МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

Кафедра інформаційних систем
(повна назва кафедри)

ДИПЛОМНА РОБОТА

РОЗРОБКА PYTHON-БІБЛІОТЕКИ ДЛЯ РОЗВ'ЯЗУВАННЯ ЗАДАЧ ТЕОРІЇ ІГОР

Виконав: студент групи ПМІ-42
спеціальності 122 – комп'ютерні науки
(шифр і назва спеціальності)

Патраманський М.І
(підпис) (прізвище та ініціали)

Керівник: Козій І. Я.
(підпис) (прізвище та ініціали)

Рецензент: Ярмола Г.П.
(підпис) (прізвище та ініціали)

2023

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет Прикладної математики та інформатики
Кафедра Інформаційних систем
Спеціальність 122 «Комп'ютерні науки»
(шифр і назва)

«ЗАТВЕРДЖУЮ»
Зав. кафедри проф. Шинкаренко Г.А.
« 7 » вересня 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Патраманському Максиму Івановичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка Python-бібліотеки для розв'язування задач теорії ігор

керівник роботи *доцент кафедри інформаційних систем, кандидат фізико-математичних наук, Козій Ірина Ярославівна*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені Вченою радою факультету від « 13 » вересня 2022 року № 15

2. Строк подання студентом роботи *12.06.2023р.*

3. Вихідні дані до роботи

Список літературних та інтернет джерел за тематикою роботи
Wikipedia: List of games in game theory
https://en.wikipedia.org/wiki/List_of_games_in_game_theory
Pandas: Documentation
https://pandas.pydata.org/docs/
Robert Martin. Clean Code: A Handbook of Agile Software Crafrmanship 1 st edition
C# documentation guide
https://learn.microsoft.com/en-us/dotnet/csharp/
Інші інтернет-ресурси

4. Зміст дипломної роботи (перелік питань, які потрібно розробити)

Ознайомитись з теоретичною базою та основними поняттями Теорії Ігор, спроектувати систему: визначити технології, визначити архітектуру та структуру, реалізувати систему на основі обраних технологій та архітектури, забезпечити для користувачів бібліотеки можливості, описані в функціональних вимогах, забезпечити докладну документацію як в коді, так і на ресурсі GitHub, провести масштабне тестування системи

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Таблиці, знімки екрану, демонстративні результати виконання програми, презентація

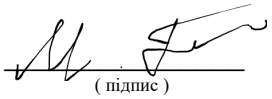
6. Консультанти розділів роботи

Розділ	**Прізвище, ініціали та посада консультанта**	**Підпис, дата**	
		завдання видав	**завдання прийняв**

7. Дата видачі завдання **14 вересня 2022 р.**

КАЛЕНДАРНИЙ ПЛАН

№ з/п	**Назва етапів дипломної роботи**	**Строк виконання етапів роботи**	**Примітка**
1.	Дослідження основ теорії ігор	вересень 2022	Виконано
2.	Постановка задач	вересень 2022	Виконано
3.	Аналіз існуючих аналогів та побудова плану роботи	жовтень 2022	Виконано
4.	Розробка архітектури системи	листопад 2022	Виконано
5.	Програмна реалізація	грудень-березень 2023	Виконано
6.	Масштабне тестування та пошук проблем	квітень 2023	Виконано
7.	Розробка документації	квітень 2023	Виконано
8.	Оформлення роботи	травень 2023	Виконано

Студент  Патраманський М. І.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Козій І. Я
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Дипломна робота: 43ст., 26 рис., 6 джерел.

Предмет дослідження: теорія ігор та задачі, побудовані в межах цього розділу математики.

Мета даної роботи полягає в розробці Python-бібліотеки, яка надає ефективні інструменти для розв'язування задач теорії ігор. Основним завданням роботи є створення програмного забезпечення, яке дозволить дослідникам, вченим та науковим керівникам використовувати теоретичні концепції теорії ігор у своїх проектах та аналізувати різноманітні гральні ситуації.

Результатом роботи є програма, яка будує моделі математичної теорії ігор та їх аналізує.

Ключові слова: теорія ігор, python, рівновага Неша, стратегія, домінування, pandas, numpy, pypi, tabulate, нормальна форма, розширена форма, платіжна матриця, use-case, гравець, github, SOLID, clean code.

ЗМІСТ

ВСТУП

У сучасному світі, де прийняття рішень є неодмінною складовою багатьох сфер життя, теорія ігор набуває все більшої актуальності. Ця наука вивчає принципи прийняття рішень в умовах конкуренції та співпраці, розглядаючи взаємодію різних сторін у гральних ситуаціях. Розвиток інформаційних технологій, зокрема мови програмування Python, відкриває нові можливості для застосування теорії ігор у різних областях.

