаційної гри лікарів

На основі результатів кластеризації побудовано скорочений граф кластерів (рисунок 3.15). У такому графі кожен кластер подається як окрема вершина, а ребра між кластерами показують наявність переходів між профілями, що належать до різних груп. Це дозволяє перейти від детального графа з 49 вершинами до більш узагальненої схеми взаємодії між групами стратегічних станів.

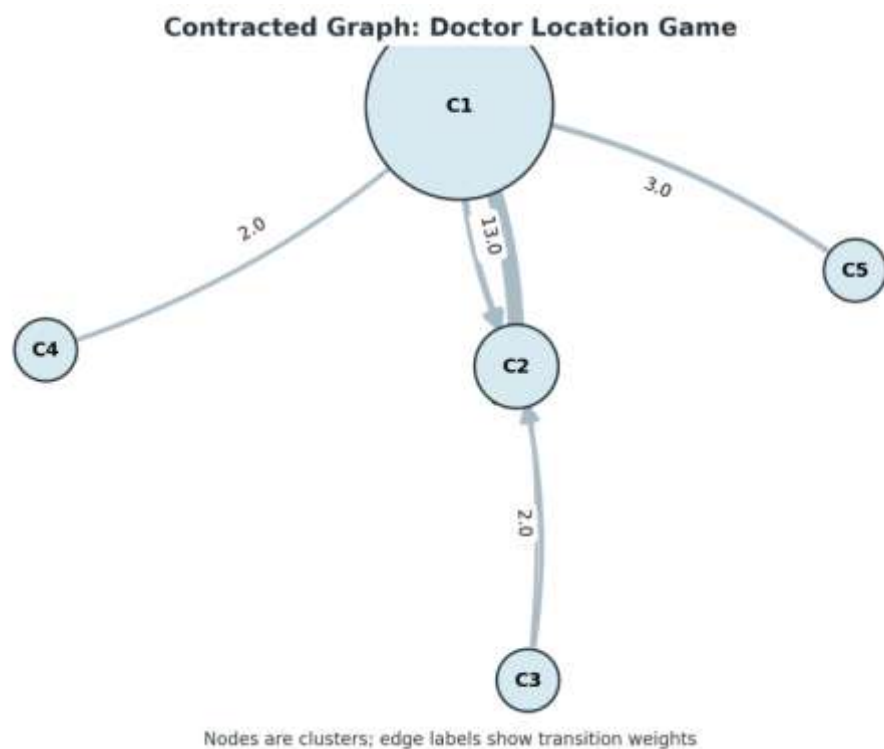


Рисунок 3.15 Скорочений граф кластерів для локаційної гри лікарів

Для локаційної гри лікарів скорочений граф містить 5 вершин, що відповідає п'яти знайденим кластерам. Найбільшим за розміром є кластер C1, який містить найбільшу кількість стратегічних профілів. Між кластерами існують напрямлені переходи з різними вагами. Зокрема, помітним є перехід між кластерами C1 і C2 з вагою 13. Також на графі наявні переходи C4→C1 з вагою 2, C1→C5 з вагою 3 та C3→C2 з вагою 2. Ваги ребер показують кількість переходів між профілями, що належать до відповідних кластерів.

Скорочений граф значно спрощує інтерпретацію початкового графа відповідей. Замість 49 окремих вершин користувач бачить п'ять узагальнених груп і переходи між ними. Це дозволяє оцінити не лише локальні зв'язки між

окремими профілями, а й загальну структуру руху між групами стратегічних станів. Такий підхід є важливим для візуального аналізу, оскільки дає змогу зменшити складність графа без повної втрати інформації про напрямки переходів.

Останнім етапом аналізу окремої гри є наближене ранжування стратегічних профілів (рисунок 3.16). У програмному середовищі воно використовується як додатковий інструмент оцінювання відносної важливості профілів у структурі гри. Таке ранжування методологічно пов'язане з ідеями α -Rank, однак у межах реалізованого середовища використовується як наближена оцінка значущості профілів, а не як повна реалізація класичного методу.

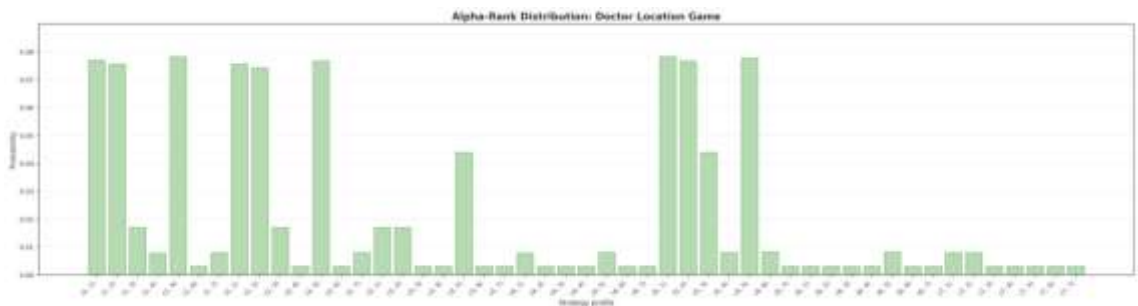


Рисунок 3.16 Наближене ранжування стратегічних профілів для локаційної гри лікарів

Ранжування дозволяє доповнити аналіз рівноваг Неша. Рівновага Неша показує, чи є профіль стійким щодо односторонніх відхилень, але не завжди дає повне уявлення про роль інших профілів у структурі гри. Наближене ранжування дозволяє оцінити, які профілі мають більшу відносну вагу з погляду структури переходів. Якщо певний профіль має високий ранг, це може свідчити про його важливе положення у графі відповідей або про те, що він часто пов'язаний із переходами з інших станів.

Отримані результати показують, що локаційна гра лікарів має складну стратегічну структуру. У ній існує дев'ять чистих рівноваг Неша, граф відповідей містить 49 вершин і 204 ребра, а також має 31 сильно зв'язану компоненту та велику кількість циклів. Це свідчить про те, що гра не зводиться

до простого руху до одного стабільного профілю. Натомість вона містить кілька стійких конфігурацій і складну систему переходів між нестійкими станами.

Таким чином, аналіз локаційної гри лікарів демонструє можливості розробленого програмного середовища для дослідження окремої гри в нормальній формі. Система дозволяє перейти від матриці виграшів до повного набору структурних і візуальних представлень: теплових карт, таблиці найкращих відповідей, рівноваг Неша, графа відповідей, статистик графа, розподілу циклів, спектрального подання, кластеризованого графа, скороченого графа кластерів і наближеного ранжування профілів. Це підтверджує, що запропоноване середовище може використовуватися не лише для обчислення окремих характеристик гри, а й для комплексної інтерпретації її стратегічної структури.

3.5 Побудова ландшафту ігор за структурними характеристиками

Окрім детального аналізу окремої гри, у розробленому програмному середовищі реалізовано режим Games Landscape, призначений для порівняння множини ігор між собою. Якщо в режимі Single Game Analysis основна увага приділяється одній вибраній грі, її матриці виграшів, графу відповідей, рівновагам Неша, циклам і кластерам стратегічних профілів, то в режимі ландшафту кожна гра розглядається як окремий об'єкт із набором числових характеристик. Це дозволяє перейти від аналізу однієї гри до порівняння різних типів стратегічної взаємодії.

Ідея побудови ландшафту ігор полягає в тому, що гра описується не безпосередньо матрицею виграшів, а набором структурних ознак, отриманих у результаті її аналізу. У дослідженні ландшафту ігор запропоновано підхід, за

якого ігри порівнюються на основі характеристик графового подання, після чого розташовуються у спільному просторі ознак [1]. У розробленому програмному середовищі використано подібну логіку: для кожної гри будується граф відповідей, обчислюються його структурні характеристики, після чого множина ігор відображається у двовимірному або тривимірному просторі за допомогою методів зниження розмірності.

У режимі Games Landscape користувач може обрати набір ігор для порівняння, метод зниження розмірності, кількість кластерів і тип візуалізації. У програмі передбачено два основні методи побудови проєкції: PCA та UMAP. PCA використовується для побудови лінійної проєкції з можливістю інтерпретації внеску окремих ознак у головні компоненти. UMAP використовується як додатковий нелінійний метод, який дає змогу краще зберігати локальні взаємозв'язки між об'єктами у просторі ознак [13].

На рисунку 3.17 показано інтерфейс режиму побудови ландшафту ігор. У цьому режимі користувач може сформувати набір ігор для порівняння, обрати двовимірне або тривимірне подання, задати метод зниження розмірності та кількість кластерів. Такий підхід робить ландшафт не лише статичною візуалізацією, а інтерактивним інструментом аналізу, за допомогою якого можна порівнювати класичні, синтетичні, прикладні та емпірично подані OpenSpiel-ігри.

Кінець таблиці 3.3

mean_out_degree	середній вихідний степінь вершин графа відповідей
max_in_degree	максимальний вхідний степінь вершини
max_out_degree	максимальний вихідний степінь вершини
pure_nash_equilibria	кількість чистих рівноваг Неша
strongly_connected_components	кількість сильно зв'язаних компонент графа відповідей
cycles	кількість циклів у графі відповідей
alpha_rank_entropy	ентропія наближеного ранжування стратегічних профілів
normalized_edges	нормалізована кількість ребер графа відповідей
normalized_cycles	нормалізована кількість циклів
normalized_scc	нормалізована кількість сильно зв'язаних компонент
normalized_nash	нормалізована кількість чистих рівноваг Неша

Наведені ознаки дозволяють порівнювати ігри не лише за розмірністю матриці виграшів, а й за внутрішньою структурою стратегічних переходів. Дві гри можуть мати однакову кількість стратегій, але суттєво відрізнятися за кількістю циклів, щільністю графа відповідей, кількістю рівноваг Неша або кількістю сильно зв'язаних компонент. Саме тому використання структурних характеристик є більш інформативним, ніж порівняння ігор лише за розміром матриці.

Після обчислення ознак формується таблиця структурних характеристик усіх ігор, що беруть участь у побудові ландшафту. Кожен рядок такої таблиці

відповідає окремій грі, а кожен стовпець – певній метриці. Така таблиця є проміжним результатом між аналізом окремих ігор і побудовою їхнього спільного візуального простору.

На рисунку 3.18 показано таблицю ознак, сформовану програмним середовищем. Вона дозволяє не лише побудувати ландшафт, а й перевірити числові значення характеристик для кожної гри. Наприклад, за цією таблицею можна визначити, які ігри мають найбільшу кількість стратегічних профілів, які характеризуються найбільшою кількістю циклів, які мають найбільшу щільність графа відповідей або більшу кількість рівноваг Неша.

Feature Table

	game	rows	columns	profiles	responses	density	mean ₁	mean ₂	max ₁	max ₂	pure ₁	strong ₁	cycles	alpha ₁	normal ₁	normal ₂	normal ₃	normal ₄
0	Prisoner's Dilemma	2	2	4	4	0.333	1	1	2	2	1	4	0	1.266	1	0	1	0.25
1	Matching Pennies	2	2	4	4	0.333	1	1	1	1	0	1	1	1.386	1	0.25	0.25	0
2	Battle of the Sexes	2	2	4	4	0.333	1	1	2	2	2	4	0	1.341	1	0	1	0.5
3	Rock-Paper-Scissors	3	3	9	12	0.167	1.333	1.333	2	2	0	4	1	1.956	1.333	0.111	0.444	0
4	Hawk-Dove Game	2	2	4	4	0.333	1	1	2	2	2	4	0	1.341	1	0	1	0.5
5	Transitive 6x6 Game	6	6	36	160	0.127	4.444	4.444	10	10	1	16	220	3.366	4.444	6.111	0.444	0.028
6	Cyclical 6x6 Game	6	6	36	60	0.048	1.667	1.667	5	2	0	26	2	2.879	1.667	0.056	0.722	0
7	11-20 Game	10	10	100	180	0.018	1.8	1.8	9	2	0	82	2	3.529	1.8	0.02	0.82	0
8	Random 5x5 Game	5	5	25	44	0.073	1.76	1.76	8	3	2	22	2	2.428	1.76	0.06	0.88	0.06
9	Zero-Sum 5x5 Game	5	5	25	40	0.067	1.6	1.6	4	2	0	22	1	2.271	1.6	0.04	0.88	0

Рисунок 3.18 Таблиця структурних характеристик ігор

Перед застосуванням методів зниження розмірності ознаки стандартизуються. Це необхідно тому, що різні характеристики мають різні масштаби. Наприклад, кількість циклів може вимірюватися сотнями або тисячами, тоді як щільність графа набуває значень від 0 до 1. Якщо не виконати стандартизацію, ознаки з більшими числовими значеннями можуть надмірно впливати на результат проєкції. Стандартизація дозволяє привести ознаки до порівнянного масштабу та забезпечити більш коректне формування простору ігор.

Після попередньої обробки до таблиці ознак застосовується метод головних компонент. Для побудови двовимірного PCA-ландшафту використовується проєкція на перші дві головні компоненти, а для

тривимірного – на перші три. У такому поданні кожна гра відображається як точка, координати якої визначаються значеннями відповідних головних компонент.

На рисунку 3.19 наведено двовимірний PCA-ландшафт ігор. Кожна точка на площині відповідає окремій грі з обраного набору. Близьке розташування точок означає, що відповідні ігри мають подібні структурні характеристики. Водночас віддалене розташування вказує на відмінності у структурі графа відповідей, кількості стратегічних профілів, наявності циклів, рівноваг або інших ознак.

На такому ландшафті можна візуально оцінити групи ігор зі схожою стратегічною структурою. Прості ігри розмірності 2×2 зазвичай розташовуються ближче одна до одної, оскільки мають невелику кількість профілів і відносно просту структуру графа відповідей. Ігри з більшою кількістю стратегій, значною кількістю циклів або складнішою системою переходів можуть віддалятися від них у просторі ознак. Таким чином, PCA-ландшафт дозволяє не лише побачити відмінності між окремими іграми, а й виявити групи структурно близьких прикладів.

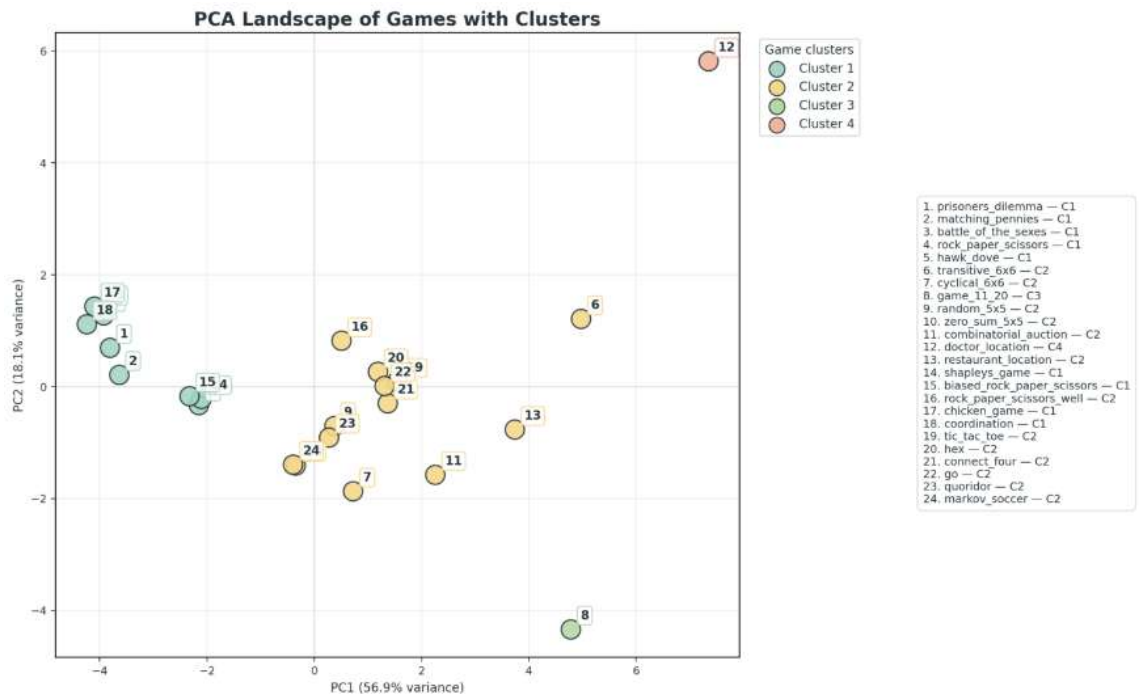


Рисунок 3.19 Двовимірний РСА-ландшафт ігор

За результатами побудови двовимірного РСА-ландшафту можна зробити висновок, що ігри розміщуються у просторі ознак не випадково, а відповідно до їхніх структурних характеристик. Перші дві головні компоненти пояснюють значну частину варіації даних: перша компонента PC1 пояснює 56,9 %, а друга компонента PC2 – 18,1 %. Разом вони відображають близько 75 % загальної мінливості ознак, тому двовимірне подання є достатньо інформативним для попереднього аналізу подібності між іграми.

На двовимірному РСА-ландшафті прості класичні ігри, зокрема дилема в'язня, гра «орлянка-решка», битва статей, камінь-ножиці-папір, яструб-голуб, chicken game та координаційна гра, утворюють окрему компактну область у лівій частині графіка. Це пояснюється тим, що такі ігри мають невелику кількість стратегій, обмежену кількість стратегічних профілів і порівняно просту структуру графа відповідей.

Більшість складніших ігор потрапляє до другого кластера. До нього належать транзитивна гра 6×6 (transitive_6x6), циклічна гра 6×6 (cyclic_6x6), випадкова гра 5×5 (random_5x5), гра з нульовою сумою 5×5 (zero_sum_5x5),

комбінаторний аукціон (`combinatorial_auction`), локаційна гра ресторану (`restaurant_location`), хрестики-нулики (`tic_tac_toe`), гекс (`hex`), чотири в ряд (`connect_four`), го (`go`), коридор (`quoridor`) та марковський футбол (`markov_soccer`). Їхнє розташування ближче до центральної та правої частини РСА-простору пов'язане з більшою кількістю стратегій, більшою кількістю стратегічних профілів і складнішою системою покращувальних переходів у графі відповідей.

Окремими випадками є гра 11–20 (`game_11_20`) та локаційна гра лікарів (`doctor_location`). Гра 11–20 утворює третій кластер і помітно віддаляється від основної групи за другою головною компонентою. Локаційна гра лікарів утворює четвертий кластер і є найбільш віддаленою точкою на РСА-ландшафті. Таке розташування пояснюється великою кількістю стратегічних профілів, прикладною природою моделі та складнішою структурою графа відповідей порівняно з базовими матричними іграми.

Для детальнішого аналізу у програмному середовищі передбачено тривимірне подання РСА-ландшафту. Воно використовує три перші головні компоненти та дозволяє частково зберегти більше інформації про варіативність даних.

На рисунку 3.20 наведено тривимірний РСА-ландшафт. Таке подання є корисним у випадках, коли перші дві головні компоненти не пояснюють достатньо повно всю структуру відмінностей між іграми. Додавання третьої компоненти дозволяє побачити додатковий напрям варіативності та краще відокремити ігри, які у двовимірній проєкції можуть накладатися одна на одну або розташовуватися надто близько.

3D PCA Landscape of Games with Clusters

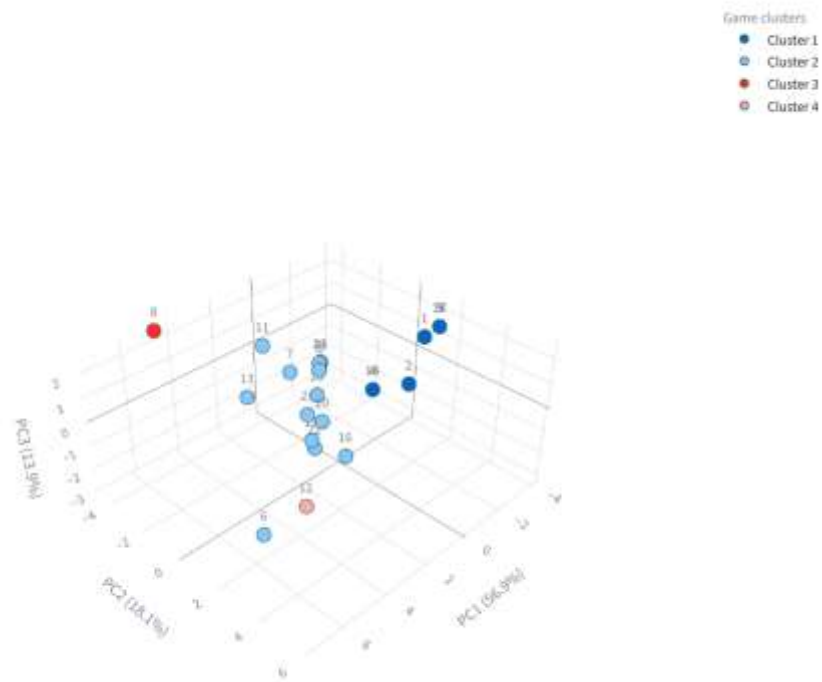


Рисунок 3.20 Тривимірний PCA-ландшафт ігор

Тривимірне PCA-подання доповнює результати двовимірного аналізу, оскільки третя головна компонента пояснює ще 13,9 % варіації. У сумі перші три компоненти відображають близько 88,9 % загальної мінливості даних. Це дозволяє точніше побачити відокремлення окремих ігор і підтверджує, що локаційна гра лікарів (*doctor_location*) та гра 11–20 (*game_11_20*) мають нетипове положення відносно більшості інших ігор. Таким чином, тривимірна візуалізація є корисною для уточнення просторової структури ландшафту, особливо у випадках, коли частина точок накладається або розташовується близько у двовимірній проєкції.

Важливою перевагою PCA є можливість оцінити пояснену дисперсію. У програмному середовищі для PCA виводиться частка дисперсії, яку пояснюють перші головні компоненти. Якщо перші дві або три компоненти пояснюють значну частину варіативності, то побудований ландшафт можна використовувати для змістовної інтерпретації подібності між іграми. Якщо ж частка поясненої дисперсії є невеликою, результати слід інтерпретувати обережніше, оскільки частина інформації залишається в інших компонентах.

Крім координат ігор у просторі головних компонент, у застосунку виводиться таблиця впливу структурних метрик на головні компоненти. Вона показує, які саме характеристики найбільше впливають на розташування ігор уздовж осей PC1, PC2 або PC3.