Мета даної дипломної роботи полягає в розробці Python-бібліотеки, яка надає ефективні інструменти для розв'язування задач теорії ігор. Основним завданням роботи є створення програмного забезпечення, яке дозволить дослідникам, вченим та науковим керівникам використовувати теоретичні концепції теорії ігор у своїх проектах та аналізувати різноманітні гральні ситуації.

Для досягнення поставленої мети, дипломну роботу було розбито на кілька основних етапів. Початковим етапом є огляд наукових джерел, пов'язаних з теорією ігор та програмуванням на мові Python. Цей етап дозволив отримати загальне уявлення про існуючі методи та підходи до розв'язання задач теорії ігор за допомогою програмного забезпечення.

Наступним етапом було проектування та розробка Python-бібліотеки. В ході цього етапу були використані сучасні методи та практики програмування для створення ефективної та зручної у використанні бібліотеки. Особлива увага приділялась розробці функціоналу, що дозволяє моделювати гральні ситуації, виконувати аналіз стратегій гравців та розв'язувати різноманітні задачі, пов'язані з теорією ігор.

На останньому етапі дипломної роботи був проведений експериментальний аналіз розробленої бібліотеки. Це дозволило оцінити її ефективність, точність та можливості застосування у практичних завданнях. Було проведено порівняння з існуючими рішеннями з метою демонстрації переваг та недоліків розробленої бібліотеки.

Очікується, що результати даної дипломної роботи стануть внеском у

розвиток теорії ігор та допоможуть практикам у застосуванні цих концепцій у різних галузях, таких як економіка, політика, біологія та інші. Розроблена бібліотека може бути використана як основний інструмент для аналізу та розв'язання ігральних задач у відповідних дослідницьких та практичних проектах.

Варто зауважити, що даний дипломний проект зосереджений на поєднанні теоретичних аспектів теорії ігор з практичним програмуванням на мові Python. Розроблена бібліотека має потенціал стати корисним інструментом для вирішення складних ігральних задач та сприяти подальшому розвитку теорії ігор у сучасному інформаційному середовищі.

1 ПОСТАНОВКА ЗАДАЧІ ТА ВИМОГИ

1.1 Формулювання основних задач

Розробити python-бібліотеку “GameTheoryPy” для опрацювання задач теорії ігор:

а) Ознайомитись з теоретичною базою та основними поняттями Теорії Ігор, такими як:
   1) Поняття теорії ігор та їх типи
      а) Кооперативні та некооперативні ігри
      б) Симетричні та антисиметричні ігри
      в) Ігри з нульовою та ненульовою сумою
      г) Паралельні та послідовні ігри
      д) Ігри з повною чи неповною інформацією
   2) Представлення ігор
      а) Нормальна форма
      б) Розгорнута форма
   3) Домінуючі та доміновані стратегії
   4) Раціоналізованість стратегій
   5) Рівновага Неша
   6) Основні приклади задач теорії ігор

б) Спроектувати систему: визначити технології, визначити архітектуру та структуру

в) Реалізувати систему на основі обраних технологій та архітектури

г) Забезпечити для користувачів бібліотеки можливості, описані в функціональних вимогах

д) Використати GitHub та PyPI для публікації бібліотеки та можливості її використання іншими розробниками

е) Забезпечити докладну документацію як в коді, так і на ресурсі GitHub

ж) Провести масштабне тестування системи

з) Провести порівняння з іншими основними python-модулями, що

підтримують опрацювання різних елементів теорії ігор

1.2 Функціональні вимоги

Бібліотека має забезпечувати:

а) Можливість створення та опрацювання гри:
   1) В нормальній формі з підтримкою платіжних матриць

б) Вивід гри в кількох можливих форматах:
   1) Нормальна форма
   2) Окремі платіжні матриці гравців (__str__)
   3) Кодова репрезентація (__repr__)

в) Обрахунок очікуваного результату гри
   1) З обраними конкретними стратегіями гравців
   2) З випадковими стратегіями гравців

г) Пошук найвигіднішої стратегії для кожного з гравців

д) Визначення чи гра є антагоністичною чи ні

е) Пошук рівноваги Неша за допомогої алгоритму Support enumeration

ж) Повний деталізований вивід гри та його рішень в різних форматах, з підтримкою налаштувань назв та імен об’єктів та суб’єктів

1.3 Нефункціональні вимоги

а) Українська мова інтерфейсу

б) Зрозумілий опис кожного методу та модулю як в документації, так і в коді

в) Підтримка Python 3+

г) Надійність та стійкість до неочікуваної поведінки

д) Оптимізованість та швидкодія

2 ТЕОРЕТИЧНІ ВІДОМОСТІ

Предмет дослідження

Теорія ігор – галузь економічної теорії, що вивчає стратегічні взаємодії різних учасників (гравців) у ситуаціях конфлікту чи співпраці. Вона розглядає прийняття рішень гравцями з урахуванням можливих реакцій інших гравців і його власних очікувань.

Теорія ігор використовує математичні моделі та формалізм для аналізу стратегічних ситуацій, де взаємодіють два або більше раціональних гравців. Головною метою теорії ігор є вивчення оптимальних стратегій, які дозволяють гравцям досягти найкращих можливих результатів у даній ситуації.

Теорія ігор має широкий спектр застосувань, включаючи економіку, політичну науку, біологію, соціологію та інші галузі. Вона дозволяє аналізувати складні ситуації, де взаємодіють різні сторони зі суперечливими інтересами, і допомагає прогнозувати можливі результати та визначати оптимальні рішення.

2.1.1 Класифікація ігор

Ігри, що розглядаються в межах «Теорії Ігор», можна класифікувати за багатьма різними параметрами. Зрозуміло, що кожна гра може характеризуватись якимось набором параметрів та якостей, які можна згрупувати і сформувати класифікацію та загальні типажі ігор. Ось деякі з цих параметрів:

а) Кількість гравців
б) Кількість стратегій для кожного з гравців
в) Кількість рівноваг Неша (цю характеристику розглянемо в подальшому)
г) Послідовність чи паралельність. Цей параметр характеризує, чи ходи гравців є послідовні один за одним (наприклад, шахи), чи вони є паралельними і гравці не можуть орієнтуватись на хід суперника (наприклад, гра «Камінь-ножиці-папір»
д) Повнота інформації
е) Сума результатів

В цілому, ігри можна класифікувати та поділяти на подібні категорії:

а) Кооперативні та некооперативні

б) Симетричні та антисиметричні

в) З нульовою та ненульовою сумою

г) Паралельні та послідовні

д) З повною або неповною інформацією [1]

Кооперативні ігри: У кооперативних іграх гравці можуть укладати угоди, співпрацювати і координувати свої дії для досягнення спільного результату. В таких іграх акцент здебільшого робиться на аналізі коаліцій - груп гравців, які об'єднуються, щоб отримати певний розділений виграш. У цих іграх можуть бути встановлені правила розподілу виграшу між гравцями в разі досягнення спільної мети.

Некооперативні ігри: У некооперативних іграх гравці діють незалежно один від одного і не можуть укладати офіційні угоди або співпрацювати. Кожен гравець діє з метою максимізувати свій власний виграш, припускаючи, що інші гравці так само раціональні і спрямовані на досягнення своїх інтересів. В таких іграх аналізується поняття рівноваги, де кожен гравець обирає оптимальну стратегію, враховуючи дії інших гравців.

Симетричні ігри: В симетричних іграх гравці мають однаковий набір стратегій і однакові виграші для кожної комбінації стратегій. Це означає, що правила гри і виграші не залежать від конкретного гравця, і кожен гравець має однакові можливості та виграші. Такі ігри вважаються симетричними у відношенні до гравців.

Антисиметричні ігри: В антисиметричних іграх стратегії та виграші можуть бути різними для різних гравців, існує розрізнення між ролями гравців. В таких іграх виникає ситуація, коли один гравець має протилежні інтереси або можливості в порівнянні з іншим гравцем. Наприклад, один гравець може мати роль "атакуючого", а інший - "захисника". Такі ігри вважаються антисиметричними через наявність розрізнення між гравцями.

Ігри з нульовою сумою: В таких іграх виграш одного гравця є прямо пропорційним до втрати іншого гравця. Це означає, що сума виграшів індивідуальних гравців завжди дорівнює нулю (або постійній константі). Гра розглядається як конкуренція, де перемога одного гравця призводить до програшу іншого гравця, і немає можливості спільної вигоди.

Ігри з ненульовою сумою: У таких іграх виграші гравців не обов'язково збігаються або пропорційні. Сума виграшів гравців може бути відмінною від нуля, що означає, що вони можуть досягти спільної вигоди або підійти до спільно прийнятого рішення

Паралельні ігри: У паралельних іграх гравці приймають свої рішення одночасно, незалежно один від одного. Вони не бачать рішень інших гравців до того, як ухвалити своє власне. Це дозволяє гравцям одночасно вибирати свої стратегії, і результат визначається комбінацією їхніх рішень. Паралельні ігри часто моделюють конкурентну ситуацію, де гравці змагаються одночасно.

Послідовні ігри: У послідовних іграх гравці приймають свої рішення послідовно, один за одним, з урахуванням дій інших гравців, які вже зробили свій хід. Кожен гравець приймає рішення, враховуючи можливі наслідки попередніх ходів і очікує відповіді від наступних гравців.