На рисунку 3.21 показано внесок окремих ознак у формування головних компонент. Такий результат є важливим для інтерпретації ландшафту, оскільки дозволяє зрозуміти, які саме властивості ігор визначають їхнє розташування. Наприклад, одна компонента може бути пов'язана переважно з розміром гри та кількістю стратегічних профілів, інша – з кількістю циклів, щільністю графа відповідей або кількістю сильно зв'язаних компонент. Завдяки цьому користувач може не лише побачити, які ігри розташовані поруч, а й пояснити, за рахунок яких характеристик формується така близькість.

PCA Metric Influence

	Metric	PC1 coefficient	PC1 influence (%)	PC2 coefficient	PC2 influence (%)	PC3 coefficient	PC3 influence (%)	Total influence in PCA space
0	columns	0.2912	8.4771	-0.2134	4.555	0.121	1.4633	6.2
1	rows	0.2912	8.4771	-0.2134	4.555	0.121	1.4633	6.2
2	strongly_connected_c	0.2272	5.1642	-0.2833	8.0268	0.3091	9.5571	6.4
3	mean_out_degree	0.2775	7.702	0.1919	3.6813	-0.2134	4.5545	6.3
4	mean_in_degree	0.2775	7.702	0.1919	3.6813	-0.2134	4.5545	6.3
5	normalized_edges	0.2775	7.702	0.1919	3.6813	-0.2134	4.5545	6.3
6	response_edges	0.309	9.5491	0.0256	0.0654	0.0985	0.97	6.2
7	profiles	0.2655	7.0498	-0.2305	5.3133	0.2086	4.3508	6.2
8	alpha_rank_entropy	0.2977	8.8631	-0.1235	1.5256	-0.0876	0.7674	6.3
9	pure_nash_equilibria	0.1428	2.0378	0.3943	15.5486	0.3222	10.3808	6.0

Рисунок 3.21 Вплив структурних метрик на головні компоненти PCA

Для доповнення PCA-проекції у програмному середовищі реалізовано побудову UMAP-ландшафту. UMAP є нелінійним методом зниження розмірності, орієнтованим на збереження локальної структури даних [13]. У контексті ландшафту ігор це означає, що метод може допомогти виявити групи ігор, які мають близькі структурні характеристики, навіть якщо ці зв'язки не є лінійними.

На рисунку 3.22 наведено UMAP-ландшафт ігор. На відміну від PCA, координати UMAP не мають прямої інтерпретації через початкові ознаки. Тому UMAP доцільно використовувати як додатковий інструмент візуального аналізу. Якщо групи ігор, які є близькими на PCA-ландшафті, також залишаються близькими в UMAP-проекції, це підсилює висновок про їхню структурну подібність. Якщо ж розташування суттєво відрізняється, це може свідчити про наявність складніших нелінійних зв'язків між ознаками.

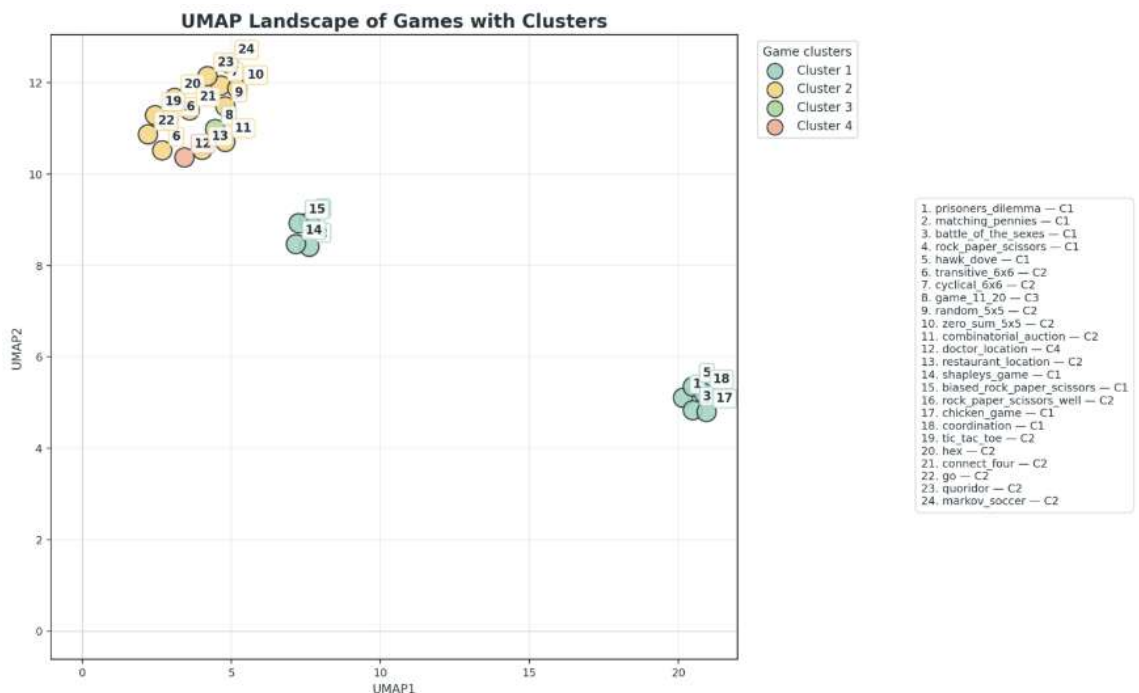


Рисунок 3.22 UMAP-ландшафт ігор

UMAP-ландшафт підтверджує загальну структуру групування, але подає її у більш локально вираженому вигляді. На ньому чітко відокремлюється група простих класичних ігор, група складніших ігор другого кластера, а також окрема група, до якої належать гра Шеплі (Shapley's game) та зміщена версія гри «камінь-ножиці-папір» (biased rock-paper-scissors). На відміну від PCA, метод UMAP краще зберігає локальну близькість між об'єктами, тому він корисний для виявлення невеликих груп ігор зі схожою структурою. Отже, спільне використання PCA та UMAP дозволяє проаналізувати як глобальну структуру простору ігор, так і локальні групи подібності між окремими іграми.

У режимі ландшафту також реалізовано кластеризацію ігор. Для цього використовується алгоритм KMeans, який групує ігри за їхніми стандартизованими характеристиками. Користувач може задати кількість кластерів або використати автоматичний вибір. У візуалізації точки, що відповідають іграм, забарвлюються відповідно до належності до певного кластера. Це дозволяє швидко побачити, які ігри утворюють спільні групи.

Кластеризація є корисною для попередньої класифікації ігор. Якщо в один кластер потрапляють ігри зі схожою структурою, це може свідчити про спільність їхніх структурних властивостей. Наприклад, в окремі групи можуть потрапляти прості ігри розмірності 2×2 , циклічні ігри, синтетичні ігри більшої розмірності або прикладні локаційні ігри. При цьому результати кластеризації не слід розглядати як остаточну класифікацію, оскільки вони залежать від набору ознак, способу масштабування, кількості кластерів і методу зниження розмірності.

Окремою функцією режиму Games Landscape є пошук структурно схожих ігор. Для вибраної гри програма обчислює відстані до інших ігор у просторі ознак і формує список найближчих прикладів. Такий підхід дозволяє порівнювати нову або користувацьку гру з уже наявним набором і визначати, до яких відомих структур вона є найближчою.

На рисунку 3.23 показано результат пошуку структурно схожих ігор. Ця функція є особливо корисною для аналізу користувацьких ігор, які можуть бути додані через режим Custom Game Input. Після збереження такої гри вона може бути включена до ландшафту, а система може визначити, до яких із наявних прикладів вона є найближчою за структурними характеристиками. Це дає змогу не лише аналізувати гру ізольовано, а й оцінювати її місце серед інших стратегічних середовищ.

Similar Games

This block finds games that are structurally similar to the selected game. Similarity is calculated using standardized graph-based metrics, not by game names. Lower structural distance means higher similarity.

Choose a game to find similar games

doctor_location

Number of similar games to show

5

Rank	Similar game	Game name	Game type	Cluster	Structural distance	similarity_score (%)
1	restaurant_location	Restaurant Location Game	location game	2	8.4409	10.5922
2	transitive_6x6	Transitive 6x6 Game	transitive game	2	8.6312	10.3829
3	hex	Hex Empirical Game	empirical OpenSpiel game	2	9.2714	9.7358
4	tic_tac_toe	Tic-Tac-Toe Empirical Game	empirical OpenSpiel game	2	9.3197	9.6902
5	connect_four	Connect Four Empirical Gar	empirical OpenSpiel game	2	9.437	9.5813

The closest game to doctor_location is restaurant_location with structural distance 8.4409.

Рисунок 3.23 Пошук структурно схожих ігор у просторі ознак

Практичне значення побудови ландшафту полягає в тому, що він поєднує результати аналізу багатьох ігор в одному візуальному просторі. Якщо аналіз окремої гри дає змогу зрозуміти її внутрішню структуру, то ландшафт дозволяє порівняти цю структуру з іншими іграми. У результаті користувач може визначити, які ігри є структурно близькими, які суттєво відрізняються, які утворюють окремі групи, а які мають нетипові характеристики.

Важливо зазначити, що ландшафт ігор не є доказом повної еквівалентності або відмінності ігор. Близьке розташування точок означає подібність за обраним набором структурних ознак, але не гарантує однаковості всіх стратегічних властивостей. Водночас віддалене розташування свідчить про відмінність у межах використаних характеристик, але також залежить від вибору ознак, методу масштабування та методу зниження розмірності. Тому результати ландшафту доцільно інтерпретувати разом із аналізом окремих ігор, їхніх таблиць виграшів, графів відповідей і структурних характеристик.

У межах цієї роботи режим Games Landscape виконує роль підсумкового інструмента аналітико-візуалізаційного етапу емпіричного аналізу. Він

об'єднує результати, отримані на попередніх етапах: таблиці виграшів, графи відповідей, структурні характеристики, ранжування профілів і методи зниження розмірності. Завдяки цьому програмне середовище дозволяє не лише дослідити окрему гру, а й порівняти її з іншими іграми у спільному просторі ознак.

3.6 Висновки до розділу 3

У третьому розділі було розглянуто розроблене програмне середовище для аналізу ігор у нормальній формі, його функціональні можливості, архітектуру та результати застосування до різних типів ігор. Показано, що система забезпечує аналітико-візуалізаційний етап емпіричного аналізу, тобто працює з уже сформованими таблицями виграшів, виконує їх обробку, будує графові представлення, обчислює структурні характеристики та формує візуальні результати для подальшої інтерпретації.

Було описано основні режими роботи програмного середовища: аналіз окремої гри, введення користувацької гри та побудову ландшафту ігор. Розроблена архітектура має модульну структуру, що дозволяє відокремити модель гри, аналітичні обчислення, побудову графів, формування візуалізацій і порівняння ігор у спільному просторі ознак. Такий підхід спрощує подальше розширення системи, зокрема додавання нових ігор, структурних характеристик і методів візуального аналізу.

Для перевірки роботи програмного середовища було сформовано набір класичних, синтетичних, прикладних та емпірично поданих OpenSpiel-ігор. Такий набір дозволив протестувати систему на іграх різної розмірності та різної стратегічної структури: іграх із чистими рівновагами Неша, іграх без чистих рівноваг, циклічних, транзитивних, випадкових, антагоністичних і

прикладних моделях. Це дало змогу перевірити не лише коректність окремих алгоритмів, а й здатність середовища виявляти структурні відмінності між іграми.

На прикладі локаційної гри лікарів було продемонстровано повний процес аналізу окремої гри у програмному середовищі. Для цієї гри було побудовано матрицю виграшів, теплові карти виграшів гравців, таблицю найкращих відповідей, визначено чисті рівноваги Неша, побудовано граф відповідей, обчислено його структурні характеристики, проаналізовано цикли, спектральне подання, кластеризований граф, скорочений граф кластерів і наближене ранжування стратегічних профілів. Отримані результати показали, що навіть одна прикладна гра може мати складну внутрішню структуру, яку важко повністю інтерпретувати лише за матрицею виграшів.

Окремо було розглянуто побудову ландшафту ігор за структурними характеристиками. Для кожної гри формується вектор ознак, що описує її розмір, граф відповідей, кількість рівноваг, циклів, сильно зв'язаних компонент та інші характеристики. Після стандартизації ознак застосовуються методи зниження розмірності, зокрема PCA та UMAP, що дозволяє відобразити множину ігор у спільному двовимірному або тривимірному просторі. Такий підхід дає змогу порівнювати ігри між собою, виявляти групи структурно подібних прикладів і визначати нетипові ігри.

Таким чином, результати третього розділу підтверджують, що розроблене програмне середовище дозволяє перейти від табличного подання гри до комплексного структурного й візуального аналізу. Система не обмежується пошуком окремих рівноважних станів, а дає змогу досліджувати граф відповідей, циклічність, зв'язність, кластери стратегічних профілів і місце гри серед інших ігор у ландшафті. Це підтверджує доцільність використання запропонованого середовища для аналізу, інтерпретації та порівняння ігор у нормальній формі.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі буде проведено функціонально-вартісний аналіз програмного продукту, розробленого для емпіричного аналізу ігор у нормальній формі. Розроблюване візуальне середовище призначене для введення та обробки матриць виграшів, аналізу стратегічної взаємодії гравців, пошуку рівноваг Неша, побудови графів відповідей, проведення спектрального аналізу, кластеризації та візуального порівняння ігор за допомогою ландшафту ігор.

Функціонально-вартісний аналіз застосовується для оцінювання доцільності прийнятих проєктних рішень, визначення основних функцій програмного продукту та порівняння можливих варіантів їх реалізації. Такий підхід дозволяє врахувати не лише технічну ефективність системи, а й витрати, пов'язані з її розробкою, підтримкою та подальшим використанням.

У межах аналізу буде визначено основні функції програмного продукту, сформовано варіанти їх реалізації, обґрунтовано систему параметрів для оцінювання якості, проведено експертне оцінювання їх вагомості та виконано економічний аналіз альтернативних варіантів розробки. На основі отриманих результатів буде обрано найбільш раціональний варіант реалізації програмного продукту з урахуванням його функціональних можливостей, технічного рівня та вартості створення.

4.1 Постановка задачі проектування

У роботі застосовується метод функціонально-вартісного аналізу для проведення техніко-економічного аналізу розробки програмного продукту для емпіричного аналізу ігор у нормальній формі. Оскільки рішення щодо структури програмного продукту, вибору алгоритмів, способів обробки даних та візуалізації результатів впливають на ефективність усієї системи, необхідно оцінити основні функції майбутнього програмного середовища та визначити найбільш раціональний варіант його реалізації.

Метою даного етапу є створення програмного продукту, який дозволяє автоматизувати процес аналізу ігор у нормальній формі та забезпечує наочне подання отриманих результатів. Програмний продукт повинен надавати інструменти для введення або вибору матриці виграшів, аналізу стратегічної взаємодії гравців, визначення рівноважних станів, побудови графових моделей гри та візуалізації результатів дослідження.

Фактичний аналіз у межах цього розділу представляє собою аналіз функцій програмного продукту, призначеного для обробки матриць виграшів, побудови графа відповідей, пошуку рівноваг Неша, проведення спектрального аналізу, кластеризації стратегічних профілів та формування ландшафту ігор. Розроблюване візуальне середовище має забезпечити зручну роботу користувача з різними типами ігор, а також можливість порівнювати результати аналізу за допомогою графіків, таблиць і діаграм.

Основними функціональними вимогами до програмного продукту є:

1. Можливість введення користувацької гри у нормальній формі через матриці виграшів двох гравців.
2. Підтримка набору попередньо заданих класичних, згенерованих та прикладних ігор.
3. Відображення матриці виграшів у зручному для аналізу вигляді.

4. Визначення чистих рівноваг Неша для заданої гри.
5. Побудова графа відповідей для відображення переходів між стратегічними профілями.
6. Розрахунок показників важливості стратегічних профілів, зокрема за допомогою підходу α -Rank.
7. Проведення спектрального аналізу графа відповідей та кластеризації стратегічних профілів.
8. Побудова ландшафту ігор для порівняння різних ігор за їх структурними характеристиками.
9. Візуалізація результатів аналізу у вигляді графів, діаграм, таблиць і проєкцій.
10. Можливість збереження результатів аналізу для подальшого використання у дослідженні.

4.2 Обґрунтування функцій програмного продукту

Головна функція програмного продукту F_0 – розробка візуального середовища, яке дозволяє виконувати емпіричний аналіз ігор у нормальній формі, обробляти матриці виграшів, застосовувати алгоритми аналізу стратегічної взаємодії та подавати результати у зручному для користувача вигляді.

Беручи за основу цю функцію, можна виділити такі основні функції програмного продукту:

- F_1 – вибір мови програмування для реалізації програмного продукту;
- F_2 – вибір способу реалізації аналітичного ядра;
- F_3 – вибір способу реалізації користувацького інтерфейсу;

F_4 – вибір способу візуалізації та збереження результатів аналізу.

Кожна з цих функцій має декілька варіантів реалізації.

Функція F_1 :

- а) Python;
- б) JavaScript;
- в) C++.

Функція F_2 :

- а) власна реалізація алгоритмів аналізу ігор;
- б) використання готових бібліотек без суттєвої модифікації;
- в) комбінований підхід.

Функція F_3 :

- а) веб-застосунок на основі Streamlit;
- б) реалізація у вигляді Jupyter Notebook або Google Colab;
- в) desktop-застосунок.

Функція F_4 :

- а) інтерактивна візуалізація з можливістю збереження графіків і текстових результатів;
- б) статичні графіки без інтерактивної взаємодії;
- в) текстовий вивід результатів у вигляді таблиць і логів.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

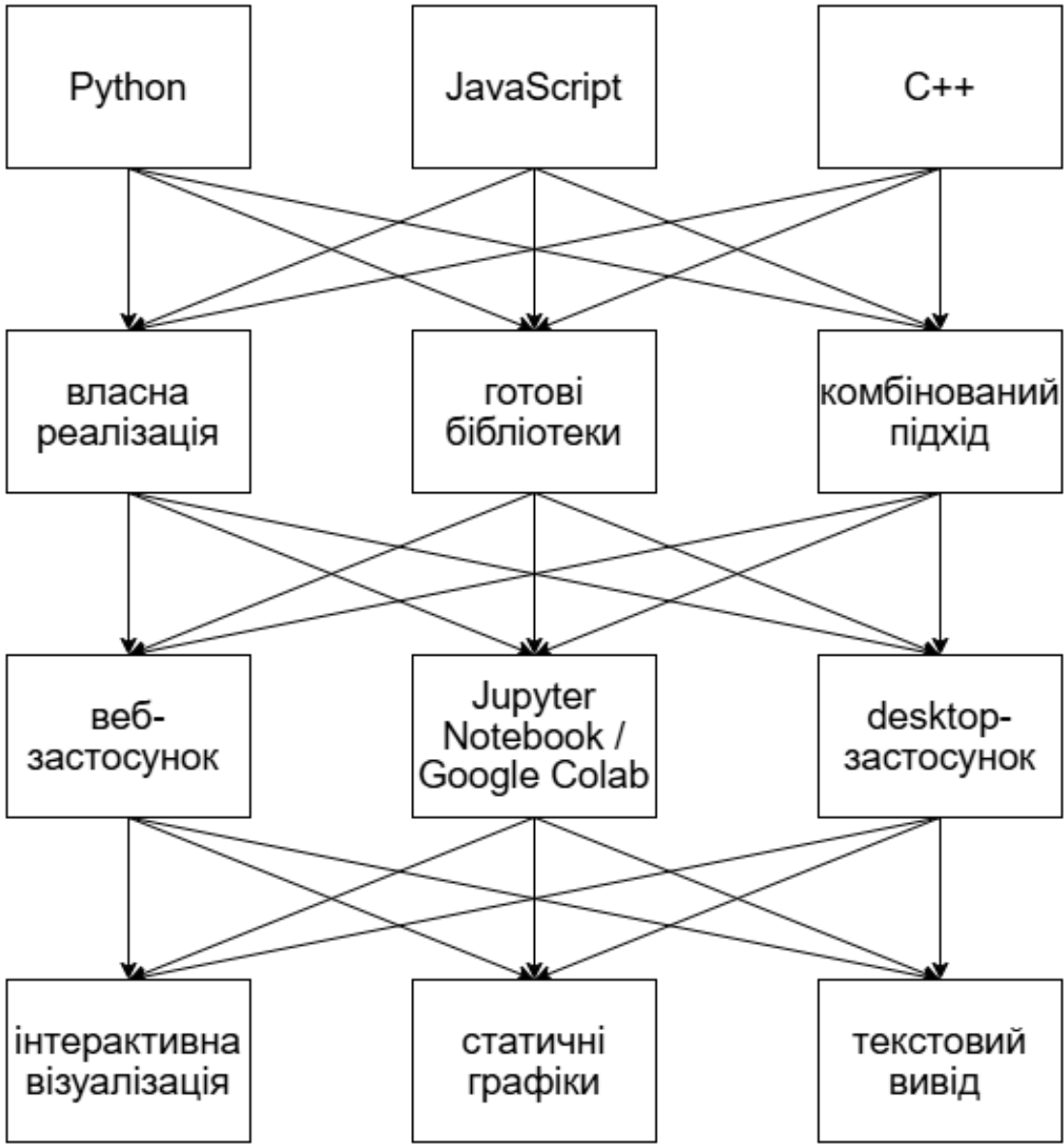


Рисунок 4.1 Морфологічна карта

Морфологічна карта відображає множину можливих варіантів реалізації основних функцій програмного продукту. Для подальшого аналізу було побудовано позитивно-негативну матрицю (таблиця 4.1), яка дозволяє оцінити переваги та недоліки кожного варіанта реалізації.

Таблиця 4.1. Позитивно-негативна матриця

Функція	Варіант реалізації	Переваги	Недоліки
F_1	а	Велика кількість бібліотек для аналізу даних, роботи з графами, візуалізації та машинного навчання; швидка розробка; зручність тестування алгоритмів.	Нижча швидкодія порівняно з C++ для дуже великих обчислень.
	б	Зручність для створення веб-інтерфейсів, можливість роботи у браузері.	Менш зручна реалізація математичного та графового аналізу, менше спеціалізованих бібліотек для наукових обчислень.
	в	Висока швидкодія та ефективне використання пам'яті	Висока складність розробки, більші витрати часу на реалізацію інтерфейсу та візуалізацію
F_2	а	Повний контроль над логікою роботи програми, можливість адаптації алгоритмів під задачу дипломної роботи	Потребує більше часу на реалізацію, тестування та перевірку коректності

Продовження таблиці 4.1

F_2	б	Швидка реалізація базових функцій, використання перевірених інструментів	Обмежена гнучкість, складність адаптації під специфіку власного програмного продукту
	в	Поєднання гнучкості власної реалізації з можливостями спеціалізованих бібліотек; зручність перевірки та розширення системи	Потребує узгодження різних модулів і додаткового тестування
F_3	а	Швидка розробка інтерфейсу, зручне відображення графіків, таблиць і параметрів; можливість інтерактивної роботи користувача	Потребує встановлення залежностей і запуску локального сервера
	б	Зручність для експериментів і поетапного запуску коду	Менш зручний формат для кінцевого користувача, обмежені можливості цілісного інтерфейсу
	в	Може працювати як окрема програма без браузера	Більша складність розробки, залежність від операційної системи

Кінець таблиці 4.1

F_4	а	Найбільш повно відповідає темі роботи; дозволяє аналізувати матриці виграшів, графи відповідей, кластери, α -Rank і ландшафт ігор у наочному вигляді	Потребує більшої кількості програмних модулів
	б	Простота реалізації, достатньо для базового аналізу	Менша зручність дослідження результатів, відсутність інтерактивності
	в	Мінімальні витрати на реалізацію, простота збереження логів	Недостатня наочність, не відповідає повною мірою ідеї візуального середовища

На основі аналізу позитивно-негативної матриці можна зробити висновок, що для розробки програмного продукту доцільно обрати ті варіанти реалізації, які забезпечують достатню функціональність, зручність користування та можливість наочного подання результатів.

Для функції F_1 перевагу надано мові Python. Цей варіант є найбільш доцільним, оскільки Python має розвинену екосистему бібліотек для математичних обчислень, аналізу даних, роботи з графами та побудови візуалізацій. Використання JavaScript більше підходить для розробки веб-інтерфейсів, однак є менш зручним для реалізації аналітичного ядра. Мова C++ забезпечує високу продуктивність, але потребує значно більших витрат часу на розробку та налагодження.

Для функції F_2 обрано комбінований підхід. Він передбачає власну реалізацію основних алгоритмів аналізу ігор у нормальній формі з використанням спеціалізованих бібліотек для окремих обчислювальних і візуалізаційних задач. Такий варіант дозволяє адаптувати програмний продукт до теми дипломної роботи, зберігаючи при цьому можливість використовувати готові інструменти для роботи з графами, матрицями, кластеризацією та побудовою графіків.

Для функції F_3 перевагу надано веб-застосунку на основі Streamlit. Такий варіант дозволяє створити зручний інтерфейс для введення даних, вибору ігор, запуску аналізу та перегляду результатів. Реалізація у вигляді Jupyter Notebook або Google Colab є зручною для початкових експериментів, однак менш придатна для використання як повноцінне візуальне середовище. Desktop-застосунок потребує більшої трудомісткості розробки та складнішої підтримки.

Для функції F_4 обрано інтерактивну візуалізацію з можливістю збереження графіків і текстових результатів. Цей варіант найбільше відповідає меті дипломної роботи, оскільки програмний продукт має не лише виконувати обчислення, а й забезпечувати зрозуміле візуальне подання матриць виграшів, графів відповідей, рівноваг Неша, спектральних проєкцій, кластерів і ландшафту ігор.

Отже, для подальшого аналізу розглядатимемо такі варіанти реалізації програмного продукту:

Варіант 1: $F_1(a) - F_2(b) - F_3(a) - F_4(a)$.

Варіант 2: $F_1(a) - F_2(a) - F_3(b) - F_4(b)$.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня програмного продукту. Обрані параметри повинні характеризувати як технічні властивості розроблюваного візуального середовища, так і зручність його подальшої експлуатації та розвитку.

Для оцінювання програмного продукту будемо використовувати такі параметри:

X_1 – потенційний обсяг програмного коду;

X_2 – обсяг оперативної пам'яті, необхідний для роботи програмного продукту;

X_3 – час виконання повного аналізу гри;

X_4 – функціональна повнота системи.

Параметр X_1 характеризує орієнтовний обсяг програмного коду, необхідний для реалізації основних модулів системи. До них належать модулі введення матриць виграшів, аналізу рівноваг Неша, побудови графа відповідей, розрахунку показників α -Rank, спектрального аналізу, кластеризації та побудови ландшафту ігор. Менше значення цього параметра є кращим, оскільки воно свідчить про нижчу складність супроводу та налагодження програмного продукту.

Параметр X_2 відображає обсяг оперативної пам'яті, який потрібен для виконання обчислень і побудови візуалізацій. Цей параметр є важливим, оскільки програмний продукт працює з матрицями виграшів, графовими структурами, результатами кластеризації та візуальними об'єктами. Менше значення цього параметра є кращим, оскільки забезпечує можливість використання системи на звичайному персональному комп'ютері.

Параметр X_3 характеризує швидкість роботи системи, а саме час, необхідний для виконання повного аналізу однієї гри. До повного аналізу входить обробка матриці виграшів, пошук рівноваг Неша, побудова графа відповідей, розрахунок аналітичних показників і формування візуалізацій. Менше значення цього параметра є кращим, оскільки воно забезпечує зручнішу взаємодію користувача з програмним продуктом.

Параметр X_4 визначає функціональну повноту системи, тобто кількість основних аналітичних і візуалізаційних можливостей, реалізованих у програмному продукті. До таких можливостей належать робота з матрицями виграшів, пошук рівноваг Неша, побудова графів відповідей, α -Rank, спектральний аналіз, кластеризація, побудова ландшафту ігор та збереження результатів. Більше значення цього параметра є кращим, оскільки воно свідчить про ширші можливості системи.

Гірші, середні та кращі значення параметрів вибираються на основі вимог користувача, особливостей розроблюваного програмного продукту та умов його експлуатації. Значення параметрів наведено в таблиці 4.2.

Таблиця 4.2. Основні параметри програмного продукту

Назва параметра	Умовне позначення	Одиниці виміру	Гірше	Середнє	Краще
Потенційний обсяг програмного коду	X_1	рядки коду	60000	45000	30000
Обсяг оперативної пам'яті, необхідний для роботи ПП	X_2	МБ	1024	512	256
Час виконання повного аналізу гри	X_3	с	20	10	3
Функціональна повнота системи	X_4	к-сть функцій	4	6	8

За даними таблиці 4.2 будуються графічні характеристики параметрів (рис. 4.2 – 4.5), які відображають залежність бальної оцінки від абсолютного значення кожного параметра. Для параметрів X_1 , X_2 та X_3 кращим є менше значення, оскільки воно відповідає меншій складності реалізації, нижчим вимогам до ресурсів та вищій швидкодії. Для параметра X_4 кращим є більше значення, оскільки воно характеризує ширшу функціональність програмного продукту.

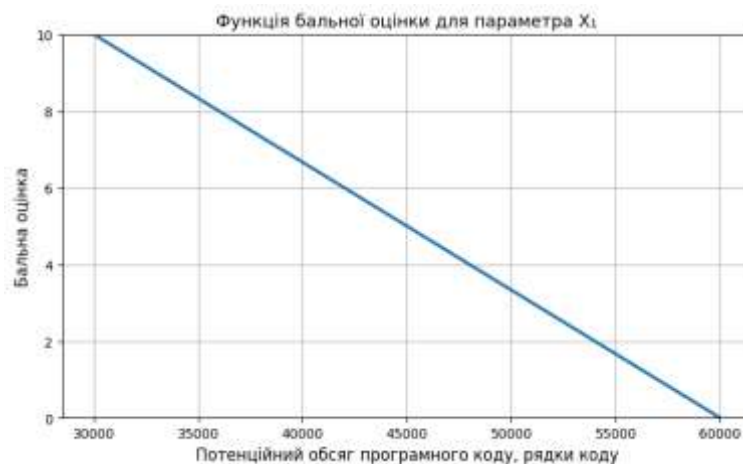


Рисунок 4.2 X_1 , потенційний обсяг програмного коду

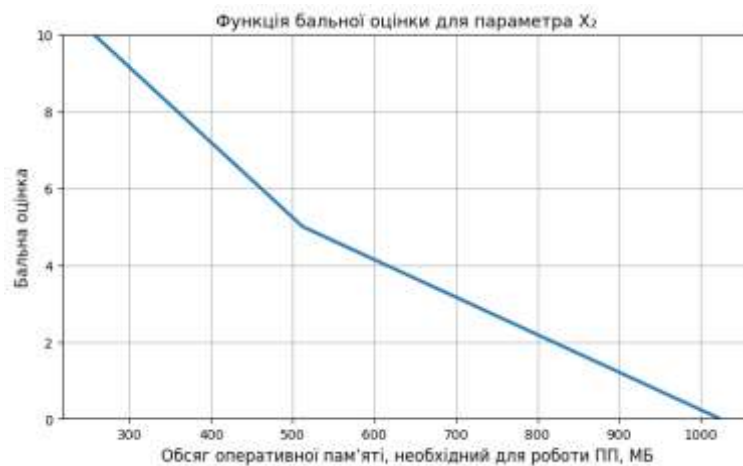
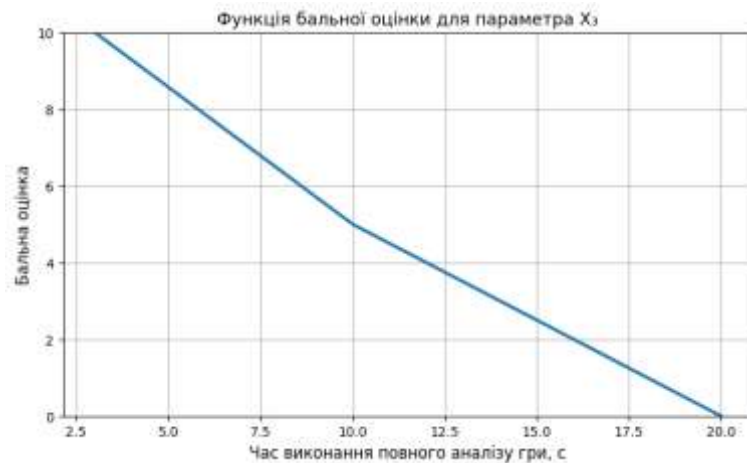
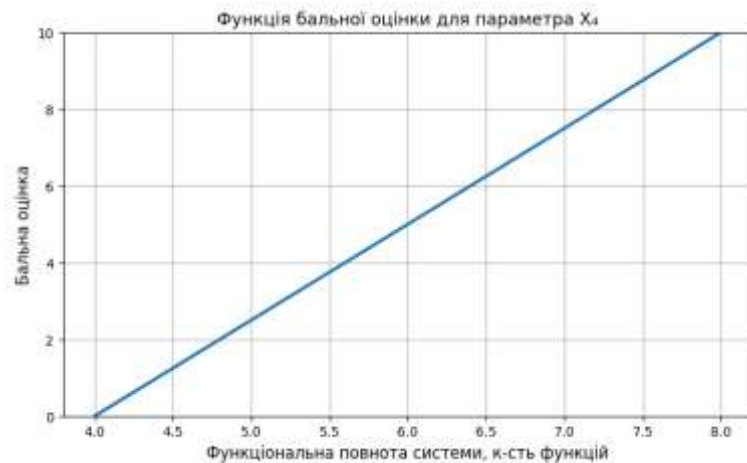


Рисунок 4.3 X_2 , обсяг оперативної пам'яті, необхідний для роботи програмного продукту

Рисунок 4.4 X_3 , час виконання повного аналізу гриРисунок 4.5 X_4 , функціональна повнота системи

4.4 Аналіз експертного оцінювання параметрів

Після визначення основних параметрів програмного продукту необхідно встановити їхню відносну значущість. Для цього використовується метод експертного оцінювання, який дозволяє визначити вагомість кожного параметра з урахуванням його впливу на якість та ефективність розроблюваного програмного продукту.

Перевіримо достовірність експертних оцінок.

Загальна сума рангів становить:

$$R_i = \sum_{j=1}^N r_{ij} R_{ij} = \frac{N \cdot n \cdot (n + 1)}{2} = \frac{7 \cdot 4 \cdot (4 + 1)}{2} = 70,$$

де N – число експертів, n – кількість параметрів.

Середня сума рангів визначається за формулою:

$$T = \frac{1}{n} R_i = \frac{70}{4} = 17,5$$

Відхилення суми рангів кожного параметра від середньої суми визначається за формулою:

$$\Delta_i = R_i - T.$$

Сума квадратів відхилень становить:

$$S = \sum_{i=1}^N \Delta_i^2 = 213.$$

Для перевірки узгодженості думок експертів розрахуємо коефіцієнт конкордації:

$$W = \frac{12 \cdot S}{N^2(n^3 - n)} = \frac{12 \cdot 213}{7^2(4^3 - 4)} = 0,87.$$

Для програмного продукту нормативне значення коефіцієнта узгодженості становить 0,67. Оскільки отримане значення більше, то експертні оцінки можна вважати узгодженими та придатними для подальших розрахунків.

Далі для визначення коефіцієнтів вагомості параметрів проведено попарне порівняння.

Результати попарного порівняння параметрів наведено в таблиці 4.4.

Таблиця 4.4 Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X_1 і X_2	<	<	<	=	<	<	<	<	0,5
X_1 і X_3	<	<	<	<	<	<	<	<	0,5
X_1 і X_4	<	<	<	<	<	<	<	<	0,5
X_2 і X_3	<	<	<	<	<	=	<	<	0,5
X_2 і X_4	<	<	<	<	<	<	<	<	0,5
X_3 і X_4	<	<	<	<	<	<	=	<	0,5

У результаті порівняння встановлено, що найбільшу значущість має параметр X_4 , тобто функціональна повнота системи, оскільки саме вона визначає відповідність програмного продукту меті дипломної роботи. Наступним за важливістю є параметр X_3 , який характеризує швидкодію системи. Параметр X_2 є важливим з точки зору ресурсомісткості, а параметр X_1 має найменшу вагу, оскільки більший обсяг коду не завжди прямо визначає якість програмного продукту.

Числове значення, що визначає ступінь переваги i -го параметра над j -тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 & \text{при } X_i > X_j \\ 1.0 & \text{при } X_i = X_j \\ 0.5 & \text{при } X_i < X_j \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю $A = \|a_{ij}\|$.

Для кожного параметра зробимо розрахунок вагомості K_{gi} за наступними формулами:

$$K_{\text{вi}} = \frac{b_i}{\sum_{i=1}^n b_i}$$

$$b_i = \sum_{j=1}^N a_{ij}$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятись від попередніх (менше 2%). На

другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{Bi}} = \frac{b'_i}{\sum_{i=1}^n b'_i},$$

$$b'_i = \sum_{j=1}^N a_{ij} b_j$$

На основі матриці попарного порівняння розраховуються коефіцієнти вагомості параметрів. Результати розрахунку наведено в таблиці 4.5.

Таблиця 4.5 Розрахунок вагомості параметрів

	Параметри x_j				Перша ітерація		Друга ітерація	
Параметри x_i	X_1	X_2	X_3	X_4	b_i	K_{Bi}	b_i^1	K_{Bi}^1
X_1	1	0,5	0,5	0,5	2,5	0,156	9,25	0,157
X_2	1,5	1	0,5	0,5	3,5	0,219	12,25	0,208
X_3	1,5	1,5	1	0,5	4,5	0,281	16,25	0,275
X_4	1,5	1,5	1,5	1	5,5	0,344	21,25	0,360
Всього:					16	1	59	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту реалізації програмного продукту окремо. Абсолютні значення параметрів X_1 , X_2 , X_3 та X_4 відповідають технічним вимогам умов функціонування розроблюваного програмного продукту.

Коефіцієнт технічного рівня для кожного варіанта реалізації програмного продукту розраховується за формулою (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^n K_{ei,j} B_{i,j},$$

де n – кількість параметрів; K_{bi} – коефіцієнт вагомості i -го параметра; B_i – оцінка i -го параметра в балах.

Бальні оцінки параметрів визначаються на основі графічних залежностей, побудованих за даними таблиці 4.2. Для параметрів X_1 , X_2 , X_3 кращим є менше абсолютне значення, оскільки воно відповідає меншому обсягу коду, нижчим вимогам до оперативної пам'яті та меншому часу виконання аналізу. Для параметра X_4 кращим є більше значення, оскільки воно характеризує функціональну повноту програмного продукту.

Для подальшого порівняння розглянемо два варіанти реалізації програмного продукту:

Варіант 1: $F_1(a) - F_2(b) - F_3(a) - F_4(a)$.

Варіант 2: $F_1(a) - F_2(a) - F_3(b) - F_4(b)$.

Перший варіант передбачає створення повноцінного візуального середовища на Python із комбінованим підходом до реалізації аналітичного ядра, веб-інтерфейсом та інтерактивною візуалізацією результатів. Другий варіант передбачає спрощену реалізацію у вигляді набору скриптів або Jupyter Notebook / Google Colab зі статичними графіками.

Таблиця 4.6 Розрахунок показників рівня якості варіантів реалізації основних функцій програмного продукту

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення	Бальна оцінка	Коефіцієнт вагомості	Коефіцієнт рівня якості
F_1	A	X_1	40000	7	0,157	1,1
F_2	B	X_2	500	5	0,208	1,04
	A	X_2	1000	1	0,208	0,208

Кінець таблиці 4.6

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення	Бальна оцінка	Коефіцієнт вагомості	Коефіцієнт рівня якості
F_3	А	X_3	4	9	0,275	2,475
	Б	X_3	18	1	0,275	0,275
F_4	А	X_4	7	7	0,36	2,52
	Б	X_4	5	3	0,36	1,08

За даними таблиці 4.6 визначимо рівень якості кожного з варіантів реалізації програмного продукту.

Для першого варіанта:

$$K_{K1} = 1,1 + 1,04 + 2,475 + 2,52 = 7,135$$

Для другого варіанта:

$$K_{K2} = 1,1 + 0,208 + 0,275 + 1,08 = 2,663$$

Порівняння отриманих результатів показує, що перший варіант реалізації має вищий коефіцієнт технічного рівня. Це пояснюється тим, що він забезпечує кращу швидкодію, більшу функціональну повноту та ширші можливості візуального подання результатів аналізу. Другий варіант є простішим у реалізації, однак поступається за зручністю використання, повнотою функцій та якістю подання результатів.

4.6 Економічний аналіз варіантів розробки програмного продукту

У даному прикладі ФВА виконується на стадії розробки програмного продукту. Вартість розробки програмного продукту розраховують за формулами, для чого спочатку проводять розрахунок трудомісткості.

Усі варіанти включають у себе три окремі завдання:

1. Реалізація структури програмного продукту та моделі представлення ігор у нормальній формі.
2. Реалізація аналітичного ядра для обробки матриць виграшів, пошуку рівноваг Неша, побудови графа відповідей, α -Rank, спектрального аналізу та кластеризації.
3. Створення компонента для візуалізації результатів аналізу.

Варіант I містить ще два завдання:

4. Розробка веб-застосунку з інтерактивним інтерфейсом.
5. Написання компонента для збереження графічних і текстових результатів аналізу.

Цим завданням у варіанті II відповідають такі:

6. Реалізація програмного продукту у вигляді Jupyter Notebook / Google Colab зі статичними графіками.
7. Написання компонента для текстового та табличного виведення результатів.

Завдання 1 за ступенем новизни належить до групи Б, завдання 2 – до групи А, завдання 3 – до групи Б, завдання 4 і 5 – до групи Б, завдання 6 і 7 – до групи Г.

За складністю алгоритми, використовувані у завданнях 1, 3, 4, 6, належать до групи 2; у завданні 2 – до групи 1; у завданнях 5 і 7 – до групи 3.

Усі завдання використовують контроль вхідної інформації, яка характеризується групою 5.2, та контроль вихідної інформації, яка

характеризується групою 5.4. Тому поправковий коефіцієнт, який враховує складність контролю вхідної та вихідної інформації, для всіх завдань дорівнює: $K_{СК} = 1$.

Для реалізації завдання 1 використовується інформація у вигляді банку даних, оскільки програмний продукт працює з наборами матриць виграшів і профілями стратегій. Реалізація завдання 2 використовує нормативно-довідкову інформацію, пов'язану з алгоритмами теорії ігор, графового аналізу та спектральної кластеризації. Завдання 3, 4 і 5 використовують інформацію у вигляді банку даних, оскільки працюють із результатами обчислень, графами, таблицями та візуальними об'єктами. Завдання 6 і 7 використовують змінну інформацію.

Розрахуємо норму часу на розробку та програмування для кожного із семи завдань.

Загальна трудомісткість обчислюється за формулою:

$$T_0 = T_P \cdot K_P \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М},$$

де T_P – трудомісткість розробки ПП;

K_P – поправковий коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

K_M – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Коефіцієнти K_M і $K_{СТ.М}$, які враховують відповідно рівень мови програмування та використання стандартного математичного забезпечення, для всіх завдань приймаємо рівними 1.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру ступеня новизни Б та групи складності алгоритму 2, трудомісткість становить $T_P = 27$ людино-днів.

Поправковий коефіцієнт, який враховує вид використаної інформації для першого завдання $K_{\Pi}=0,9$.

Оскільки під час розробки першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта:

$$K_{CT}=0,8.$$

Отже, загальна трудомісткість програмування першого завдання дорівнює:

$$T_1=27 \cdot 0,9 \cdot 0,8=19,44 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для інших завдань.

Для другого завдання, що пов'язане з реалізацією аналітичного ядра програмного продукту, використовується алгоритм першої групи складності, ступінь новизни А:

$$T_P=90 \text{ людино-днів;}$$

$$K_{\Pi}=1,7;$$

$$K_{CT}=0,8.$$

Тоді:

$$T_2=90 \cdot 1,7 \cdot 0,8=122,40 \text{ людино-днів.}$$

Для третього завдання, що передбачає створення компонента для візуалізації результатів аналізу, використовується алгоритм другої групи складності, ступінь новизни Б:

$$T_P=27 \text{ людино-днів;}$$

$$K_{\Pi}=0,9;$$

$$K_{CT}=0,8.$$

Тоді:

$$T_3=27 \cdot 0,9 \cdot 0,8=19,44 \text{ людино-днів.}$$

Для четвертого завдання, що передбачає розробку веб-застосунку з інтерактивним інтерфейсом, використовується алгоритм другої групи складності, ступінь новизни Б:

$$T_P=27 \text{ людино-днів;}$$

$$K_{\Pi}=0,9;$$

$$K_{CT}=0,8.$$

Тоді:

$$T_4=27 \cdot 0,9 \cdot 0,8=19,44 \text{ людино-днів.}$$

Для п'ятого завдання, що передбачає написання компонента для збереження графічних і текстових результатів аналізу, використовується алгоритм третьої групи складності, ступінь новизни Б:

$$T_P=27 \text{ людино-днів;}$$

$$K_{\Pi}=0,9;$$

$$K_{CT}=1.$$

Тоді:

$$T_5=27 \cdot 0,9 \cdot 1=24,30 \text{ людино-днів.}$$

Для шостого завдання, що передбачає реалізацію програмного продукту у вигляді Jupyter Notebook / Google Colab зі статичними графіками, використовується алгоритм другої групи складності, ступінь новизни Г:

$$T_P=12 \text{ людино-днів;}$$

$$K_{\Pi}=0,72;$$

$$K_{CT}=0,8.$$

Тоді:

$$T_6=12 \cdot 0,72 \cdot 0,8=6,91 \text{ людино-днів.}$$

Для сьомого завдання, що передбачає текстове та табличне виведення результатів, використовується алгоритм третьої групи складності, ступінь новизни Г:

$$T_P=8 \text{ людино-днів;}$$

$$K_{\Pi}=0,6;$$

$$K_{CT}=1.$$

Тоді:

$$T_7=8 \cdot 0,6 \cdot 1=4,80 \text{ людино-днів.}$$

Складемо трудомісткість відповідних завдань для кожного з двох обраних варіантів реалізації програмного продукту.