Ігри з повною інформацією: В таких іграх кожен гравець знає повну інформацію про вибір стратегій і виграші всіх гравців. Гравці знають всі можливі ходи, правила гри та виграшні функції. Ігри з повною інформацією дозволяють гравцям приймати раціональні рішення, враховуючи всю доступну інформацію.

Ігри з неповною інформацією: У таких іграх гравці мають обмежену або неповну інформацію про вибір стратегій і/або виграші інших гравців. Ігри з неповною інформацією можуть включати елементи невизначеності або ризику, де гравці не мають повної впевненості щодо дій інших гравців.

2.1.2 Представлення ігор

Існує кілька різноманітних варіантів того, як гра представляється на письмі та візуально. Найпопулярнішими є два способи: нормальна форма та нозгорнута

форма.

У теорії ігор **нормальна форма** — це один з видів опису гри. Основна ідея цього формату – це представлення гри за допомогою матриці. Якщо допускати нормальну форму, то гра в ній складається з трьох компонент: множина гравців, множина стратегій кожного гравця, множина платіжних функцій кожного гравця. Таким чином, гру в нормальній формі можна подати у вигляді n-розмірної матриці (таблиці), елементи якої це n-розмірні платіжні вектори. Ця таблиця називається **платіжною матрицею** (англ. payoff matrix).

Як приклад можна навести наступну таблицю, що демонструє можливі стратегії двох гравців в грі:

Таблиця 1 – Нормальна форма гри

		Петро		
		Камінь	Ножиці	Папір
Микола	Камінь	0, 0	1, -1	-1, 1
	Ножиці	-1, 1	0, 0	1, -1
	Папір	1, -1	-1, 1	0, 0

Як можна спостерігати на таблиці вище, у нас є два гравці («Микола» та «Петро»), що вирішили зіграти між собою в гру «Камінь-ножиці-папір». Для того, щоб змоделювати цю гру та перенести в нормальну форму, ми будуємо таблицю. Рядки – це стратегії Миколи, а стовпчики – стратегії Петра. Відповідно, комірки, що знаходяться на пересіченні різних стратегій гравців, символізують ймовірні результати гри за цих же стратегій. Для зручності «ваги» (або, простішою мовою, результати) виділені відповідним кольором гравця.

До прикладу, візьмемо, що Микола вирішив зіграти стратегію «Ножиці», а Петро – «Папір». Тоді бачимо, що Микола отримує 1 очко (в нашому випадку це дорівнює виграшу), а Петро отримує -1 очко, тобто програє.

Розширена форма — це специфікація гри, яка дозволяє (як випливає з назви) чітке представлення ряду ключових аспектів гри, таких як послідовність можливих ходів гравців, їхній вибір у кожній точці прийняття рішення, (можливо

недосконала) інформація, яку має кожен гравець про ходи іншого гравця, коли він приймає рішення, і їхні виграші для всіх можливих результатів гри. Основна відмінність від нормальної форми в тому, що гра представляється не матрично, а за допомогою графів-дерев.

Вершини цього дерева поділяють на термінальні (кінцева) і нетермінальні. Кожна нетермінальна вершина характеризується безліччю допустимих ходів і доступною для гравця інформацією. Термінальні вершини повідомляють про розміри виграшу, отриманого по їх досягненню.

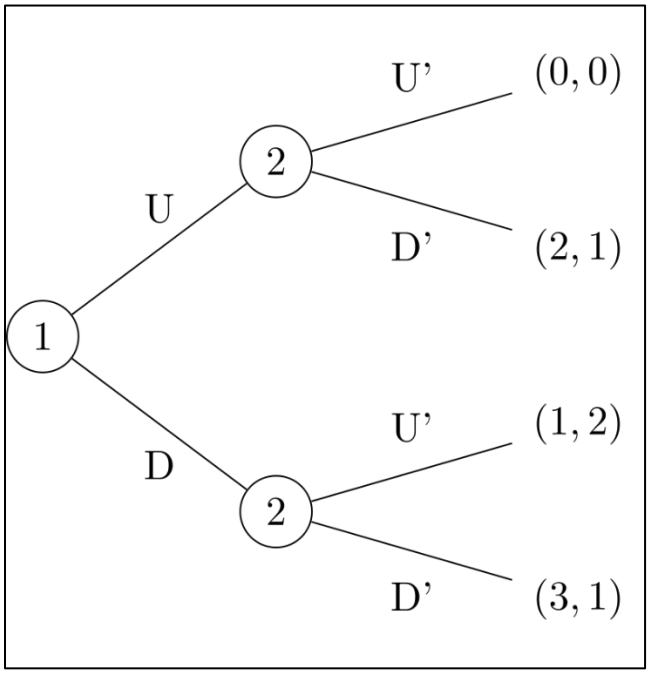


Рисунок 2.1 – Розширена форма гри

На рисунку вище можна бачити, що коренева вершина характеризує хід першого гравця (1). Далі, в залежності від ходу першого гравця, є вибір другого гравця (2). В результаті, маючи огляд їх послідовних виборів, приходимо до термінальних вершин, які характеризують результати гри подібно до комірок матриці в нормальній формі.