Для першого варіанта:

$$T_I = (19,44 + 122,40 + 19,44 + 19,44 + 24,30) \cdot 8 = 1640,16 \text{ людино-годин.}$$

Для другого варіанта:

$$T_{II} = (19,44 + 122,40 + 19,44 + 6,91 + 4,80) \cdot 8 = 1383,92 \text{ людино-годин.}$$

Очевидно, що найвищу трудомісткість має варіант I, оскільки він передбачає створення повноцінного веб-застосунку з інтерактивною візуалізацією та збереженням результатів аналізу.

У розробці бере участь один інженер-програміст з окладом 55000 грн. Визначимо зарплату програміста за годину:

$$C_q = \frac{M}{T_m \cdot t} \text{ грн,}$$

де M – місячний оклад працівника; T_m – кількість робочих днів на місяць; t – кількість робочих годин на день.

$$C_q = \frac{55000}{21 \cdot 8} = 327,38 \text{ грн}$$

Тоді зарплата розробника за варіантами відповідно до формули:

$$C_{зп} = C_q \cdot T_i \cdot K_d$$

де C_q – величина погодинної оплати праці програміста; T_i – трудомісткість відповідного варіанта; K_d – норматив, який враховує додаткову заробітну плату.

Зарплата розробника за варіантами становить:

$$C_{зп.I} = 327,38 \cdot 1640,16 \cdot 1,2 = 644346,7 \text{ грн.}$$

$$C_{зп.II} = 327,38 \cdot 1383,92 \cdot 1,2 = 543681,27 \text{ грн.}$$

Відрахування на єдиний соціальний внесок становить 22%:

$$C_{від} = C_{зп} \cdot 0,22.$$

Для першого варіанта:

$$C_{від.I} = 644346,7 \cdot 0,22 = 141756,27 \text{ грн.}$$

Для другого варіанта:

$$C_{\text{від.п}} = 543681,27 \cdot 0,22 = 119609,88 \text{ грн.}$$

Потім визначимо витрати на оплату однієї машино-години C_M .

Оскільки ЕОМ обслуговує один спеціаліст з окладом 55000 грн, з коефіцієнтом зайнятості 0,2, то для однієї машини отримаємо:

$$C_{\Gamma} = 12 \cdot M \cdot K_3 = 12 \cdot 55000 \cdot 0,2 = 132000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$C_{\text{зп}} = C_{\Gamma} \cdot (1 + K_3) = 132000 \cdot (1 + 0,2) = 158400 \text{ грн.}$$

Відрахування на соціальний внесок:

$$C_{\text{від}} = C_{\text{зп}} \cdot 0,22 = 158400 \cdot 0,22 = 34848 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 60000 грн.

$$C_A = K_{\text{тм}} \cdot K_A \cdot C_{\text{пп}} = 1,2 \cdot 0,25 \cdot 60000 = 18000 \text{ грн.,}$$

де $K_{\text{тм}}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; K_A – річна норма амортизації; $C_{\text{пп}}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$C_P = K_{\text{тм}} \cdot C_{\text{пп}} \cdot K_P = 1,2 \cdot 60000 \cdot 0,05 = 3600 \text{ грн.,}$$

де K_P – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$\begin{aligned} T_{\text{еф}} &= (D_K - D_B - D_C - D_P) \cdot t_3 \cdot K_B = (365 - 104 - 0 - 16) \cdot 8 \cdot 0,40 = \\ &= 784 \text{ години,} \end{aligned}$$

де D_K – календарна кількість днів у році; D_B , D_C – відповідно кількість вихідних та святкових днів; D_P – кількість днів планових ремонтів устаткування; t – кількість робочих годин в день; K_B – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$C_{\text{ел}} = T_{\text{еф}} \cdot N_C \cdot K_3 \cdot C_{\text{ен}} = 784 \cdot 0,3 \cdot 0,2 \cdot 13,64 = 641,63 \text{ грн,}$$

де N_C – середньо-споживча потужність приладу; K_3 – коефіцієнтом зайнятості приладу; Π_{EH} – тариф за 1 кВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$C_H = \Pi_{EP} \cdot 0,67 = 60000 \cdot 0,67 = 40200 \text{ грн.}$$

Тоді річні експлуатаційні витрати:

$$C_{EKC} = C_{3П} + C_{ВІД} + C_A + C_{РЕМ} + C_{ЕЛ} + C_{НАКЛ},$$

$$C_{EKC} = 158400 + 34848 + 18000 + 3600 + 641,63 + 40200 = 255689,63 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{M-Г} = C_{EKC} / T_{ЕФ} = 255689,63 / 784 = 326,135 \text{ грн/год}$$

Оскільки в даному випадку всі роботи, які пов'язані з розробкою апаратно-програмного засобу, ведуться на ЕОМ, витрати на оплату машинного часу, залежно від обраного варіанта реалізації, складають:

$$C_M = C_{M-Г} \cdot T.$$

$$C_{M I} = 326,135 \cdot 1640,16 = 534913,58 \text{ грн.}$$

$$C_{M II} = 326,135 \cdot 1383,92 = 451344,75 \text{ грн.}$$

Накладні витрати складають 67% від заробітної плати:

$$C_H = C_{3П} \cdot 0,67,$$

$$C_{H I} = 644346,7 \cdot 0,67 = 431712,29 \text{ грн.}$$

$$C_{H II} = 543681,27 \cdot 0,67 = 364266,45 \text{ грн.}$$

Отже, вартість розробки за варіантами становить:

$$C = C_{3П} + C_{ВІД} + C_M + C_H,$$

$$C_I = 644346,7 + 141756,274 + 534913,58 + 431712,29 = 1752728,84 \text{ грн.}$$

$$C_{II} = 543681,27 + 119609,88 + 451344,75 + 364266,45 = 1478902,35 \text{ грн.}$$

4.7 Вибір оптимального варіанта реалізації програмного продукту

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$K_{\text{TEP}j} = K_{\text{K}j} / C_{\text{Ф}j}$$

$$K_{\text{TEP}1} = 7,135 / 1752728,84 = 0,0000040708,$$

$$K_{\text{TEP}2} = 2,663 / 1478902,35 = 0,0000018006.$$

Порівняємо отримані значення:

$$K_{\text{TEP}1} > K_{\text{TEP}2}.$$

Отже, найбільш ефективним є перший варіант реалізації програмного продукту, оскільки він має вищий коефіцієнт техніко-економічного рівня.

$$K_{\text{TEP}} = 0,0000040708.$$

Після виконання функціонально-вартісного аналізу програмного продукту можна зробити висновок, що з альтернативних варіантів реалізації оптимальним є варіант I. Він передбачає використання мови програмування Python, комбінованого підходу до реалізації аналітичного ядра, веб-застосунку та інтерактивної візуалізації результатів.

Хоча перший варіант має більшу вартість розробки, він забезпечує значно вищий рівень якості, більшу функціональну повноту, зручніший інтерфейс та кращу відповідність темі дипломної роботи – створенню візуального середовища для емпіричного аналізу ігор у нормальній формі. Саме тому для подальшої реалізації обирається перший варіант програмного продукту.

4.8 Висновки до розділу 4

У даному розділі було проведено функціонально-вартісний аналіз програмного продукту для емпіричного аналізу ігор у нормальній формі.

Було визначено основні функції програмного продукту, а саме: вибір мови програмування, вибір способу реалізації аналітичного ядра, вибір способу реалізації користувацького інтерфейсу та вибір способу візуалізації й збереження результатів аналізу. Для кожної функції було розглянуто декілька варіантів реалізації, побудовано морфологічну карту та позитивно-негативну матрицю, а також обґрунтовано вибір найбільш доцільних рішень.

Для оцінювання якості програмного продукту було обґрунтовано систему параметрів, до якої увійшли потенційний обсяг програмного коду, обсяг оперативної пам'яті, час виконання повного аналізу гри та функціональна повнота системи. На основі експертного оцінювання було визначено вагомість кожного параметра та розраховано коефіцієнти рівня якості для двох варіантів реалізації.

У результаті аналізу було обрано варіант реалізації, що передбачає використання мови програмування Python, комбінованого підходу до реалізації аналітичного ядра, веб-застосунку на основі Streamlit, а також інтерактивної візуалізації та збереження результатів аналізу. Такий варіант найбільш повно відповідає меті дипломної роботи, оскільки забезпечує роботу з таблицями виграшів, пошук рівноваг Неша, побудову графів відповідей, розрахунок показників α -Rank, спектральний аналіз, кластеризацію та побудову ландшафту ігор.

ВИСНОВКИ

У цій роботі було досліджено задачу аналізу ігор у нормальній формі в контексті емпіричного теоретико-ігрового аналізу. Розглянуто проблему інтерпретації таблиць виграшів, а також необхідність використання графових і візуальних методів для дослідження структури стратегічної взаємодії.

Описано основні методи, що використовуються для аналізу ігор: пошук найкращих відповідей, рівноваг Неша, побудова графа відповідей, визначення структурних характеристик, застосування α -Rank, спектрального аналізу, кластеризації та методів зниження розмірності.

Створено програмний продукт, який дозволяє аналізувати ігри у нормальній формі на основі вже сформованих таблиць виграшів. Розроблене середовище забезпечує побудову графів відповідей, аналіз циклів, рівноваг, кластерів стратегічних профілів, а також порівняння різних ігор за допомогою ландшафту ігор.

Проведено обчислювальні експерименти на прикладах класичних, синтетичних, прикладних та емпірично поданих OpenSpiel-ігор. Показано, що ландшафт ігор дає змогу порівнювати стратегічні структури, виявляти подібні ігри та створює основу для подальшого формування нових ігрових структур із заданими властивостями.

Описано потенційні витрати на реалізацію програмного продукту та проведено функціонально-вартісний аналіз, за результатами якого обґрунтовано вибір найбільш ефективного варіанта реалізації системи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Omidshafiei S. et al. Navigating the landscape of multiplayer games. *Nature Communications*. 2020. Vol. 11. Article 5603. DOI: <https://doi.org/10.1038/s41467-020-19244-4>.
2. Кравець П. О. Ігрові стратегії прийняття рішень в ієрархічних системах. І. Математична модель стохастичної гри. *Системні дослідження та інформаційні технології*. 2019. № 3. С. 63–75. DOI: <https://doi.org/10.20535/SRIT.2308-8893.2019.3.06>
3. Барановська Л. В. Теорія ігор: курс лекцій: навч. посіб. Київ: КПІ ім. Ігоря Сікорського, 2022. 245 с. URL: <https://ela.kpi.ua/handle/123456789/49092> (дата звернення: 15.05.2026).
4. Prisner E. *Game Theory Through Examples*. Washington: Mathematical Association of America, 2014. 308 p. URL: <https://www.maa.org/press/ebooks/game-theory-through-examples> (дата звернення: 15.05.2026).
5. Nash J. F. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the United States of America*. 1950. Vol. 36, No. 1. P. 48–49. DOI: <https://doi.org/10.1073/pnas.36.1.48>.
6. Wellman M. P. Methods for empirical game-theoretic analysis. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2006. Vol. 21, No. 2. P. 1552–1556. URL: <https://aaai.org/papers/01552-aaai06-248-methods-for-empirical-game-theoretic-analysis/> (дата звернення: 15.05.2026).
7. Tuyls K., Perolat J., Lanctot M., Leibo J. Z., Graepel T. A generalised method for empirical game theoretic analysis. *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent*

- Systems*. Richland: International Foundation for Autonomous Agents and Multiagent Systems, 2018. P. 77–85. URL: <https://dl.acm.org/doi/10.5555/3237383.3237402> (дата звернення: 15.05.2026).
8. Van Steen M. *Graph Theory and Complex Networks: An Introduction*. 2010. URL: <https://www.distributed-systems.net/index.php/books/gtcn/> (дата звернення: 15.05.2026).
 9. Omidshafiei S. et al. α -Rank: multi-agent evaluation by evolution. *Scientific Reports*. 2019. Vol. 9. Article 9937. DOI: <https://doi.org/10.1038/s41598-019-45619-9>.
 10. Mohar B. The Laplacian spectrum of graphs. *Graph Theory, Combinatorics, and Applications*. 1991. Vol. 2. P. 871–898. URL: <https://users.fmf.uni-lj.si/mohar/Papers/Spec.pdf> (дата звернення: 15.05.2026).
 11. Cvetkovic D.M., Doob M., Sachs H. *Spectra of Graphs: Theory and Applications*. 3rd ed. Johann Ambrosius Barth Verlag, Heidelberg-Leipzig, 1995. 447 p.
 12. Malliaros F. D., Vazirgiannis M. Clustering and community detection in directed networks: a survey. *Physics Reports*. 2013. Vol. 533, No. 4. P. 95–142. DOI: <https://doi.org/10.1016/j.physrep.2013.08.002>.
 13. McInnes L., Healy J., Melville J. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. 2018. URL: <https://arxiv.org/abs/1802.03426> (дата звернення: 15.05.2026).
 14. Lanctot M. et al. OpenSpiel: a framework for reinforcement learning in games. 2019. URL: <https://arxiv.org/abs/1908.09453> (дата звернення: 15.05.2026).
 15. Arad A., Rubinstein A. The 11–20 money request game: a level-k reasoning study. *American Economic Review*. 2012. Vol. 102, No. 7. P. 3561–3573. DOI: <https://doi.org/10.1257/aer.102.7.3561>.

ДОДАТОК А. ТЕКСТ ПРОГРАМИ

A.1 Код файлу app.py

```

import json
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import networkx as nx
import plotly.express as px
from sklearn.preprocessing import StandardScaler
from src.games import get_predefined_games, create_location_graph
from src.game_model import NormalFormGame
from src.analysis import (
    build_response_graph,
    find_best_responses,
    find_nash_equilibria,
    compute_graph_statistics,
    calculate_alpha_rank_fallback,
)
from src.landscape import (
    build_games_feature_table,
    compute_games_landscape,
    draw_games_landscape,
)
from src.visualization import (
    draw_response_graph,
    draw_spectral_graph,
    draw_clustered_graph,
    draw_contracted_graph,
    draw_alpha_rank_distribution,
    draw_payoff_heatmaps,
    draw_cycle_distribution,
)
from src.config import OUTPUT_DIR
st.set_page_config(
    page_title="Normal-Form Game Analysis",
    page_icon="🎲",
    layout="wide"
)
def make_safe_filename(text):
    filename = text.lower()
    filename = filename.replace(" ", "_")
    filename = filename.replace("'", "")
    filename = filename.replace("-", "_")
    filename = filename.replace(":", "")
    filename = filename.replace("/", "_")
    return filename
def format_strategy_number(index):
    return int(index) + 1
def format_strategy_list(strategies):
    return "[" + ", ".join(
        str(format_strategy_number(strategy))
        for strategy in strategies
    ) + "]"
def format_strategy_profile(profile):

```

```

    if isinstance(profile, tuple):
        return "(" + ", ".join(
            str(format_strategy_number(strategy))
            for strategy in profile
        ) + ")"
    return str(format_strategy_number(profile))
CUSTOM_GAMES_PATH = OUTPUT_DIR / "custom_games.json"
def make_custom_game_key(game_name):
    base = make_safe_filename(game_name).strip("_")
    if not base:
        base = "game"
    return f"custom_{base}"
def load_custom_games_from_file():
    if not CUSTOM_GAMES_PATH.exists():
        return {}
    try:
        with open(CUSTOM_GAMES_PATH, "r", encoding="utf-8") as file:
            data = json.load(file)
    except Exception:
        return {}
    custom_games = {}
    for game_key, game_data in data.items():
        try:
            payoff_matrix = np.array(
                game_data["payoff_matrix"],
                dtype=float
            )
            game_name = game_data.get("name", game_key)
            custom_games[game_key] = NormalFormGame(
                payoff_matrix,
                name=game_name
            )
        except Exception:
            pass
    return custom_games
def save_custom_games_to_file(custom_games):
    OUTPUT_DIR.mkdir(exist_ok=True)
    data = {}
    for game_key, game in custom_games.items():
        data[game_key] = {
            "name": game.name,
            "payoff_matrix": game.payoff_matrix.tolist()
        }
    with open(CUSTOM_GAMES_PATH, "w", encoding="utf-8") as file:
        json.dump(data, file, ensure_ascii=False, indent=4)
def initialize_custom_games():
    if "custom_games" not in st.session_state:
        st.session_state.custom_games = load_custom_games_from_file()
def save_custom_game(game):
    base_key = make_custom_game_key(game.name)
    game_key = base_key
    original_name = game.name
    counter = 2
    while game_key in st.session_state.custom_games:
        game_key = f"{base_key}_{counter}"
        game.name = f"{original_name} ({counter})"
        counter += 1
    st.session_state.custom_games[game_key] = game
    save_custom_games_to_file(
        st.session_state.custom_games
    )

```

```

        return game_key
def delete_custom_game(game_key):
    if game_key not in st.session_state.custom_games:
        return False
    del st.session_state.custom_games[game_key]
    save_custom_games_to_file(
        st.session_state.custom_games
    )
    return True
def render_custom_games_management_panel():
    st.markdown("### Manage saved custom games")
    if not st.session_state.custom_games:
        st.info("No saved custom games yet.")
        return
    custom_game_keys = list(st.session_state.custom_games.keys())
    selected_custom_game_key = st.selectbox(
        "Choose saved custom game to delete",
        custom_game_keys,
        format_func=lambda key: (
            f"{st.session_state.custom_games[key].name} ({key})"
        ),
        key="delete_custom_game_selectbox"
    )
    confirm_delete = st.checkbox(
        "Confirm deletion",
        key="confirm_delete_custom_game"
    )
    if st.button(
        "Delete selected custom game",
        disabled=not confirm_delete,
        type="secondary"
    ):
        deleted = delete_custom_game(selected_custom_game_key)

        if deleted:
            st.success("Custom game deleted successfully.")
            st.rerun()
        else:
            st.error("Could not delete this custom game.")
def get_game_information(game_key, game_name):
    game_information = {
        "prisoners_dilemma": {
            "rules": """
                Two players simultaneously choose one of two actions:
                Cooperate or Defect.
                If both players cooperate, both receive a moderate payoff,
                because cooperation
                is beneficial for both sides.
                If one player defects while the other cooperates, the defector
                receives the
                highest payoff, while the cooperating player receives the
                lowest payoff.
                If both players defect, both receive a low payoff. This
                outcome is worse than
                mutual cooperation, but each player has an individual
                incentive to defect.
            """
        },
        "matching_pennies": {
            "rules": """

```

```

    Two players simultaneously choose one of two sides, for
example Heads or Tails.
    Player 1 wants both players to choose the same side.
    Player 2 wants the players to choose different sides.
    If the choices are the same, Player 1 wins and Player 2 loses.
    If the choices are different, Player 2 wins and Player 1
loses.
    """
    },
    "battle_of_the_sexes": {
        "rules": """
        Two players simultaneously choose between two alternatives.
        Both players prefer to coordinate and choose the same
alternative rather than
        choose different alternatives. However, each player prefers
a different
        coordinated outcome.
        If both players choose Player 1's preferred alternative,
Player 1 receives a
        higher payoff than Player 2. If both players choose Player
2's preferred
        alternative, Player 2 receives a higher payoff than Player 1.
        If the players fail to coordinate, both receive low payoffs.
        """
    },
    "rock_paper_scissors": {
        "rules": """
        Two players simultaneously choose one of three strategies:
Rock, Paper or Scissors.
        Rock beats Scissors.
        Scissors beats Paper.
        Paper beats Rock.
        If both players choose the same strategy, the result is a
draw.
        """
    },
    "shapleys_game": {
        "rules": """
        Shapley's Game is a two-player normal-form game. Each player
simultaneously
        chooses one of three available strategies.
        The main feature of this game is its cyclic best-response
structure. There is no
        single strategy that is best against all possible actions of
the opponent. Instead,
        one profitable response can be followed by another profitable
response of the
        other player, forming a cycle.
        """
    },
    "biased_rock_paper_scissors": {
        "rules": """
        Biased Rock-Paper-Scissors is a modified version of the
classical
        Rock-Paper-Scissors game.
        The basic cyclic structure remains the same:
        Rock beats Scissors.
        Scissors beats Paper.
        Paper beats Rock.
        """
    },
    },

```

```

"rock_paper_scissors_well": {
  "rules": ""
  Rock-Paper-Scissors-Well extends the classical Rock-Paper-
Scissors game by
  adding a fourth strategy: Well.
  The strategies interact as follows:
  Rock beats Scissors.
  Scissors beats Paper.
  Paper beats Rock and Well.
  Well beats Rock and Scissors.
  If both players choose the same strategy, the result is a
draw.
  ""
},
"chicken_game": {
  "rules": ""
  Chicken Game is a two-player game of strategic confrontation.
Each player
  simultaneously chooses between a cautious action and an
aggressive action.
  If one player chooses the aggressive action while the other
chooses the cautious
  action, the aggressive player receives a better payoff.
  If both players choose the cautious action, they avoid
conflict and receive
  moderate payoffs.
  If both players choose the aggressive action, both receive
the worst payoff,
  because mutual aggression leads to a costly conflict.
  ""
},
"coordination": {
  "rules": ""
  Coordination Game is a two-player game where both players
benefit from choosing
  compatible actions.
  Each player simultaneously chooses one of two strategies.
  If both players choose the same strategy, they receive
positive payoffs.
  If they choose different strategies, both receive lower
payoffs.
  The main difficulty is not direct conflict, but coordination.
Both players want
  to match their choices, but they need to select the same
coordinated outcome.
  ""
},
"hawk_dove": {
  "rules": ""
  Two players compete for the same resource. Each player chooses
one of two
  strategies: Hawk or Dove.
  Hawk is aggressive. It performs well against Dove, because
the Hawk player
  takes the resource while the Dove player avoids conflict.
  If both players choose Hawk, conflict occurs and both players
receive a low
  payoff due to the cost of fighting.
  If both players choose Dove, they share the resource
peacefully and receive
  moderate payoffs.