2.1.3 Домінуючі та доміновані стратегії

Домінуючі стратегії в теорії ігор - це стратегії, які завжди приносять гравцеві кращий результат, незалежно від вибору інших гравців. Гравець вважається

раціональним, якщо він вибирає домінуючу стратегію, оскільки це гарантує йому максимальний виграш у будь-якій ситуації.

Домінованою стратегією називається та стратегія, яка домінується іншими, домінуючими.

Також важливо розділяти **строгу** домінацію від **слабої** домінації. **Строго** домінуючою стратегією називається та, яка строго приносить більший результат за іншу. Натомість, **слабо** домінуюча підтримує нестрогу нерівність.

Якщо одна стратегія завжди приносить гравцеві більший виграш, ніж інша, то ми говоримо про сильне домінування. Якщо у гравця є одна стратегія, яка сильно або слабо домінує всі інші, то ми можемо очікувати, з великою впевненістю, що він зіграє саме її. Якщо така стратегія є у кожного гравця, то ми отримали рішення гри - прогноз щодо того, що зробить кожен гравець [2].

Також, можна зауважити, що будь яка сильно домінуюча стратегія є в той же час слабо домінуючою, але не навпаки [2].

2.1.4 Рівновага Неша

Однією із найчастіше використовуваних концепцій в межах теорії ігор є рівновага Неша. Грубо кажучи, **рівновага Неша** – це випадок, коли, нікому із **активних елементів (гравців)** не вигідно змінювати свою стратегію, за умови, що інші **активні елементи** не міняють своїх стратегій.

Слід зазначити, що використання концепції рівноваги Неша вимагає введення наступної гіпотези: гравці не можуть домовитися і піти із цієї точки спільно. Тобто рівновага Неша припускає відсутність коаліцій гравців, що передбачається для некооперативних ігор [3].

У рівновазі Неша особисте рішення гравця повертається на нього самого. Інакше кажучи, якщо він прийняв «не те» рішення, що обумовлено вимогою рівноваги Неша, то він «одержує менше», тобто програє.

Таким чином, рівновага Неша «повертає» на певного гравця всі його «невдалі» рішення. Ця рівновага сформульована в термінах діяльності того ж самого гравця, і, у випадку програшу, він може «повернути свій гнів» винятково на самого себе!

Більш того: рівновага Неша вимагає довіри до того, що всі інші гравці – також «розумні», і добре знають та «можуть обчислити» свою власну вигоду. Понад те: рівновага Неша вимагає - якщо якийсь один гравець «зрозумів», яким чином можна досягти такої рівноваги, то найкраща його стратегія полягає в тому, щоб негайно інформувати інших гравців про всі ті стратегії, яких вони повинні дотримуватися, щоб збільшити їхній виграш (тобто перейти до рівноваги Неша) [3].

3 ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

3.1 Структура

3.1.1 Use Case діаграма

Рисунок 3.1 – Use Case діаграма

3.1.2 Опис основних функцій бібліотеки

Архітектурно, бібліотека GameTheoryPy міститиме два основних класи для роботи з функціоналом теорії ігор – це класи Player та Game.

3.1.2.1 Клас Player (гравець)

а) **Ініціалізація (__init__())**: конструювання абстрактної моделі гравця, підтримує параметри:
   1) Ім’я (або назва) гравця
   2) Платіжна матриця гравця – його результати за різних результатів гри
   3) Назви стратегій (якщо відсутні, записуються назви Strategy 1, 2, 3…)
   4) Конкретна стратегія, яку планує грати гравець (частіше всього відсутня на етапі створення гри)

б) Поле **name**: Назва гравця (або його ім’я)

в) Поле **payoff_matrix**: платіжна матриця гравця, яку можна редагувати

г) Поле **strategy_names**: Список стратегій, які може зіграти гравець. Розмір списку має сходитись з розмірністю платіжної матриці

д) Поле **current_strategy(strategy: str)** : Задання стратегії гравцеві. Цієї стратегії він буде дотримуватись під час процесу самої «Гри» та обраховування її результату бібліотекою

е) Функція **__str__()**: Використовується для виводу моделі гравця. Виводиться ім’я гравця та його платіжна матриця, включно з назвами стратегій

ж) Функція **__repr__():** Використовується для виводу гравця в вигляді простої форми позначення класу та імені