```

```

    ""
  },
  "transitive_6x6": {
    "rules": ""
    Two players simultaneously choose one of six strategies.
    The strategies have a clear order of strength. A strategy
with a higher index
    beats a strategy with a lower index. If both players choose
the same strategy,
    the result is a draw.
    ""
  },
  "cyclical_6x6": {
    "rules": ""
    Two players simultaneously choose one of six strategies.
    Some strategies beat others in a cyclic order. This means
that there is no
    simple global ranking from weakest to strongest strategy.
    A strategy may be strong against one opponent strategy but
weak against another.
    Because of this, the game contains non-transitive strategic
relations.
    ""
  },
  "game_11_20": {
    "rules": ""
    Two players simultaneously choose an integer number from 11
to 20.
    Each player receives the value of the number they selected.
    In addition, if a player chooses exactly one number less than
the opponent,
    that player receives an extra bonus of 20.
    ""
  },
  "random_5x5": {
    "rules": ""
    Two players simultaneously choose one of five strategies.
    The payoff values are generated randomly. Therefore, the game
does not have a
    predefined strategic interpretation or a clear built-in
structure.
    ""
  },
  "zero_sum_5x5": {
    "rules": ""
    Two players simultaneously choose one of five strategies.
    The payoff of Player 2 is always the opposite of the payoff
of Player 1.
    Therefore, one player's gain is exactly the other player's
loss.
    ""
  },
  "combinatorial_auction": {
    "rules": ""
    Two players participate in a simplified combinatorial
auction.
    There are several goods, and each player chooses one bundle
of goods. A bundle
    may contain one good or several goods.
    Each player has a value for each possible bundle. The payoff
depends on the

```

```

value of the chosen bundle.
If both players choose overlapping bundles, a conflict penalty
is applied,
because both players compete for the same goods.
"""
},
"doctor_location": {
  "rules": """
On a small island, different towns, all having the same number
of residents,
are connected by a system of streets.
Two medical doctors, Ann and Beth, are about to settle on the
island. Each
doctor simultaneously chooses one town as the location of
their office.
On average, every islander visits a doctor once a year and
always chooses the
nearest doctor. Distance is measured by the number of roads
one has to travel
from one town to another.
If both doctors are at the same distance from a town, half of
the residents
of that town go to Ann and the other half go to Beth.
If both doctors choose the same town, they split all islanders
equally, so each
doctor receives a payoff of 3.5 towns.
The payoff of each doctor equals the total number of towns'
residents served.
Since every resident is assigned to exactly one doctor, this
is a constant-sum
location game.
"""
},
"restaurant_location": {
  "rules": """
On a small island, different towns, all having the same number
of residents,
are connected by a system of streets.
Two sushi restaurants, Ann's and Beth's, are about to open.
They have the same
quality and price level, so customers choose between them
only according to
location.
Each restaurant simultaneously chooses one town as its
location.
A customer with a sushi restaurant in their home town visits
a restaurant more
often. A customer with no restaurant in the home town, but
with a restaurant
in an adjacent town, may also visit it. Nobody visits a
restaurant that is not
in the home town or in an adjacent town.
If both restaurants are equally available to a customer, the
customer is split
equally between the two restaurants.
The payoff of each restaurant equals the expected number of
customers per year.
Unlike the doctor location game, this game is not necessarily
constant-sum,
because some customers may decide not to visit any restaurant.
"""
}

```

```

    },
    "tic_tac_toe": {
        "rules": ""
        Tic-Tac-Toe is a two-player turn-based game played on a 3 by
3 grid.
        Players take turns placing their marks on empty cells of the
board. One player
        uses one mark, and the other player uses another mark.
        The goal is to place three marks in a row, column or diagonal
before the opponent
        does. If all cells are filled and no player has completed
such a line, the game
        ends in a draw.
        ""
    },
    "hex": {
        "rules": ""
        Hex is a two-player connection game played on a hexagonal
grid.
        Players take turns placing their pieces on empty cells. Each
player tries to
        connect two opposite sides of the board with a continuous
chain of their own
        pieces.
        One player attempts to connect one pair of opposite sides,
while the other player
        attempts to connect the other pair.
        Unlike many board games, Hex cannot end in a draw when the
board is completely
        filled.
        ""
    },
    "connect_four": {
        "rules": ""
        Connect Four is a two-player turn-based game played on a
vertical grid.
        Players take turns dropping pieces into columns. A piece falls
to the lowest
        available position in the selected column.
        The goal is to connect four of one's own pieces in a row. The
connection can be
        horizontal, vertical or diagonal.
        The game ends when one player forms such a line or when the
board becomes full.
        ""
    },
    "go": {
        "rules": ""
        Go is a two-player board game played on a grid. Players take
turns placing stones
        on empty intersections of the board.
        The goal is to control territory and capture the opponent's
stones by surrounding
        them. Stones or groups of stones without liberties can be
removed from the board.
        The full game of Go is highly complex. In this project, a
small OpenSpiel version
        is used so that it can be analyzed computationally.
        ""
    },
    "quoridor": {

```

```

        "rules": """
Quoridor is a two-player board game where each player tries
to move their pawn
to the opposite side of the board.
On each turn, a player can either move their pawn or place a
wall. Walls block
movement paths and can be used to slow down the opponent.
The strategic difficulty of the game comes from balancing
progress toward one's
own goal with blocking the opponent's possible routes.
        """
    },
    "markov_soccer": {
        "rules": """
Markov Soccer is a two-player stochastic game that models a
simplified soccer
environment.
Players move in a small grid-like field and try to score by
reaching the opponent's
goal area while controlling the ball.
The state of the game changes over time, so the result depends
not only on a single
move, but on a sequence of decisions. The game may include
simultaneous actions,
movement conflicts and changes of ball possession.
        """
    },
}
return game_information.get(
    game_key,
    {
        "condition": f"""
{game_name} is a user-defined normal-form game.
        """,
        "rules": """
The user specifies the payoff matrix manually. Rows correspond
to Player 1
strategies, columns correspond to Player 2 strategies, and
each cell contains
payoffs for both players.
The application analyzes the entered payoff matrix using best
responses,
Nash equilibria, response graphs, Alpha-Rank and visual
methods.
        """
    }
)
def payoff_matrix_to_dataframe(game, selected_game_key=None):
    rows = []
    for i in range(game.n_rows):
        row = []

        for j in range(game.n_cols):
            p1, p2 = game.payoff_matrix[i, j]
            row.append(f"({p1:.2f}, {p2:.2f})")
        rows.append(row)
    if selected_game_key in ["doctor_location", "restaurant_location"]:
        columns = [
            f"Beth: town {j + 1}"
            for j in range(game.n_cols)
        ]

```

```

        index = [
            f"Ann: town {i + 1}"
            for i in range(game.n_rows)
        ]
    else:
        columns = [
            f"P2 strategy {j + 1}"
            for j in range(game.n_cols)
        ]
        index = [
            f"P1 strategy {i + 1}"
            for i in range(game.n_rows)
        ]
    return pd.DataFrame(rows, columns=columns, index=index)

def best_responses_to_dataframe(game, selected_game_key=None):
    best_responses_p1, best_responses_p2 = find_best_responses(game)
    rows = []
    if selected_game_key in ["doctor_location", "restaurant_location"]:
        player_1_name = "Ann"
        player_2_name = "Beth"
        strategy_name = "town"
    else:
        player_1_name = "Player 1"
        player_2_name = "Player 2"
        strategy_name = "strategy"
    for strategy, responses in best_responses_p1.items():
        rows.append({
            "Player": player_1_name,
            "Opponent strategy": f"{player_2_name} {strategy_name}"
            {strategy + 1}",
            "Best responses": format_strategy_list(responses)
        })
    for strategy, responses in best_responses_p2.items():
        rows.append({
            "Player": player_2_name,
            "Opponent strategy": f"{player_1_name} {strategy_name}"
            {strategy + 1}",
            "Best responses": format_strategy_list(responses)
        })
    return pd.DataFrame(rows)

def graph_statistics_to_dataframe(graph):
    stats = compute_graph_statistics(graph)
    rows = [
        {
            "Metric": "Number of nodes",
            "Value": stats["number_of_nodes"]
        },
        {
            "Metric": "Number of edges",
            "Value": stats["number_of_edges"]
        },
        {
            "Metric": "Strongly connected components",
            "Value": len(stats["strongly_connected_components"])
        },
        {
            "Metric": "Cycles",
            "Value": len(stats["cycles"])
        },
    ]
    return pd.DataFrame(rows)

```

```

def alpha_rank_to_dataframe(graph):
    scores = calculate_alpha_rank_fallback(graph)
    rows = []
    for node, score in sorted(
        scores.items(),
        key=lambda item: item[1],
        reverse=True
    ):
        rows.append({
            "Strategy profile": format_strategy_profile(node),
            "Alpha-Rank approximation": round(score, 4)
        })
    return pd.DataFrame(rows)

def show_current_matplotlib_figures(width_ratio=0.65):
    figure_numbers = plt.get_fignums()
    if not figure_numbers:
        st.info("No figure was generated.")
        return
    left, center, right = st.columns([
        (1 - width_ratio) / 2,
        width_ratio,
        (1 - width_ratio) / 2
    ])
    with center:
        for figure_number in figure_numbers:
            fig = plt.figure(figure_number)
            st.pyplot(fig, width="stretch")
    plt.close("all")

def show_saved_images(game_key):
    game_dir = OUTPUT_DIR / make_safe_filename(game_key)
    if not game_dir.exists():
        st.info("No saved images found for this game yet.")
        return
    image_files = sorted(list(game_dir.glob("*.png")))
    if not image_files:
        st.info("No saved images found for this game yet.")
        return
    for image_file in image_files:
        st.image(str(image_file), caption=image_file.name)

def render_location_graph_reference():
    graph = create_location_graph()
    pos = {
        7: (0.0, 1.0),
        2: (1.0, 1.35),
        3: (2.0, 1.0),
        4: (2.0, 0.0),
        5: (1.0, -0.35),
        6: (0.0, 0.0),
        1: (1.0, 0.45),
    }
    label_pos = {
        1: (0.84, 0.48),
        2: (0.92, 1.58),
        3: (2.14, 1.00),
        4: (2.14, -0.02),
        5: (0.98, -0.58),
        6: (-0.18, -0.02),
        7: (-0.18, 1.00),
    }
    fig, ax = plt.subplots(figsize=(5.6, 4.6), dpi=150)
    nx.draw_networkx_edges(

```

```

        graph,
        pos,
        ax=ax,
        width=1.8,
        edge_color="black"
    )
    nx.draw_networkx_nodes(
        graph,
        pos,
        ax=ax,
        node_color="white",
        edgecolors="black",
        linewidths=1.4,
        node_size=320
    )
    nx.draw_networkx_labels(
        graph,
        label_pos,
        ax=ax,
        labels={node: str(node) for node in graph.nodes()},
        font_size=13,
        font_color="#8B5A2B",
        font_weight="bold"
    )
    ax.set_title("Reference graph for location games", fontsize=11,
pad=14)
    ax.set_axis_off()
    ax.set_xlim(-0.45, 2.45)
    ax.set_ylim(-0.75, 1.85)
    plt.tight_layout(pad=1.2)
    left, center, right = st.columns([1, 2, 1])
    with center:
        st.pyplot(fig, width="stretch")
    plt.close(fig)
def create_custom_game_from_input():
    st.subheader("Custom Game Input")
    st.markdown(
        """
        Here you can enter your own two-player normal-form game.
        Rows correspond to Player 1 strategies, columns correspond to
Player 2 strategies.
        """
    )

    game_name = st.text_input(
        "Game name",
        value="Custom Game"
    )

    col1, col2 = st.columns(2)
    with col1:
        n_rows = st.number_input(
            "Number of Player 1 strategies",
            min_value=2,
            max_value=10,
            value=3,
            step=1
        )
    with col2:
        n_cols = st.number_input(
            "Number of Player 2 strategies",
            min_value=2,

```

```

        max_value=10,
        value=3,
        step=1
    )
    n_rows = int(n_rows)
    n_cols = int(n_cols)
    row_labels = [f"P1_{i + 1}" for i in range(n_rows)]
    col_labels = [f"P2_{j + 1}" for j in range(n_cols)]
    default_p1 = pd.DataFrame(
        np.zeros((n_rows, n_cols)),
        index=row_labels,
        columns=col_labels
    )
    default_p2 = pd.DataFrame(
        np.zeros((n_rows, n_cols)),
        index=row_labels,
        columns=col_labels
    )
    st.markdown("### Player 1 payoff matrix")
    payoff_p1_df = st.data_editor(
        default_p1,
        width="stretch",
        key=f"custom_p1_{n_rows}_{n_cols}"
    )
    st.markdown("### Player 2 payoff matrix")
    payoff_p2_df = st.data_editor(
        default_p2,
        width="stretch",
        key=f"custom_p2_{n_rows}_{n_cols}"
    )
    try:
        payoff_p1 = payoff_p1_df.astype(float).to_numpy()
        payoff_p2 = payoff_p2_df.astype(float).to_numpy()
    except ValueError:
        st.error("Please enter only numeric payoff values.")
        return None
    payoff_matrix = np.zeros((n_rows, n_cols, 2))
    payoff_matrix[:, :, 0] = payoff_p1
    payoff_matrix[:, :, 1] = payoff_p2
    game = NormalFormGame(
        payoff_matrix,
        name=game_name
    )
    st.markdown("### Save custom game")
    if st.button(
        "Save custom game and add to landscape",
        type="primary"
    ):
        saved_key = save_custom_game(game)
        st.success(
            f"Game saved as `{saved_key}`. "
            "It is now available in Single Game Analysis and Games
Landscape."
        )
    return game
def draw_games_landscape_3d(landscape_df, explained_variance):
    coordinate_columns = landscape_df.attrs.get(
        "coordinate_columns",
        ["PC1", "PC2", "PC3"]
    )
    reduction_method = landscape_df.attrs.get(

```

```

        "reduction_method",
        "PCA"
    )
    if len(coordinate_columns) < 3:
        st.warning("3D landscape requires at least three dimensions.")
        return
    x_column = coordinate_columns[0]
    y_column = coordinate_columns[1]
    z_column = coordinate_columns[2]
    plot_df = landscape_df.copy().reset_index(drop=True)
    plot_df["id"] = plot_df.index + 1
    if "game_cluster" not in plot_df.columns:
        plot_df["game_cluster"] = 1
    plot_df["game_cluster"] = plot_df["game_cluster"].astype(int)
    plot_df["game_cluster_label"] = plot_df["game_cluster"].apply(
        lambda cluster: f"Cluster {cluster}"
    )
    if reduction_method == "PCA" and explained_variance is not None:
        axis_labels = {
            x_column: f"{x_column} ({explained_variance[0] * 100:.1f}%)",
            y_column: f"{y_column} ({explained_variance[1] * 100:.1f}%)",
            z_column: f"{z_column} ({explained_variance[2] * 100:.1f}%)",
        }
    else:
        axis_labels = {
            x_column: x_column,
            y_column: y_column,
            z_column: z_column,
        }
    fig = px.scatter_3d(
        plot_df,
        x=x_column,
        y=y_column,
        z=z_column,
        color="game_cluster_label",
        text="id",
        hover_name="game_key",
        hover_data={
            "game": True,
            "game_cluster": True,
            x_column: ":.3f",
            y_column: ":.3f",
            z_column: ":.3f",
            "id": False,
            "game_key": False,
            "game_cluster_label": False,
        },
        title=f"3D {reduction_method} Landscape of Games with Clusters",
        labels={
            x_column: axis_labels[x_column],
            y_column: axis_labels[y_column],
            z_column: axis_labels[z_column],
            "game_cluster_label": "Game clusters",
        },
    )
    fig.update_traces(
        marker=dict(
            size=7,
            line=dict(
                width=1,
                color="DarkSlateGrey"
            )
        )
    )

```

```

        )
    ),
    textposition="top center"
)
fig.update_layout(
    height=760,
    legend_title_text="Game clusters",
    margin=dict(l=0, r=0, t=45, b=0),
    scene=dict(
        xaxis_title=axis_labels[x_column],
        yaxis_title=axis_labels[y_column],
        zaxis_title=axis_labels[z_column],
    )
)
chart_column, list_column = st.columns([3, 1])
with chart_column:
    st.plotly_chart(fig, width="stretch")
with list_column:
    st.markdown("### Game list")
    game_list_columns = [
        "id",
        "game_key",
        "game_cluster"
    ]
    if "game_type" in plot_df.columns:
        game_list_columns.insert(2, "game_type")
    game_list_df = plot_df[game_list_columns].copy()
    game_list_df = game_list_df.rename(
        columns={
            "id": "ID",
            "game_key": "Game",
            "game_type": "Type",
            "game_cluster": "Cluster"
        }
    )
    st.dataframe(
        game_list_df,
        width="stretch",
        height=760,
        hide_index=True
    )
def classify_game_type(game_key):
    game_key = str(game_key).lower()
    if "location" in game_key:
        return "location game"
    if "rock_paper_scissors" in game_key or "rps" in game_key:
        return "cyclic game"
    if "shapley" in game_key:
        return "cyclic game"
    if "cyclical" in game_key:
        return "cyclic game"
    if "transitive" in game_key:
        return "transitive game"
    if "coordination" in game_key:
        return "coordination game"
    if "battle_of_the_sexes" in game_key:
        return "coordination game"
    if "stag_hunt" in game_key:
        return "coordination game"
    if "matching_pennies" in game_key:
        return "zero-sum game"

```

```

if "zero_sum" in game_key:
    return "zero-sum game"
if "auction" in game_key:
    return "auction game"
if "random" in game_key:
    return "random game"
if game_key in [
    "tic_tac_toe",
    "hex",
    "connect_four",
    "go",
    "quoridor",
    "markov_soccer"
]:
    return "empirical OpenSpiel game"
return "general game"
def get_top_pca_metrics(landscape_df, top_n=5):
    pca_metric_influence_df = landscape_df.attrs.get(
        "pca_metric_influence"
    )
    if pca_metric_influence_df is None:
        return []
    if "Total influence in PCA space (%)" not in
pca_metric_influence_df.columns:
        return []
    top_metrics_df = pca_metric_influence_df.sort_values(
        by="Total influence in PCA space (%)",
        ascending=False
    ).head(top_n)
    top_metrics = []
    for _, row in top_metrics_df.iterrows():
        top_metrics.append(
            (
                row["Metric"],
                row["Total influence in PCA space (%)"]
            )
        )
    return top_metrics
def generate_landscape_interpretation(
    landscape_df,
    explained_variance,
    reduction_method
):
    lines = []
    number_of_games = len(landscape_df)
    lines.append(
        f"The landscape was built for **{number_of_games} games**."
    )
    if reduction_method == "PCA" and explained_variance is not None:
        total_variance = float(np.sum(explained_variance) * 100)
        component_parts = []
        for index, value in enumerate(explained_variance):
            component_parts.append(
                f"PC{index + 1}: {value * 100:.2f}%"
            )
        lines.append(
            "The selected PCA components explain **"
            f"{total_variance:.2f}%** of the total variance "
            f"({' '.join(component_parts)})."
        )
    top_metrics = get_top_pca_metrics(

```

```

        landscape_df,
        top_n=5
    )
    if top_metrics:
        metric_text = ", ".join(
            [
                f"{metric} ({influence:.2f}%)"
                for metric, influence in top_metrics
            ]
        )
        lines.append(
            "The strongest influence on the PCA landscape comes from:
"
            f"**{metric_text}**."
        )
    else:
        lines.append(
            "UMAP was used as a nonlinear dimensionality reduction method.
"
            "Unlike PCA, it does not provide explained variance or direct
metric "
            "coefficients, so the interpretation is based mainly on
distances "
            "between games and cluster structure."
        )
    if "game_cluster" in landscape_df.columns:
        number_of_clusters = landscape_df["game_cluster"].nunique()
        lines.append(
            f"The games were divided into **{number_of_clusters}
clusters** "
            "according to their graph-based metrics."
        )
    for cluster_id in sorted(landscape_df["game_cluster"].unique()):
        cluster_df = landscape_df[
            landscape_df["game_cluster"] == cluster_id
        ]
        cluster_games = cluster_df["game_key"].tolist()
        game_types = [
            classify_game_type(game_key)
            for game_key in cluster_games
        ]
        type_counts = pd.Series(game_types).value_counts()
        dominant_type = type_counts.index[0]
        games_text = ", ".join(cluster_games)
        lines.append(
            f"**Cluster {int(cluster_id)}** contains "
            f"{len(cluster_games)} game(s): {games_text}. "
            f"The dominant type in this cluster is
**{dominant_type}**."
        )
    return "\n\n".join(lines)

def classify_game_type(game_key):
    game_key = str(game_key).lower()
    if "location" in game_key:
        return "location game"
    if (
        "rock_paper_scissors" in game_key
        or "rps" in game_key
        or "shapley" in game_key
        or "cyclical" in game_key
    ):

```

```

        return "cyclic game"
    if "transitive" in game_key:
        return "transitive game"
    if (
        "coordination" in game_key
        or "battle_of_the_sexes" in game_key
        or "stag_hunt" in game_key
    ):
        return "coordination game"
    if (
        "matching_pennies" in game_key
        or "zero_sum" in game_key
    ):
        return "zero-sum game"
    if "auction" in game_key:
        return "auction game"
    if "random" in game_key:
        return "random game"
    if game_key in [
        "tic_tac_toe",
        "hex",
        "connect_four",
        "go",
        "quoridor",
        "markov_soccer"
    ]:
        return "empirical OpenSpiel game"
    if game_key.startswith("custom_"):
        return "custom game"
    return "general game"
def compute_similar_games(
    feature_df,
    selected_game_key,
    top_n=5
):
    if "game_key" not in feature_df.columns:
        return pd.DataFrame()
    if selected_game_key not in feature_df["game_key"].values:
        return pd.DataFrame()
    excluded_columns = [
        "game_key",
        "game",
        "game_type",
        "game_cluster",
        "PC1",
        "PC2",
        "PC3",
        "UMAP1",
        "UMAP2",
        "UMAP3",
    ]
    feature_columns = [
        column
        for column in feature_df.columns
        if column not in excluded_columns
        and pd.api.types.is_numeric_dtype(feature_df[column])
    ]
    if not feature_columns:
        return pd.DataFrame()
    X = feature_df[feature_columns].copy()
    X = X.replace([np.inf, -np.inf], np.nan)

```

```

X = X.fillna(0)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
selected_positions = np.where(
    feature_df["game_key"].to_numpy() == selected_game_key
)[0]
if len(selected_positions) == 0:
    return pd.DataFrame()
selected_position = selected_positions[0]
selected_vector = X_scaled[selected_position]
distances = np.linalg.norm(
    X_scaled - selected_vector,
    axis=1
)
result_df = feature_df.copy()
result_df["structural_distance"] = distances
result_df = result_df[
    result_df["game_key"] != selected_game_key
].copy()
if result_df.empty:
    return pd.DataFrame()
result_df["similarity_score (%)"] = (
    1 / (1 + result_df["structural_distance"])
) * 100
result_df = result_df.sort_values(
    by="structural_distance",
    ascending=True
).head(top_n)
result_df.insert(
    0,
    "Rank",
    range(1, len(result_df) + 1)
)
display_columns = [
    "Rank",
    "game_key",
    "game",
]
if "game_type" in result_df.columns:
    display_columns.append("game_type")
if "game_cluster" in result_df.columns:
    display_columns.append("game_cluster")
display_columns.extend([
    "structural_distance",
    "similarity_score (%)"
])
result_df = result_df[display_columns]
result_df = result_df.rename(
    columns={
        "game_key": "Similar game",
        "game": "Game name",
        "game_type": "Game type",
        "game_cluster": "Cluster",
        "structural_distance": "Structural distance",
    }
)
return result_df
def render_similar_games_panel(feature_df, landscape_df):
    st.markdown("### Similar Games")
    st.markdown(
        """

```

This block finds games that are structurally similar to the selected game.