з) Функція **has_random_strategy():** визначає, чи обрана стратегія у гравця випадкова, чи задана якоюсь специфічною зі списку існуючих стратегій

3.1.2.2 Клас Game (гра)

а) **Ініціалізація (__init__())**: конструювання абстрактної моделі гри,

підтримує параметри:

1) Гравець 1 – модель першого гравця
2) Гравець 2 – модель другого гравця
3) Назва гри (відображається при виводі гри на екран)

б) Поле **player1**: Модель гравця 1

в) Поле **player2**: Модель гравця 2

г) Поле **title**: Назва гри

д) Функція **__str__()**: Використовується для виводу моделі гри в вигляді двох матриць гравців.

е) Функція **__repr__():** Використовується для виводу гри в вигляді простої форми позначення класу та назви гри

ж) Функція **print_normal():** Використовується для виводу гри в нормальній формі. Будує відформатований вивід, з об'єднаною платіжною матрицею, назвою гри та відповідними гравцями

з) Функція **is_zero_sum():** Визначає, чи гра є грою з нульовою сумою (антагоністичною), базуючись на платіжних матрицях гравців

и) Функція **play():** Симулює гру та визначає її результат, базуючись на обраних стратегіях гравців. Результат повертається у вигляді рахунку (результату) кожного з гравців в числовому значенні. Підтримує кілька варіантів гри:

1) З конкретно заданими стратегіями гравців. Найпростіший випадок, визначає результат, базуючись на заданих стратегіях
2) З можливими випадковими стратегіями гравців. В такому випадку результатом буде математичне очікування виграшу обох гравців. Можливий варіант, коли лише один гравець грає випадкову стратегію, тоді ймовірність вираховується з урахуванням цих особливостей

к) Функція **find_dominant_strategies():** Знаходить можливі домінуючі стратегії для обох гравців базуючись на платіжних матрицях цих гравців. Метод працює наступним чином: серед усіх стратегій гравця обирає ту,

середній результат якої найвищий, не беручи уваги стратегію суперника. Якщо подібних стратегій кілька (коли дві стратегії в середньому приносять однаковий результат), то обраховується домінуюча стратегія суперника і з прийняттям цього до уваги серед цих рівнозначних стратегій обирається краща.

л) Функція **nash_equilibrium():** Шукає всі можливі рівноваги Неша за прямих стратегій гравців, не враховуючи змішані варіації та ймовірності. Може не містити ніякого результату, якщо не існує чистих рівноваг Неша, а можливі тільки змішані

3.2 Використані технології та ресурси

3.2.1 Мова програмування

Для написання бібліотеки подібного типу можна розглядати різні мови програмування з їх вбудованими технологіями: C++, C#, Java, TypeScript, R та ін., але мною було обрано Python з декількох, на мою думку, важливих причин:

а) Швидкість розробки та освоєння

б) Широкий набір інструментів та бібліотек для роботи з математичними даними (NumPy), табличними структурами (Pandas, Tabulate), деревоподібними структурами (BinaryTreePy) та ін.

в) Наявність власного досвіду роботи з мовою Python

г) Велика поширеність мови, що сприятиме більш розгорнутому використанню бібліотеки серед інших розробників

д) Наявність зручного менеджеру бібліотек PyPi, що дозволить зручно інтегровувати бібліотеку в інші проекти іншими розробниками

Також, для мене було важливим підтримувати принципи Clean Code [4] та SOLID. Застосовуючи SOLID та дизайн патерни, я зумів побудувати гнучку систему, яка відкрита до розширень і проста в модифікації, причому з мінімальною кількість багів та помилок.

Також, обравши Python, я одразу визначився з бібліотеками, які дозволяють комфортно опрацьовувати та вираховувати дані, а також в цілому пришвидшують

досягнення цілей, поставлених вище:

а) **NumPy** – для роботи з обрахунками та масивами

б) **Pandas** – це бібліотека з відкритим вихідним кодом, ліцензована BSD, надає високопродуктивні, прості у використанні структури даних та інструменти аналізу даних для мови програмування Python [5]. Конкретно мені вона потрібна для опрацювання таблиць (в першу чергу, платіжних матриць, які зберігатимуться як датафрейми). Важливою особливістю бібліотеки є дуже широкий опис її функціоналу в вигляді документації

в) **Tabulate** – спеціальна бібліотека для форматування виводу датафреймів, що, наприклад, дозволяє коректно виводити нормальну форму гри на екран

3.2.2 GitHub

Для зручного контролю коду, вочевидь, найкраще використовувати безкоштовний ресурс GitHub. Також, що зручно, це дозволить запровадити детальну документацію.

Загальна структура проекту виглядає наступним чином:

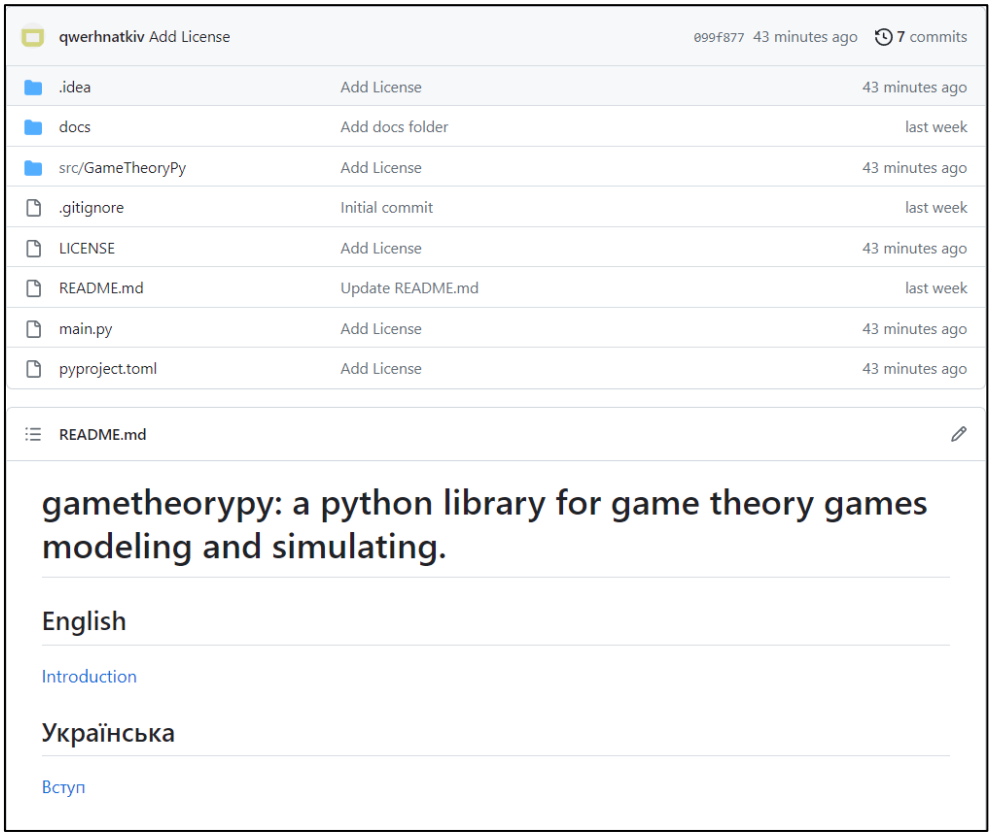


Рисунок 3.2 – Структура GitHub

Як бачимо, основними двома папками є src (кодова база) та docs (документація двома мовами).

На основній сторінці можна спостерігати посилання на документацію англійською та українською.

В майбутньому планується поширити документацію на ресурс ReadTheDocs.io.

3.2.3 Python Package Index

Для того, щоб ефективніше розповсюджувати свій бібліотечний пакет, було вирішено використовувати найпопулярніший та найочевидніший спосіб це виконати – через Python Package Index.

Python Package Index (скорочено **PyPI**) - це каталог програмного забезпечення, написаного на мові програмування Python, який включає в себе близько півмільйона пакетів.

Спочатку я перевірив, чи немає там пакету з такою ж назвою, якою я планую назвати власну бібліотеку (GameTheoryPy):

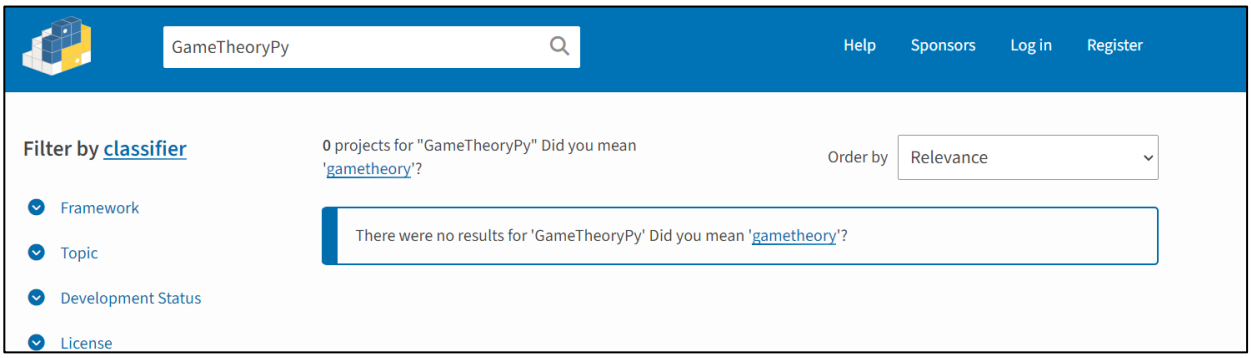


Рисунок 3.3 – Перевірка на зайнятість назви бібліотеки

Як виявилось, назва є унікальною, тому проблем з публікацією пакету на цьому ресурсі не має бути.