Similarity is calculated using standardized graph-based metrics, not by game names.

Lower structural distance means higher similarity.

"""

```
)
similarity_df = feature_df.copy()
if "game_cluster" in landscape_df.columns:
    cluster_df = landscape_df[
        [
            "game_key",
            "game_cluster"
        ]
    ].copy()
    similarity_df = similarity_df.merge(
        cluster_df,
        on="game_key",
        how="left"
    )
available_games = similarity_df["game_key"].tolist()
selected_game_for_similarity = st.selectbox(
    "Choose a game to find similar games",
    available_games,
    key="similar_game_selectbox"
)
max_top_n = max(
    1,
    min(10, len(available_games) - 1)
)
top_n = st.slider(
    "Number of similar games to show",
    min_value=1,
    max_value=max_top_n,
    value=min(5, max_top_n),
    step=1
)
similar_games_df = compute_similar_games(
    feature_df=similarity_df,
    selected_game_key=selected_game_for_similarity,
    top_n=top_n
)
if similar_games_df.empty:
    st.info("Similar games could not be calculated.")
    return
st.dataframe(
    similar_games_df.round(4),
    width="stretch",
    hide_index=True
)
closest_game = similar_games_df.iloc[0]
st.info(
    f"The closest game to **{selected_game_for_similarity}** is "
    f"**{closest_game['Similar game']}** with structural distance "
    f"**{closest_game['Structural distance']:.4f}**."
)
)
```

```
def render_games_landscape_page(games):
```

```
    st.subheader("Games Landscape")
```

```
    st.markdown(
```

```
        """
```

This section compares several normal-form games as points in a

transformed two-dimensional or three-dimensional landscape. Each game is into a response graph, graph-based features are extracted, and a dimensionality reduction method is used to project games into a low-dimensional space.

```

    """
)
selected_landscape_games = st.multiselect(
    "Choose games for landscape",
    list(games.keys()),
    default=list(games.keys())
)
reduction_method = st.selectbox(
    "Dimensionality reduction method",
    [
        "PCA",
        "UMAP"
    ]
)
landscape_view = st.radio(
    "Choose visualization",
    [
        "2D Landscape",
        "3D Landscape"
    ],
    horizontal=True
)
if landscape_view == "2D Landscape":
    n_components = 2
    min_games_required = 3 if reduction_method == "UMAP" else 2
else:
    n_components = 3
    min_games_required = 4 if reduction_method == "UMAP" else 3
cluster_option = st.selectbox(
    "Number of game clusters",
    [
        "Auto",
        2,
        3,
        4,
        5,
        6
    ],
    index=2
)
if cluster_option == "Auto":
    n_game_clusters = None
else:
    n_game_clusters = int(cluster_option)
if len(selected_landscape_games) < min_games_required:
    st.warning(
        f"Choose at least {min_games_required} games to build this
landscape."
    )
)
return
landscape_games = {
    key: games[key]
    for key in selected_landscape_games
}
feature_df = build_games_feature_table(landscape_games)

```

```

feature_df["game_type"] = feature_df["game_key"].apply(
    classify_game_type
)
try:
    landscape_df, explained_variance = compute_games_landscape(
        feature_df,
        n_components=n_components,
        n_game_clusters=n_game_clusters,
        method=reduction_method
    )
except Exception as error:
    st.error(str(error))
    return
coordinate_columns = landscape_df.attrs.get(
    "coordinate_columns",
    []
)
st.markdown("### Feature Table")
st.dataframe(
    feature_df.round(3),
    width="stretch"
)
st.markdown("### Game Type Table")
game_type_df = feature_df[
    [
        "game_key",
        "game",
        "game_type"
    ]
].copy()
game_type_df = game_type_df.rename(
    columns={
        "game_key": "Game key",
        "game": "Game name",
        "game_type": "Game type"
    }
)
st.dataframe(
    game_type_df,
    width="stretch",
    hide_index=True
)
st.markdown("### Landscape Coordinates and Game Clusters")
coordinate_display_columns = [
    column
    for column in (
        ["game_key", "game", "game_type", "game_cluster"] +
coordinate_columns
    )
    if column in landscape_df.columns
]
st.dataframe(
    landscape_df[coordinate_display_columns].round(3),
    width="stretch"
)
st.markdown("### Game Cluster Summary")
game_cluster_summary_df = landscape_df.attrs.get(
    "game_cluster_summary"
)
if game_cluster_summary_df is not None:
    st.dataframe(

```

```

        game_cluster_summary_df,
        width="stretch"
    )
else:
    st.info("Game cluster summary is not available.")
render_similar_games_panel(
    feature_df=feature_df,
    landscape_df=landscape_df
)
if reduction_method == "PCA":
    st.markdown("### PCA Explained Variance")
    explained_variance_rows = []
    for index, value in enumerate(explained_variance):
        explained_variance_rows.append({
            "Principal component": f"PC{index + 1}",
            "Explained variance ratio": value,
            "Explained variance (%)": value * 100
        })
    explained_variance_df = pd.DataFrame(explained_variance_rows)
    st.dataframe(
        explained_variance_df.round(4),
        width="stretch"
    )
    total_explained_variance = np.sum(explained_variance) * 100
    st.info(
        f"The selected PCA components explain "
        f"{total_explained_variance:.2f}% of the total variance."
    )
    st.markdown("### PCA Metric Influence")
    pca_metric_influence_df = landscape_df.attrs.get(
        "pca_metric_influence"
    )
    if pca_metric_influence_df is not None:
        st.dataframe(
            pca_metric_influence_df.round(4),
            width="stretch"
        )
    else:
        st.info("PCA metric influence is not available.")
else:
    st.markdown("### UMAP Interpretation")
    st.info(
        """
        UMAP is a nonlinear dimensionality reduction method. Unlike
        PCA, it does not
        provide explained variance ratios or direct metric
        coefficients. Therefore,
        the UMAP landscape is interpreted mainly by distances between
        games and by
        cluster structure: games located close to one another have
        similar graph-based
        feature profiles.
        """
    )
    st.markdown("### Automatic Landscape Interpretation")
    landscape_interpretation = generate_landscape_interpretation(
        landscape_df=landscape_df,
        explained_variance=explained_variance,
        reduction_method=reduction_method
    )
    st.markdown(landscape_interpretation)

```

```

st.markdown("### Landscape Visualization")
if landscape_view == "2D Landscape":
    draw_games_landscape(
        landscape_df,
        explained_variance,
        save=False,
        show=True
    )
    show_current_matplotlib_figures()
else:
    draw_games_landscape_3d(
        landscape_df,
        explained_variance
    )
def render_single_game_analysis_page(
    game,
    n_clusters,
    show_saved=False,
    selected_game_key=None
):
    response_graph = build_response_graph(game)
    st.subheader(f"Selected game: {game.name}")
    game_information = get_game_information(
        selected_game_key,
        game.name
    )
    st.markdown("### Game information")
    with st.expander("Rules", expanded=True):
        st.write(game_information["rules"])
    if selected_game_key in ["doctor_location", "restaurant_location"]:
        st.markdown("### Graph used in the location game")
        render_location_graph_reference()
    tab1, tab2, tab3, tab4 = st.tabs([
        "Payoff Matrix",
        "Text Analysis",
        "Graphs",
        "Alpha-Rank"
    ])
    with tab1:
        st.header("Payoff Matrix")
        payoff_df = payoff_matrix_to_dataframe(
            game,
            selected_game_key
        )
        st.markdown("### Matrix representation")
        st.dataframe(payoff_df, width="stretch")
        st.markdown("### Payoff heatmaps")
        draw_payoff_heatmaps(
            game,
            save=False,
            show=True
        )
        show_current_matplotlib_figures()
    with tab2:
        st.header("Text Analysis")
        st.markdown("### Best Responses")
        st.dataframe(
            best_responses_to_dataframe(game, selected_game_key),
            width="stretch"
        )
        st.markdown("### Pure Nash Equilibria")

```

```

nash_equilibria = find_nash_equilibria(game)
if nash_equilibria:
    nash_df = pd.DataFrame({
        "Pure Nash equilibrium": [
            format_strategy_profile(equilibrium)
            for equilibrium in nash_equilibria
        ]
    })
    st.dataframe(nash_df, width="stretch")
else:
    st.info("No pure Nash equilibria found.")
st.markdown("### Response Graph Statistics")
st.dataframe(
    graph_statistics_to_dataframe(response_graph),
    width="stretch"
)
stats = compute_graph_statistics(response_graph)
with st.expander("In-degrees"):
    st.json({
        format_strategy_profile(node): degree
        for node, degree in stats["in_degrees"].items()
    })
with st.expander("Out-degrees"):
    st.json({
        format_strategy_profile(node): degree
        for node, degree in stats["out_degrees"].items()
    })
with st.expander("Strongly connected components"):
    components = [
        [format_strategy_profile(node) for node in component]
        for component in stats["strongly_connected_components"]
    ]
    st.json(components)
with st.expander("Cycles"):
    cycles = [
        [format_strategy_profile(node) for node in cycle]
        for cycle in stats["cycles"]
    ]
    if len(cycles) <= 10:
        st.json(cycles)
    else:
        st.write(f"Too many cycles to display: {len(cycles)}")
with tab3:
    st.header("Graph Visualizations")
    graph_type = st.radio(
        "Choose graph type",
        [
            "Response Graph",
            "Spectral Graph",
            "Clustered Graph",
            "Contracted Graph",
            "Cycle Distribution"
        ],
        horizontal=True
    )
    if graph_type == "Response Graph":
        draw_response_graph(
            game,
            response_graph,
            save=False,
            show=True

```

```

    )
    elif graph_type == "Spectral Graph":
        draw_spectral_graph(
            game,
            response_graph,
            save=False,
            show=True
        )
    elif graph_type == "Clustered Graph":
        draw_clustered_graph(
            game,
            response_graph,
            n_clusters=n_clusters,
            save=False,
            show=True
        )
    elif graph_type == "Contracted Graph":
        draw_contracted_graph(
            game,
            response_graph,
            n_clusters=n_clusters,
            save=False,
            show=True
        )
    elif graph_type == "Cycle Distribution":
        draw_cycle_distribution(
            game,
            response_graph,
            save=False,
            show=True
        )
    show_current_matplotlib_figures()
    if show_saved and selected_game_key is not None:
        st.markdown("### Saved images")
        show_saved_images(selected_game_key)
    with tab4:
        st.header("Alpha-Rank Approximation")
        st.markdown("### Alpha-Rank table")
        alpha_df = alpha_rank_to_dataframe(response_graph)
        st.dataframe(
            alpha_df,
            width="stretch"
        )
        st.markdown("### Alpha-Rank distribution")
        draw_alpha_rank_distribution(
            game,
            response_graph,
            save=False,
            show=True
        )
    show_current_matplotlib_figures()
def main():
    st.title("Visual Environment for Empirical Analysis of Normal-Form
Games")
    st.markdown(
        """
        This application provides visual and graph-based analysis of two-
player
normal-form games. It supports payoff matrix visualization,
response graph

```

```

        construction, Nash equilibrium detection, Alpha-Rank
approximation, clustering,
        contracted graph representation and landscape analysis.
"""

```

```

)
initialize_custom_games()
predefined_games = get_predefined_games()
games = {
    **predefined_games,
    **st.session_state.custom_games
}
with st.sidebar:
    st.header("Settings")
    analysis_mode = st.radio(
        "Analysis mode",
        [
            "Single Game Analysis",
            "Custom Game Input",
            "Games Landscape"
        ]
    )
    selected_game_key = st.selectbox(
        "Choose a game",
        list(games.keys())
    )
    n_clusters_option = st.selectbox(
        "Number of clusters",
        ["Auto", 2, 3, 4, 5]
    )
    if n_clusters_option == "Auto":
        n_clusters = None
    else:
        n_clusters = int(n_clusters_option)
    show_saved = st.checkbox(
        "Show saved images from outputs folder",
        value=False
    )
    st.divider()
    render_custom_games_management_panel()
    if analysis_mode == "Games Landscape":
        render_games_landscape_page(games)
    elif analysis_mode == "Custom Game Input":
        custom_game = create_custom_game_from_input()
        if custom_game is not None:
            render_single_game_analysis_page(
                game=custom_game,
                n_clusters=n_clusters,
                show_saved=False,
                selected_game_key=None
            )
    else:
        selected_game = games[selected_game_key]
        render_single_game_analysis_page(
            game=selected_game,
            n_clusters=n_clusters,
            show_saved=show_saved,
            selected_game_key=selected_game_key
        )
if __name__ == "__main__":
    main()

```

A.2 Код файла src/games.py

```

import numpy as np
import networkx as nx
from src.game_model import NormalFormGame
try:
    import pyspiel
    OPENSPIEL_AVAILABLE = True
except ImportError:
    pyspiel = None
    OPENSPIEL_AVAILABLE = False
def create_prisoners_dilemma():
    payoff_matrix = [
        [[3, 3], [0, 5]],
        [[5, 0], [1, 1]],
    ]
    return NormalFormGame(payoff_matrix, name="Prisoner's Dilemma")
def create_matching_pennies():
    payoff_matrix = [
        [[1, -1], [-1, 1]],
        [[-1, 1], [1, -1]],
    ]
    return NormalFormGame(payoff_matrix, name="Matching Pennies")
def create_battle_of_the_sexes():
    payoff_matrix = [
        [[3, 2], [0, 0]],
        [[0, 0], [2, 3]],
    ]
    return NormalFormGame(payoff_matrix, name="Battle of the Sexes")
def create_rock_paper_scissors():
    payoff_matrix = [
        [[0, 0], [-1, 1], [1, -1]],
        [[1, -1], [0, 0], [-1, 1]],
        [[-1, 1], [1, -1], [0, 0]],
    ]
    return NormalFormGame(payoff_matrix, name="Rock-Paper-Scissors")
def create_hawk_dove_game():
    payoff_matrix = [
        [[-1, -1], [4, 0]],
        [[0, 4], [2, 2]],
    ]
    return NormalFormGame(payoff_matrix, name="Hawk-Dove Game")
def create_random_game(size=5, seed=42):
    rng = np.random.default_rng(seed)
    payoff_matrix = rng.integers(
        low=-5,
        high=6,
        size=(size, size, 2)
    )
    return NormalFormGame(payoff_matrix, name=f"Random {size}x{size}
Game")
def create_zero_sum_game(size=5, seed=42):
    rng = np.random.default_rng(seed)
    p1_payoffs = rng.integers(
        low=-5,
        high=6,
        size=(size, size)
    )
    p2_payoffs = -p1_payoffs
    payoff_matrix = np.stack([p1_payoffs, p2_payoffs], axis=2)
    return NormalFormGame(payoff_matrix, name=f"Zero-Sum {size}x{size}
Game")

```

```

def create_11_20_game():
    strategies = list(range(11, 21))
    size = len(strategies)
    payoff_matrix = np.zeros((size, size, 2))
    for i, choice_p1 in enumerate(strategies):
        for j, choice_p2 in enumerate(strategies):
            payoff_p1 = choice_p1
            payoff_p2 = choice_p2
            if choice_p1 == choice_p2 - 1:
                payoff_p1 += 20
            if choice_p2 == choice_p1 - 1:
                payoff_p2 += 20
            payoff_matrix[i, j, 0] = payoff_p1
            payoff_matrix[i, j, 1] = payoff_p2
    return NormalFormGame(payoff_matrix, name="11-20 Game")

def create_transitive_game(size=6):
    payoff_matrix = np.zeros((size, size, 2))
    for i in range(size):
        for j in range(size):
            if i > j:
                payoff_matrix[i, j] = [1, -1]
            elif i < j:
                payoff_matrix[i, j] = [-1, 1]
            else:
                payoff_matrix[i, j] = [0, 0]
    return NormalFormGame(payoff_matrix, name=f"Transitive {size}x{size}
Game")

def create_cyclical_game(size=6):
    payoff_matrix = np.zeros((size, size, 2))
    for i in range(size):
        for j in range(size):
            if i == j:
                payoff_matrix[i, j] = [0, 0]
            elif (i - j) % size == 1:
                payoff_matrix[i, j] = [1, -1]
            elif (j - i) % size == 1:
                payoff_matrix[i, j] = [-1, 1]
            else:
                if i > j:
                    payoff_matrix[i, j] = [0.5, -0.5]
                else:
                    payoff_matrix[i, j] = [-0.5, 0.5]
    return NormalFormGame(payoff_matrix, name=f"Cyclical {size}x{size}
Game")

def create_combinatorial_auction_game():
    bundles = [
        {"A"},
        {"B"},
        {"C"},
        {"A", "B"},
        {"B", "C"},
        {"A", "C"},
        {"A", "B", "C"},
    ]
    values_p1 = [3, 2, 2, 7, 5, 6, 9]
    values_p2 = [2, 3, 2, 5, 7, 6, 9]
    size = len(bundles)
    payoff_matrix = np.zeros((size, size, 2))
    for i, bundle_p1 in enumerate(bundles):
        for j, bundle_p2 in enumerate(bundles):
            overlap = bundle_p1.intersection(bundle_p2)

```

```

        payoff_p1 = values_p1[i]
        payoff_p2 = values_p2[j]
        if overlap:
            penalty = 2 * len(overlap)
            payoff_p1 -= penalty
            payoff_p2 -= penalty
            if bundle_p1.issuperset(bundle_p2) and bundle_p1 !=
bundle_p2:
                payoff_p1 += 1
            if bundle_p2.issuperset(bundle_p1) and bundle_p1 !=
bundle_p2:
                payoff_p2 += 1
            payoff_matrix[i, j, 0] = payoff_p1
            payoff_matrix[i, j, 1] = payoff_p2
    return NormalFormGame(payoff_matrix, name="Combinatorial Auction
Game")

def create_location_graph():
    graph = nx.Graph()
    graph.add_nodes_from([1, 2, 3, 4, 5, 6, 7])
    graph.add_edges_from([
        (7, 2),
        (2, 3),
        (3, 4),
        (4, 5),
        (5, 6),
        (6, 7),
        (1, 2),
        (1, 3),
        (1, 5),
    ])
    return graph

def compute_all_pairs_distances(graph):
    distances = dict(nx.all_pairs_shortest_path_length(graph))
    return distances

def create_doctor_location_game():
    graph = create_location_graph()
    distances = compute_all_pairs_distances(graph)
    locations = list(graph.nodes())
    size = len(locations)
    payoff_matrix = np.zeros((size, size, 2))
    for i, doctor_1_location in enumerate(locations):
        for j, doctor_2_location in enumerate(locations):
            payoff_1 = 0.0
            payoff_2 = 0.0
            for resident_location in locations:
                distance_to_doctor_1 =
distances[resident_location][doctor_1_location]
                distance_to_doctor_2 =
distances[resident_location][doctor_2_location]
                if distance_to_doctor_1 < distance_to_doctor_2:
                    payoff_1 += 1.0
                elif distance_to_doctor_2 < distance_to_doctor_1:
                    payoff_2 += 1.0
                else:
                    payoff_1 += 0.5
                    payoff_2 += 0.5
            payoff_matrix[i, j, 0] = payoff_1
            payoff_matrix[i, j, 1] = payoff_2
    return NormalFormGame(payoff_matrix, name="Doctor Location Game")

def create_restaurant_location_game():
    graph = create_location_graph()

```

```

distances = compute_all_pairs_distances(graph)
locations = list(graph.nodes())
size = len(locations)
payoff_matrix = np.zeros((size, size, 2))
maximum_acceptable_distance = 1
for i, restaurant_1_location in enumerate(locations):
    for j, restaurant_2_location in enumerate(locations):
        payoff_1 = 0.0
        payoff_2 = 0.0
        for customer_location in locations:
            distance_to_restaurant_1 =
distances[customer_location][restaurant_1_location]
            distance_to_restaurant_2 =
distances[customer_location][restaurant_2_location]
            restaurant_1_available = (
                distance_to_restaurant_1 <=
maximum_acceptable_distance
            )
            restaurant_2_available = (
                distance_to_restaurant_2 <=
maximum_acceptable_distance
            )
            if not restaurant_1_available and not
restaurant_2_available:
                continue
            if restaurant_1_available and not restaurant_2_available:
                payoff_1 += 1.0
            elif restaurant_2_available and not
restaurant_1_available:
                payoff_2 += 1.0
            else:
                if distance_to_restaurant_1 <
distance_to_restaurant_2:
                    payoff_1 += 1.0
                elif distance_to_restaurant_2 <
distance_to_restaurant_1:
                    payoff_2 += 1.0
                else:
                    payoff_1 += 0.5
                    payoff_2 += 0.5
            payoff_matrix[i, j, 0] = payoff_1
            payoff_matrix[i, j, 1] = payoff_2
        return NormalFormGame(payoff_matrix, name="Restaurant Location Game")
def load_openspiel_matrix_game(game_id, display_name=None):
    if not OPENSPIEL_AVAILABLE:
        raise ImportError(
            "OpenSpiel is not installed. Install it with: pip install
open_spiel"
        )
    os_game = pyspiel.load_game(game_id)
    if os_game.num_players() != 2:
        raise ValueError(
            f"Only two-player games are supported. "
            f"Game {game_id} has {os_game.num_players()} players."
        )
    state = os_game.new_initial_state()
    if not state.is_simultaneous_node():
        raise ValueError(
            f"Game {game_id} is not a one-shot simultaneous matrix game."
        )
    actions_p1 = list(state.legal_actions(0))

```

```

actions_p2 = list(state.legal_actions(1))
payoff_matrix = np.zeros((len(actions_p1), len(actions_p2), 2))
for i, action_p1 in enumerate(actions_p1):
    for j, action_p2 in enumerate(actions_p2):
        state = os_game.new_initial_state()
        state.apply_actions([action_p1, action_p2])
        if not state.is_terminal():
            raise ValueError(
                f"Game {game_id} did not terminate after one
simultaneous move."
            )
        returns = state.returns()
        payoff_matrix[i, j, 0] = returns[0]
        payoff_matrix[i, j, 1] = returns[1]
if display_name is None:
    display_name = f"OpenSpiel {game_id}"
return NormalFormGame(
    payoff_matrix,
    name=display_name
)
def get_openspiel_matrix_games():
    if not OPENSPIEL_AVAILABLE:
        return {}
    openspiel_game_specs = [
        (
            "shapleys_game",
            "matrix_shapleys_game",
            "Shapley's Game"
        ),
        (
            "biased_rock_paper_scissors",
            "matrix_brps",
            "Biased Rock-Paper-Scissors"
        ),
        (
            "rock_paper_scissors_well",
            "matrix_rpsw",
            "Rock-Paper-Scissors-Well"
        ),
        (
            "chicken_game",
            "matrix_cd",
            "Chicken Game"
        ),
        (
            "coordination",
            "matrix_coordination",
            "Coordination Game"
        ),
    ]
    games = {}
    for game_key, game_id, display_name in openspiel_game_specs:
        try:
            games[game_key] = load_openspiel_matrix_game(
                game_id,
                display_name
            )
        except Exception as error:
            print(f"Could not load {game_id} from OpenSpiel: {error}")
    return games
def try_load_openspiel_game(game_id_candidates):

```

```

if isinstance(game_id_candidates, str):
    game_id_candidates = [game_id_candidates]
last_error = None
for game_id in game_id_candidates:
    try:
        return pyspiel.load_game(game_id), game_id
    except Exception as error:
        last_error = error
raise last_error
def get_default_empirical_policy_specs():
    return [
        {
            "name": "Random policy 1",
            "type": "random",
            "seed": 1
        },
        {
            "name": "Random policy 2",
            "type": "random",
            "seed": 2
        },
        {
            "name": "First legal action",
            "type": "first",
            "seed": 3
        },
        {
            "name": "Last legal action",
            "type": "last",
            "seed": 4
        },
        {
            "name": "Middle legal action",
            "type": "middle",
            "seed": 5
        },
    ]
def choose_action_by_policy(state, player_id, policy_spec, rng):
    legal_actions = list(state.legal_actions(player_id))
    if not legal_actions:
        return None
    policy_type = policy_spec.get("type", "random")
    if policy_type == "first":
        return int(min(legal_actions))
    if policy_type == "last":
        return int(max(legal_actions))
    if policy_type == "middle":
        legal_actions = sorted(legal_actions)
        return int(legal_actions[len(legal_actions) // 2])
    return int(rng.choice(legal_actions))
def sample_chance_action(state, rng):
    chance_outcomes = state.chance_outcomes()
    actions = [outcome[0] for outcome in chance_outcomes]
    probabilities = [outcome[1] for outcome in chance_outcomes]
    return int(rng.choice(actions, p=probabilities))
def simulate_openspiel_match(
    os_game,
    policy_p1,
    policy_p2,
    seed,
    max_steps=500

```

```

):
    rng = np.random.default_rng(seed)
    state = os_game.new_initial_state()
    steps = 0
    while not state.is_terminal() and steps < max_steps:
        if state.is_chance_node():
            action = sample_chance_action(state, rng)
            state.apply_action(action)
        elif state.is_simultaneous_node():
            actions = []
            for player_id in range(os_game.num_players()):
                if player_id == 0:
                    policy = policy_p1
                else:
                    policy = policy_p2
                action = choose_action_by_policy(
                    state,
                    player_id,
                    policy,
                    rng
                )
                actions.append(action)
            state.apply_actions(actions)
        else:
            current_player = state.current_player()
            if current_player < 0:
                break
            if current_player == 0:
                policy = policy_p1
            else:
                policy = policy_p2
            action = choose_action_by_policy(
                state,
                current_player,
                policy,
                rng
            )
            if action is None:
                break
            state.apply_action(action)
        steps += 1
    try:
        returns = state.returns()
    except Exception:
        returns = [0.0, 0.0]
    return np.array(returns[:2], dtype=float)
def load_openspiel_empirical_game(
    game_id_candidates,
    display_name,
    num_matches=8,
    max_steps=500,
    policy_specs=None
):
    if not OPENSPIEL_AVAILABLE:
        raise ImportError(
            "OpenSpiel is not installed. Install it with: pip install
open_spiel"
        )
    os_game, loaded_game_id =
try_load_openspiel_game(game_id_candidates)
    if os_game.num_players() != 2:

```

```

        raise ValueError(
            f"Only two-player games are supported. "
            f"Game {loaded_game_id} has {os_game.num_players()} players."
        )
    if policy_specs is None:
        policy_specs = get_default_empirical_policy_specs()
    number_of_policies = len(policy_specs)
    payoff_matrix = np.zeros((number_of_policies, number_of_policies, 2))
    for i, policy_p1 in enumerate(policy_specs):
        for j, policy_p2 in enumerate(policy_specs):
            total_returns = np.zeros(2)
            for match_index in range(num_matches):
                seed = (
                    (i + 1) * 100000
                    + (j + 1) * 1000
                    + match_index
                )
                total_returns += simulate_openspiel_match(
                    os_game=os_game,
                    policy_p1=policy_p1,
                    policy_p2=policy_p2,
                    seed=seed,
                    max_steps=max_steps
                )
            payoff_matrix[i, j, :] = total_returns / num_matches
    return NormalFormGame(
        payoff_matrix,
        name=f"{display_name} Empirical Game"
    )
def get_openspiel_empirical_games():
    if not OPENSPIEL_AVAILABLE:
        return {}
    openspiel_empirical_specs = [
        (
            "tic_tac_toe",
            ["tic_tac_toe"],
            "Tic-Tac-Toe",
            10,
            200
        ),
        (
            "hex",
            [
                "hex(board_size=3)",
                "hex"
            ],
            "Hex",
            8,
            200
        ),
        (
            "connect_four",
            [
                "connect_four(rows=4,columns=4)",
                "connect_four"
            ],
            "Connect Four",
            6,
            300
        ),
    ]

```

```

        "go",
        [
            "go(board_size=3,komi=0.5)",
            "go(board_size=3)"
        ],
        "Go",
        6,
        300
    ),
    (
        "quoridor",
        [
            "quoridor(board_size=3)",
            "quoridor"
        ],
        "Quoridor",
        4,
        500
    ),
    (
        "markov_soccer",
        [
            "markov_soccer"
        ],
        "Markov Soccer",
        10,
        300
    ),
]
games = {}
for (
    game_key,
    game_id_candidates,
    display_name,
    num_matches,
    max_steps
) in openspiel_empirical_specs:
    try:
        games[game_key] = load_openspiel_empirical_game(
            game_id_candidates=game_id_candidates,
            display_name=display_name,
            num_matches=num_matches,
            max_steps=max_steps
        )
    except Exception as error:
        print(
            f"Could not load empirical OpenSpiel game "
            f"{display_name}: {error}"
        )
return games

def get_predefined_games():
    games = {
        "prisoners_dilemma": create_prisoners_dilemma(),
        "matching_pennies": create_matching_pennies(),
        "battle_of_the_sexes": create_battle_of_the_sexes(),
        "rock_paper_scissors": create_rock_paper_scissors(),
        "hawk_dove": create_hawk_dove_game(),
        "transitive_6x6": create_transitive_game(size=6),
        "cyclical_6x6": create_cyclical_game(size=6),
        "game_11_20": create_11_20_game(),
        "random_5x5": create_random_game(size=5, seed=42),
    }

```

```

        "zero_sum_5x5": create_zero_sum_game(size=5, seed=42),
        "combinatorial_auction": create_combinatorial_auction_game(),
        "doctor_location": create_doctor_location_game(),
        "restaurant_location": create_restaurant_location_game(),
    }
    games.update(get_openspiel_matrix_games())
    games.update(get_openspiel_empirical_games())
    return games

```

A.3 Код файла src/landscape.py

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import networkx as nx
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from src.config import OUTPUT_DIR, FIGURE_DPI
from src.analysis import (
    build_response_graph,
    find_nash_equilibria,
    calculate_alpha_rank_fallback,
)
try:
    import umap.umap_ as umap
    UMAP_AVAILABLE = True
except ImportError:
    umap = None
    UMAP_AVAILABLE = False
COLOR_POINT = "#a8d5c5"
COLOR_POINT_EDGE = "#263238"
COLOR_TEXT = "#263238"
COLOR_GRID = "#d9d9d9"
GAME_COLORS = [
    "#a8d5c5",
    "#f6d98b",
    "#b7d9a8",
    "#f2b6a0",
    "#c9c0e8",
    "#a8cce8",
    "#f4a6a6",
    "#d7eaf2",
    "#a8d5a2",
    "#e6c6a8",
    "#d5b8d8",
    "#b9d7ea",
    "#f7c7a3",
    "#c8e6c9",
    "#ffe0b2",
    "#d1c4e9",
]
plt.rcParams.update({
    "axes.titlesize": 12,
    "axes.labelsize": 10,
    "xtick.labelsize": 9,
    "ytick.labelsize": 9,
    "text.color": COLOR_TEXT,
    "axes.labelcolor": COLOR_TEXT,
    "xtick.color": COLOR_TEXT,
    "ytick.color": COLOR_TEXT,
})
def make_safe_filename(text):

```

```

filename = text.lower()
filename = filename.replace(" ", "_")
filename = filename.replace("'", "")
filename = filename.replace("-", "_")
filename = filename.replace(":", "")
filename = filename.replace("/", "_")
return filename
def calculate_entropy(values):
    values = np.array(values, dtype=float)
    total = values.sum()
    if total <= 0:
        return 0.0
    probabilities = values / total
    probabilities = probabilities[probabilities > 0]
    entropy = -np.sum(probabilities * np.log(probabilities))
    return float(entropy)
def extract_game_features(game):
    graph = build_response_graph(game)
    n_nodes = graph.number_of_nodes()
    n_edges = graph.number_of_edges()
    if n_nodes == 0:
        density = 0.0
        mean_in_degree = 0.0
        mean_out_degree = 0.0
        max_in_degree = 0.0
        max_out_degree = 0.0
    else:
        density = nx.density(graph)
        in_degrees = dict(graph.in_degree())
        out_degrees = dict(graph.out_degree())
        mean_in_degree = np.mean(list(in_degrees.values()))
        mean_out_degree = np.mean(list(out_degrees.values()))
        max_in_degree = max(in_degrees.values())
        max_out_degree = max(out_degrees.values())
    strongly_connected_components =
list(nx.strongly_connected_components(graph))
cycles = list(nx.simple_cycles(graph))
nash_equilibria = find_nash_equilibria(game)
alpha_scores = calculate_alpha_rank_fallback(graph)
alpha_entropy = calculate_entropy(list(alpha_scores.values()))
if n_nodes > 0:
    normalized_edges = n_edges / n_nodes
    normalized_cycles = len(cycles) / n_nodes
    normalized_scc = len(strongly_connected_components) / n_nodes
    normalized_nash = len(nash_equilibria) / n_nodes
else:
    normalized_edges = 0.0
    normalized_cycles = 0.0
    normalized_scc = 0.0
    normalized_nash = 0.0
features = {
    "game": game.name,
    "rows": game.n_rows,
    "columns": game.n_cols,
    "profiles": n_nodes,
    "response_edges": n_edges,
    "density": density,
    "mean_in_degree": mean_in_degree,
    "mean_out_degree": mean_out_degree,
    "max_in_degree": max_in_degree,
    "max_out_degree": max_out_degree,

```

```

        "pure_nash_equilibria": len(nash_equilibria),
        "strongly_connected_components":
len(strongly_connected_components),
        "cycles": len(cycles),
        "alpha_rank_entropy": alpha_entropy,
        "normalized_edges": normalized_edges,
        "normalized_cycles": normalized_cycles,
        "normalized_scc": normalized_scc,
        "normalized_nash": normalized_nash,
    }
    return features

def build_games_feature_table(games):
    rows = []
    for game_key, game in games.items():
        row = extract_game_features(game)
        row["game_key"] = game_key
        rows.append(row)
    feature_df = pd.DataFrame(rows)
    return feature_df

def compute_pca_metric_influence(
    feature_columns,
    pca,
    explained_variance
):
    influence_data = {
        "Metric": feature_columns
    }
    total_influence = np.zeros(len(feature_columns))
    explained_sum = np.sum(explained_variance)
    for component_index in range(len(explained_variance)):
        component_name = f"PC{component_index + 1}"
        coefficients = pca.components_[component_index]
        squared_coefficients = coefficients ** 2
        squared_sum = np.sum(squared_coefficients)
        if squared_sum == 0:
            influence = np.zeros_like(squared_coefficients)
        else:
            influence = squared_coefficients / squared_sum * 100
            influence_data[f"{component_name} coefficient"] = coefficients
            influence_data[f"{component_name} influence (%)"] = influence
            if explained_sum != 0:
                total_influence += (
                    influence * explained_variance[component_index]
                ) / explained_sum
    influence_data["Total influence in PCA space (%)"] = total_influence
    pca_metric_influence_df = pd.DataFrame(influence_data)
    pca_metric_influence_df = pca_metric_influence_df.sort_values(
        by="Total influence in PCA space (%)",
        ascending=False
    ).reset_index(drop=True)
    return pca_metric_influence_df

def compute_games_landscape(
    feature_df,
    n_components=2,
    n_game_clusters=3,
    method="PCA"
):
    feature_columns = [
        column
        for column in feature_df.columns

```

```

        if column not in ["game_key", "game"]
        and pd.api.types.is_numeric_dtype(feature_df[column])
    ]
    if not feature_columns:
        raise ValueError("No numeric feature columns are available for
landscape computation.")
    X = feature_df[feature_columns].copy()
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    number_of_games = len(feature_df)
    max_components = min(
        n_components,
        X_scaled.shape[0],
        X_scaled.shape[1]
    )
    if max_components < 1:
        raise ValueError("At least one PCA/UMAP component is required.")
    method = method.upper()
    landscape_df = feature_df.copy()
    if method == "PCA":
        pca = PCA(n_components=max_components)
        coordinates = pca.fit_transform(X_scaled)
        coordinate_columns = []
        for component_index in range(max_components):
            column_name = f"PC{component_index + 1}"
            landscape_df[column_name] = coordinates[:, component_index]
            coordinate_columns.append(column_name)
        explained_variance = pca.explained_variance_ratio_
        pca_metric_influence_df = compute_pca_metric_influence(
            feature_columns=feature_columns,
            pca=pca,
            explained_variance=explained_variance
        )
        landscape_df.attrs["pca_metric_influence"] =
pca_metric_influence_df
        landscape_df.attrs["explained_variance"] = explained_variance
        landscape_df.attrs["reduction_method"] = "PCA"
        landscape_df.attrs["coordinate_columns"] = coordinate_columns
    elif method == "UMAP":
        if not UMAP_AVAILABLE:
            raise ImportError(
                "UMAP is not installed. Install it with: pip install umap-
learn"
            )
        if number_of_games < 3:
            raise ValueError(
                "UMAP requires at least three games to build a stable
embedding."
            )
        n_neighbors = min(
            10,
            max(2, number_of_games - 1)
        )
        reducer = umap.UMAP(
            n_components=max_components,
            n_neighbors=n_neighbors,
            min_dist=0.15,
            metric="euclidean",
            random_state=42
        )
        coordinates = reducer.fit_transform(X_scaled)

```

```

coordinate_columns = []
for component_index in range(max_components):
    column_name = f"UMAP{component_index + 1}"
    landscape_df[column_name] = coordinates[:, component_index]
    coordinate_columns.append(column_name)
    explained_variance = None
    landscape_df.attrs["pca_metric_influence"] = None
    landscape_df.attrs["explained_variance"] = None
    landscape_df.attrs["reduction_method"] = "UMAP"
    landscape_df.attrs["coordinate_columns"] = coordinate_columns
else:
    raise ValueError(
        f"Unknown dimensionality reduction method: {method}"
    )
if n_game_clusters is None:
    n_game_clusters = min(
        4,
        max(2, int(np.sqrt(number_of_games)))
    )
n_game_clusters = int(n_game_clusters)
n_game_clusters = min(
    n_game_clusters,
    number_of_games
)
if n_game_clusters < 2:
    cluster_labels = np.zeros(number_of_games, dtype=int)
else:
    clustering_model = KMeans(
        n_clusters=n_game_clusters,
        random_state=42,
        n_init=10
    )
    cluster_labels = clustering_model.fit_predict(X_scaled)
    landscape_df["game_cluster"] = cluster_labels + 1
    cluster_summary_df = (
        landscape_df
        .groupby("game_cluster")
        .agg(
            number_of_games=("game_key", "count"),
            games=("game_key", lambda values: ", ".join(values))
        )
        .reset_index()
    )
    landscape_df.attrs["game_cluster_summary"] = cluster_summary_df
    return landscape_df, explained_variance
def draw_games_landscape(
    landscape_df,
    explained_variance,
    save=True,
    show=True,
    output_dir=OUTPUT_DIR
):
    reduction_method = landscape_df.attrs.get(
        "reduction_method",
        "PCA"
    )
    coordinate_columns = landscape_df.attrs.get(
        "coordinate_columns",
        ["PC1", "PC2"]
    )
    if len(coordinate_columns) < 2:

```

```

        raise ValueError("At least two coordinate columns are required.")
    x_column = coordinate_columns[0]
    y_column = coordinate_columns[1]
    print("=" * 80)
    print(f"{reduction_method} Games Landscape")
    print("=" * 80)
    if reduction_method == "PCA" and explained_variance is not None:
        print("PCA explained variance:")
        for index, value in enumerate(explained_variance):
            print(f"  PC{index + 1}: {value:.4f}")
        print()
    print("Game coordinates:")
    for _, row in landscape_df.iterrows():
        print(
            f"  {row['game_key']}: "
            f"{x_column} = {row[x_column]:.4f}, "
            f"{y_column} = {row[y_column]:.4f}, "
            f"cluster = {row.get('game_cluster', '-')}"
        )
    print()
    pca_metric_influence_df = landscape_df.attrs.get("pca_metric_influence")
    if pca_metric_influence_df is not None:
        print("PCA metric influence:")
        print(pca_metric_influence_df.round(4).to_string(index=False))
        print()
    plot_df = landscape_df.copy().reset_index(drop=True)
    plot_df["id"] = plot_df.index + 1
    plot_df["plot_x"] = plot_df[x_column]
    plot_df["plot_y"] = plot_df[y_column]
    rounded_coordinates = {}
    for index, row in plot_df.iterrows():
        key = (
            round(row[x_column], 2),
            round(row[y_column], 2)
        )
        if key not in rounded_coordinates:
            rounded_coordinates[key] = 0
        duplicate_number = rounded_coordinates[key]
        rounded_coordinates[key] += 1
        if duplicate_number > 0:
            angle = duplicate_number * 1.2
            radius = 0.12 * duplicate_number
            plot_df.loc[index, "plot_x"] = (
                row[x_column] + radius * np.cos(angle)
            )
            plot_df.loc[index, "plot_y"] = (
                row[y_column] + radius * np.sin(angle)
            )
    plt.figure(figsize=(13, 8), dpi=FIGURE_DPI)
    clusters = sorted(plot_df["game_cluster"].unique())
    cluster_color_map = {
        cluster: GAME_COLORS[(int(cluster) - 1) % len(GAME_COLORS)]
        for cluster in clusters
    }
    for _, row in plot_df.iterrows():
        cluster = row["game_cluster"]
        color = cluster_color_map[cluster]
        plt.scatter(
            row["plot_x"],
            row["plot_y"],

```

```

        s=240,
        color=color,
        edgecolor=COLOR_POINT_EDGE,
        linewidth=1.2,
        alpha=0.9,
        zorder=3
    )
    plt.annotate(
        str(row["id"]),
        xy=(row["plot_x"], row["plot_y"]),
        xytext=(8, 8),
        textcoords="offset points",
        fontsize=10,
        fontweight="bold",
        color=COLOR_TEXT,
        bbox=dict(
            boxstyle="round,pad=0.25",
            fc="white",
            ec=color,
            alpha=0.9
        ),
        zorder=4
    )
for cluster in clusters:
    plt.scatter(
        [],
        [],
        s=160,
        color=cluster_color_map[cluster],
        edgecolor=COLOR_POINT_EDGE,
        label=f"Cluster {int(cluster)}"
    )
plt.axhline(0, color=COLOR_GRID, linewidth=1)
plt.axvline(0, color=COLOR_GRID, linewidth=1)
plt.grid(color=COLOR_GRID, alpha=0.45)
plt.title(
    f"{reduction_method} Landscape of Games with Clusters",
    fontsize=15,
    fontweight="bold",
    color=COLOR_TEXT
)
if reduction_method == "PCA" and explained_variance is not None:
    x_label = f"{x_column} ({explained_variance[0] * 100:.1f}%
variance)"
    y_label = f"{y_column} ({explained_variance[1] * 100:.1f}%
variance)"
else:
    x_label = x_column
    y_label = y_column
plt.xlabel(
    x_label,
    color=COLOR_TEXT
)
plt.ylabel(
    y_label,
    color=COLOR_TEXT
)
plt.legend(
    title="Game clusters",
    loc="upper left",
    bbox_to_anchor=(1.02, 1.0),

```

```

        frameon=True
    )
    legend_lines = []
    for _, row in plot_df.iterrows():
        legend_lines.append(
            f"{int(row['id'])}. {row['game_key']}
C{int(row['game_cluster'])}"
        )
    legend_text = "\n".join(legend_lines)
    plt.gcf().text(
        0.80,
        0.5,
        legend_text,
        fontsize=9,
        va="center",
        ha="left",
        bbox=dict(
            boxstyle="round,pad=0.5",
            fc="white",
            ec=COLOR_GRID,
            alpha=0.9
        )
    )
    plt.tight_layout(rect=[0, 0, 0.76, 1])
    if save:
        output_dir.mkdir(exist_ok=True)
        filename = f"{reduction_method.lower()}_games_landscape.png"
        output_path = output_dir / filename
        plt.savefig(output_path, bbox_inches="tight")
        print(f"Saved games landscape to: {output_path}")
    if show:
        pass
    else:
        plt.close()
def save_landscape_tables(
    feature_df,
    landscape_df,
    output_dir=OUTPUT_DIR
):
    output_dir.mkdir(exist_ok=True)
    reduction_method = landscape_df.attrs.get(
        "reduction_method",
        "landscape"
    ).lower()
    features_path = output_dir /
f"{reduction_method}_games_landscape_features.csv"
    coordinates_path = output_dir /
f"{reduction_method}_games_landscape_coordinates.csv"
    pca_influence_path = output_dir /
f"{reduction_method}_games_landscape_pca_metric_influence.csv"
    cluster_summary_path = output_dir /
f"{reduction_method}_games_landscape_cluster_summary.csv"
    feature_df.to_csv(features_path, index=False)
    landscape_df.to_csv(coordinates_path, index=False)
    pca_metric_influence_df =
landscape_df.attrs.get("pca_metric_influence")
    if pca_metric_influence_df is not None:
        pca_metric_influence_df.to_csv(
            pca_influence_path,
            index=False
        )

```

```

        game_cluster_summary_df
landscape_df.attrs.get("game_cluster_summary")
    if game_cluster_summary_df is not None:
        game_cluster_summary_df.to_csv(
            cluster_summary_path,
            index=False
        )
    print(f"Saved landscape features to: {features_path}")
    print(f"Saved landscape coordinates to: {coordinates_path}")
    if pca_metric_influence_df is not None:
        print(f"Saved PCA metric influence to: {pca_influence_path}")
    if game_cluster_summary_df is not None:
        print(f"Saved game cluster summary to: {cluster_summary_path}")
def run_games_landscape(
    games,
    save=True,
    show=True,
    output_dir=OUTPUT_DIR,
    n_components=2,
    n_game_clusters=3,
    method="PCA"
):
    feature_df = build_games_feature_table(games)
    landscape_df, explained_variance = compute_games_landscape(
        feature_df,
        n_components=n_components,
        n_game_clusters=n_game_clusters,
        method=method
    )
    draw_games_landscape(
        landscape_df,
        explained_variance,
        save=save,
        show=show,
        output_dir=output_dir
    )
    save_landscape_tables(
        feature_df,
        landscape_df,
        output_dir=output_dir
    )
    pca_metric_influence_df
landscape_df.attrs.get("pca_metric_influence")
    game_cluster_summary_df
landscape_df.attrs.get("game_cluster_summary")
    return {
        "features": feature_df,
        "landscape": landscape_df,
        "explained_variance": explained_variance,
        "pca_metric_influence": pca_metric_influence_df,
        "game_cluster_summary": game_cluster_summary_df,
    }

```

ДОДАТОК Б. ІЛЮСТРАТИВНИЙ МАТЕРІАЛ



Об'єкт, предмет і мета роботи

Об'єкт дослідження – процеси емпіричного аналізу та порівняння структурних характеристик ігор у нормальній формі.

Предмет дослідження – методи графового, спектрального та візуального аналізу ігор у нормальній формі.

Мета роботи – дослідити та реалізувати аналітико-візуалізаційний етап емпіричного аналізу ігор у нормальній формі, що передбачає роботу з уже сформованими таблицями виграшів.

Актуальність дослідження

- сучасні дослідження стратегічних і багатоагентних взаємодій потребують не лише аналізу окремих ігор, а й порівняння різних ігрових середовищ;
- традиційний підхід, орієнтований на пошук сильних стратегій або рівноважних станів, не завжди дозволяє зрозуміти внутрішню структуру гри;
- актуальним є дослідження структурних властивостей ігор: стратегічних переходів, циклічності, подібності між іграми та можливості їх класифікації;
- ландшафт ігор дає змогу кількісно порівнювати ігри різного розміру та складності й виявляти групи структурно подібних ігор.

3

Емпіричний аналіз ігор у межах роботи

Емпіричний аналіз ігор – це дослідження стратегічної взаємодії на основі скінченного набору стратегій і відповідних значень вигащів.

Загальні етапи емпіричного аналізу ігор:



У межах цієї роботи розглядається аналітико-візуалізаційний етап: система працює з уже сформованими таблицями вигащів і досліджує їхню структуру.

4

Розроблене програмне середовище

Призначення

аналіз ігор у нормальній формі на основі таблиць виграшів

1

Аналітична обробка

рівноваги Неша, граф відповідей, характеристики графа

3

Вхідні дані

двоосібна біматрична гра: матриці виграшів A та B

2

Візуальні результати

графові представлення, ландшафт ігор

4

5

Склад та технології програмного середовища



Технології:

Python, Streamlit, NumPy, Pandas, NetworkX, Matplotlib / Plotly, scikit-learn, UMAP.

6

Режими роботи з програмою

Аналіз гри



Введення власної гри



Ландшафт ігор



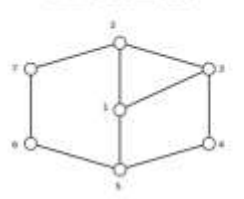
7

Порівняння ігор за структурними властивостями

Локаційна гра лікарів

На острові розташовані кілька містечок, у кожному з яких проживає однакова кількість мешканців. Містечка з'єднані між собою системою доріг. Два лікарі планують відкрити свої кабінети. Кожен лікар одночасно обирає одне містечко як місце розташування свого кабінету. У середньому кожен мешканець острова відвідує лікаря один раз на рік і завжди обирає найближчого лікаря. Відстань вимірюється кількістю доріг, які потрібно пройти від одного містечка до іншого. Виграш кожного лікаря дорівнює загальній кількості мешканців містечок, яких він обслуговує.

Network graph for location games



Локаційна гра ресторанів

На острові розташовані кілька містечок, у кожному з яких проживає однакова кількість мешканців. Містечка з'єднані між собою системою доріг. Два ресторани планують відкритися на острові. Вони мають однакову якість і рівень цін, тому клієнти обирають між ними лише за розташуванням. Кожен ресторан одночасно обирає одне містечко як місце свого розташування. Клієнт, у містечку якого є ресторан, відвідує його частіше. Клієнт, у містечку якого немає ресторану, але він є в сусідньому містечку, також може його відвідати. Ніхто не відвідує ресторан, який не розташований ані в містечку клієнта, ані в сусідньому з ним. Якщо обидва ресторани є однаково доступними для клієнта, клієнт розподіляється між ними порівну. Виграш кожного ресторану дорівнює очікуваній кількості клієнтів за рік.

8

Порівняння ігор за структурними властивостями

Локаційна гра лікарів. Матриця вигравів

Matrix representation

	Health Center 1	Health Center 2	Health Center 3	Health Center 4	Health Center 5	Health Center 6	Health Center 7
Health Center 1	(3.50, 3.50)	(3.50, 1.50)	(4.50, 3.00)	(4.50, 1.50)	(3.50, 3.50)	(4.00, 3.00)	(4.50, 2.50)
Health Center 2	(3.50, 3.50)	(3.50, 1.50)	(4.50, 3.00)	(4.50, 1.50)	(3.50, 3.50)	(4.00, 3.00)	(4.50, 2.50)
Health Center 3	(3.50, 4.00)	(3.50, 4.00)	(3.50, 3.50)	(4.50, 3.00)	(3.50, 3.50)	(4.00, 3.00)	(4.50, 3.00)
Health Center 4	(2.50, 4.50)	(3.00, 4.50)	(3.00, 4.50)	(3.00, 3.50)	(2.50, 4.50)	(3.00, 3.50)	(3.50, 3.50)
Health Center 5	(3.50, 3.50)	(3.50, 2.50)	(3.50, 2.50)	(4.50, 2.50)	(3.50, 3.50)	(4.00, 2.50)	(4.50, 2.00)
Health Center 6	(3.50, 4.00)	(3.50, 4.00)	(3.50, 4.50)	(3.50, 4.50)	(2.50, 4.50)	(3.50, 3.50)	(3.50, 3.50)
Health Center 7	(2.50, 4.50)	(2.50, 4.50)	(3.00, 4.50)	(3.00, 3.50)	(2.50, 4.50)	(3.00, 3.50)	(3.50, 3.50)

Локаційна гра ресторанів. Матриця вигравів

Matrix representation

	Restaurant 1	Restaurant 2	Restaurant 3	Restaurant 4	Restaurant 5	Restaurant 6	Restaurant 7
Restaurant 1	(2.00, 2.00)	(2.50, 2.50)	(2.50, 2.50)	(3.00, 2.00)	(3.00, 3.00)	(3.50, 2.00)	(3.50, 2.00)
Restaurant 2	(3.50, 3.50)	(3.00, 2.00)	(3.50, 2.50)	(3.50, 3.50)	(3.50, 3.50)	(3.50, 2.00)	(3.50, 2.00)
Restaurant 3	(2.50, 2.50)	(2.50, 2.50)	(2.00, 2.00)	(2.00, 2.00)	(3.00, 3.00)	(4.00, 3.00)	(3.50, 2.50)
Restaurant 4	(2.00, 3.00)	(3.50, 3.50)	(3.50, 3.50)	(3.50, 3.50)	(3.50, 3.50)	(3.50, 3.50)	(3.50, 3.50)
Restaurant 5	(3.00, 3.00)	(3.50, 3.50)	(3.50, 3.50)	(3.50, 3.50)	(3.50, 3.50)	(3.50, 2.00)	(3.50, 2.50)
Restaurant 6	(3.50, 3.50)	(3.50, 3.50)	(3.50, 3.50)	(3.50, 3.50)	(3.50, 3.50)	(3.50, 3.50)	(3.50, 2.00)
Restaurant 7	(3.50, 3.50)	(3.50, 3.50)	(3.50, 3.50)	(3.50, 3.50)	(3.50, 3.50)	(3.50, 2.00)	(3.50, 2.00)

9

Порівняння ігор за структурними властивостями

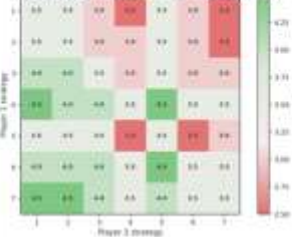
Локаційна гра лікарів. Теплові карти



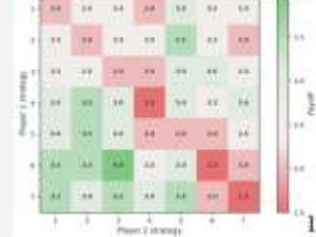
Локаційна гра ресторанів. Теплові карти



Player 2 Payoffs: Doctor Location Game



Player 2 Payoffs: Restaurant Location Game



10

Порівняння ігор за структурними властивостями

Локаційна гра лікарів. Таблиця найкращих відповідей. Чисті рівноваги Неша

Best Responses

Player	Opponent Strategy	Best Response
1: doc	Both doctor 1	(1, 1, 0)
1: doc	Both doctor 2	(1, 1, 0)
1: doc	Both doctor 3	(1, 1, 0)
1: doc	Both doctor 4	(1, 1, 0)
1: doc	Both doctor 5	(1, 1, 0, 0)
1: doc	Both doctor 6	(1, 1, 0)
1: doc	Both doctor 7	(1, 1, 0)
1: doc	Both doctor 8	(1, 1, 0)
1: doc	Both doctor 9	(1, 1, 0)
1: doc	Both doctor 10	(1, 1, 0)
1: doc	Both doctor 11	(1, 1, 0)
1: doc	Both doctor 12	(1, 1, 0)
1: doc	Both doctor 13	(1, 1, 0)

Pure Nash Equilibria

Pure Nash Equilibria
1: (1, 1, 0)
1: (1, 1, 0)
1: (1, 1, 0)
1: (1, 1, 0)
1: (1, 1, 0)
1: (1, 1, 0)
1: (1, 1, 0)
1: (1, 1, 0)
1: (1, 1, 0)
1: (1, 1, 0)
1: (1, 1, 0)
1: (1, 1, 0)
1: (1, 1, 0)

Локаційна гра ресторанів. Таблиця найкращих відповідей. Чисті рівноваги Неша

Best Responses

Player	Opponent Strategy	Best Response
1: doc	Both doctor 1	(1, 1, 0)
1: doc	Both doctor 2	(1, 1, 0)
1: doc	Both doctor 3	(1, 1, 0)
1: doc	Both doctor 4	(1, 1, 0)
1: doc	Both doctor 5	(1, 1, 0)
1: doc	Both doctor 6	(1, 1, 0)
1: doc	Both doctor 7	(1, 1, 0)
1: doc	Both doctor 8	(1, 1, 0)
1: doc	Both doctor 9	(1, 1, 0)
1: doc	Both doctor 10	(1, 1, 0)
1: doc	Both doctor 11	(1, 1, 0)
1: doc	Both doctor 12	(1, 1, 0)
1: doc	Both doctor 13	(1, 1, 0)

Pure Nash Equilibria

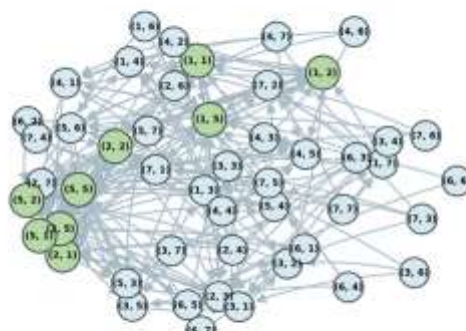
Pure Nash Equilibria
1: (1, 1, 0)
1: (1, 1, 0)
1: (1, 1, 0)
1: (1, 1, 0)
1: (1, 1, 0)
1: (1, 1, 0)
1: (1, 1, 0)
1: (1, 1, 0)
1: (1, 1, 0)
1: (1, 1, 0)
1: (1, 1, 0)
1: (1, 1, 0)
1: (1, 1, 0)

11

Порівняння ігор за структурними властивостями

Локаційна гра лікарів. Граф відповідей

Response Graph: Doctor Location Game

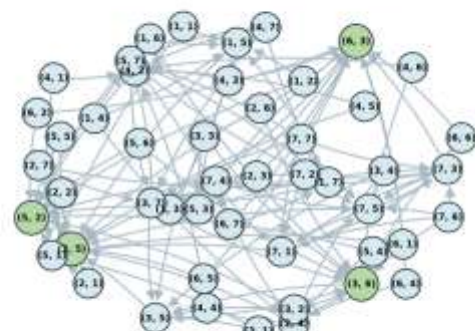


Response Graph Statistics

Metric	Value
Number of nodes	45
Number of edges	104
Strongly connected components	31
Cycles	10004

Локаційна гра ресторанів. Граф відповідей

Response Graph: Restaurant Location Game



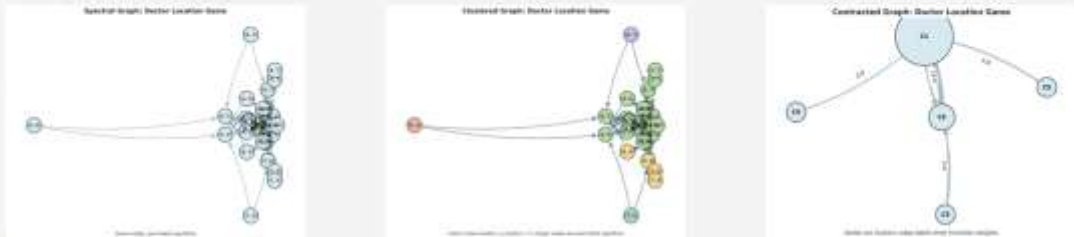
Response Graph Statistics

Metric	Value
Number of nodes	45
Number of edges	104
Strongly connected components	41
Cycles	11

12

Порівняння ігор за структурними властивостями

Локаційна гра лікарів. Спектральний, кластеризований граfi. Скорочений граф кластерів



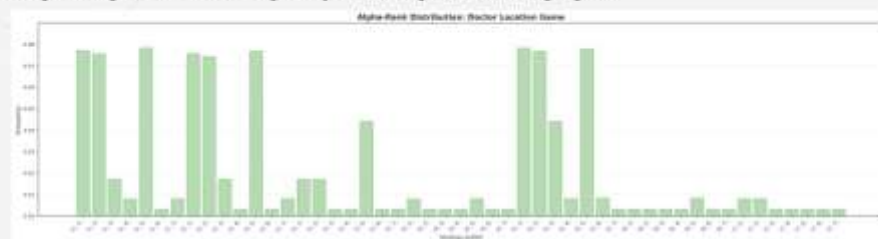
Локаційна гра ресторанів. Спектральний, кластеризований граfi. Скорочений граф кластерів



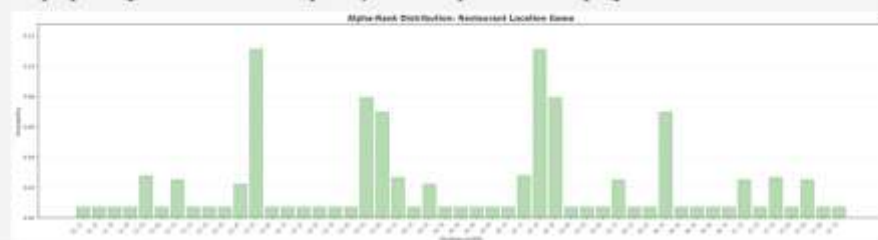
13

Порівняння ігор за структурними властивостями

Локаційна гра лікарів. Наближене ранжування стратегічних профілів



Локаційна гра ресторанів. Наближене ранжування стратегічних профілів



14

Структурні характеристики ігор

rows – кількість стратегій першого гравця

columns – кількість стратегій другого гравця

profiles – кількість стратегічних профілів гри

response_edges – кількість ребер у графі відповідей

density – щільність графа відповідей

mean_in_degree – середній вхідний ступінь вершин графа відповідей

mean_out_degree – середній вихідний ступінь вершин графа відповідей

max_in_degree – максимальний вхідний ступінь вершини

max_out_degree – максимальний вихідний ступінь вершини

pure_nash_equilibria – кількість чистих рівноваг Неша

strongly_connected_components – кількість сильно зв'язаних компонент графа відповідей

cycles – кількість циклів у графі відповідей

alpha_rank_entropy – ентропія наближеного ранжування стратегічних профілів

normalized_edges – нормалізована кількість ребер графа відповідей

normalized_cycles – нормалізована кількість циклів

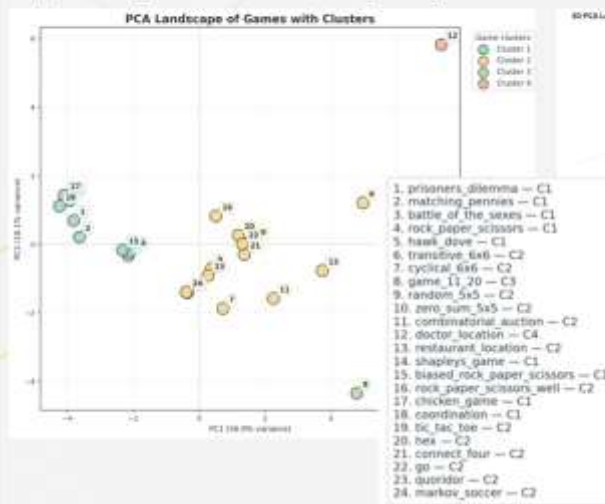
normalized_scc – нормалізована кількість сильно зв'язаних компонент

normalized_nash – нормалізована кількість чистих рівноваг Неша

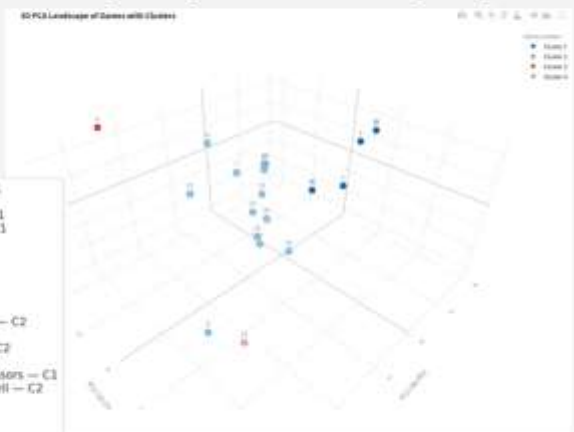
15

Ландшафт ігор

Двовимірний PCA-ландшафт ігор



Тривимірний PCA-ландшафт ігор

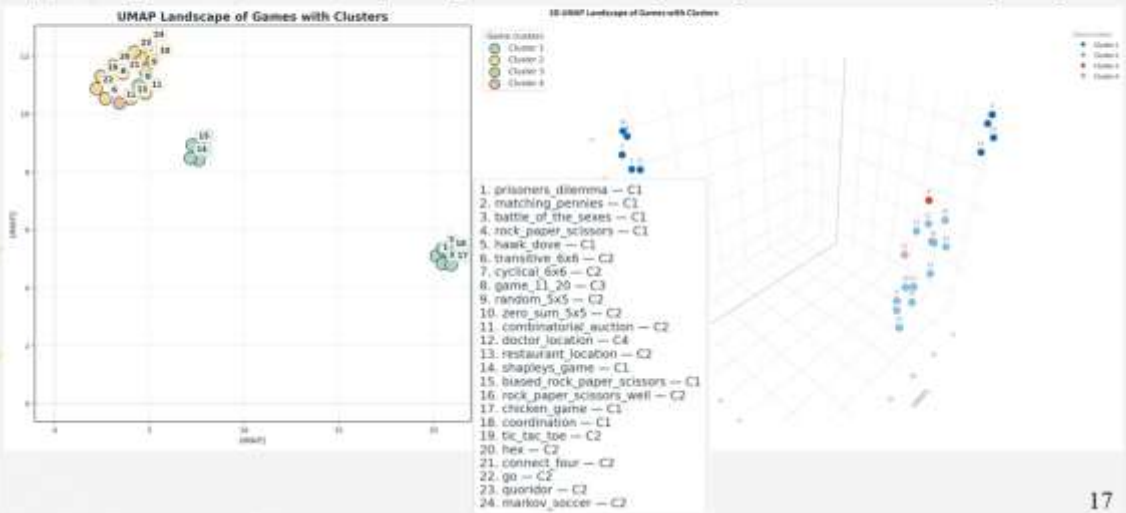


16

Ландшафт ігор

Двовимірний UMAP-ландшафт ігор

Тривимірний UMAP-ландшафт ігор



17

Висновки

- реалізовано програмне середовище для аналітико-візуалізаційного етапу емпіричного аналізу ігор у нормальній формі;
- реалізовано аналіз матриць виграшів та визначення рівноваг Неша;
- реалізовано побудову графа відповідей та обчислення структурних характеристик;
- реалізовано побудову ландшафту ігор для їх порівняння;
- проведено обчислювальні експерименти на класичних, синтетичних та прикладних іграх;
- мету дипломної роботи досягнуто.

18