Потім, встановлення бібліотеки буде доступне за командою

pip install gametheorypy

Причому, встановлення менеджеру пакетів PyPI відбувається під час інсталювання Python на ПК, отже доступ до моєї бібліотеки буде максимально простим. Саме така доступність бібліотеки дозволить поширити її максимально успішно.

4 ДЕМОНСТРАЦІЯ РОБОТИ СИСТЕМИ

4.1 Створення гравця

Перед тим ,як створювати та запускати саму гру, потрібно проаналізувати гравців цієї гри, та створити їх в коді. Ц відбувається за допомогою класу Player, яким можна по-різному маніпулювати:

```
volodymyr = Player(name='Володимир', payoff_matrix=[[0, 1, -1], [-1, 0, 1], [1, -1, 0]],
                   strategy_names=['Камінь', 'Ножиці', 'Папір'])
petro = Player(name='Петро', payoff_matrix=[[0, -1, 1], [1, 0, -1], [-1, 1, 0]],
                   strategy_names=['Камінь', 'Ножиці', 'Папір'])
```

Рисунок 4.1 - Створення моделей гравців

В даному випадку я застосував відому гру “Камінь-ножиці-папір». Спочатку присвоїв кожному гравцю його ім’я, далі передав платіжну матрицю з результатами гри для кожної окремої стратегії (де 1 – це виграш, -1 – це програш, а 0 – це нічия). Фінальним параметром стали назви стратегій для можливості застосувати стратегію за назвою.

Для повної демонстрації можна вивести на екран гравців та показати їх властивості:

Платіжна матриця для гравця **'Володимир'**:

	Стратегія 1	Стратегія 2	Стратегія 3
Стратегія 1	0	1	-1
Стратегія 2	-1	0	1
Стратегія 3	1	-1	0

Рисунок 4.2 – Виведення першого гравця на екран

Платіжна матриця для гравця **'Петро'**:

	Стратегія 1	Стратегія 2	Стратегія 3
Стратегія 1	0	-1	1
Стратегія 2	1	0	-1
Стратегія 3	-1	1	0

Рисунок 4.3 – Виведення другого гравця на екран

Як можна спостерігати вище, в примітивній платіжній матриці можна спостерігати не реальні назви стратегій, а «стандартні» назви в стилі **Стратегія 1, Стратегія 2…** Це пов’язано з тим, що на момент, коли гра ще не створена, не є зрозумілим, чи гравець відповідає за рядок, чи за стовпчик (себто чи він гравець 1, чи гравець 2).

Також, можна зазначити, що окрім перевантаження оператора __str__ (що використовувався вище в методі print(), також було перевантажено оператор __repr__, вивід якого виглядає наступним чином:

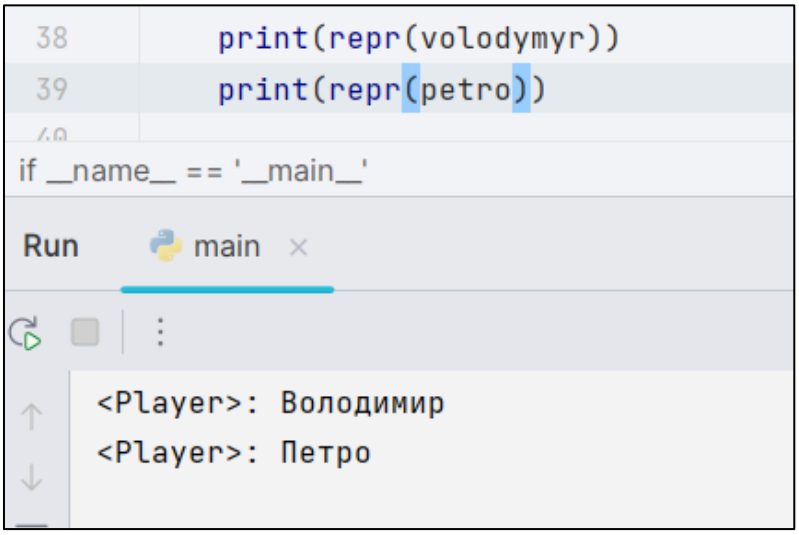


Рисунок 4.4 – Репрезентація __repr__ моделі гравця

4.2 Робота з моделлю гравця

Однією з інших функцій для гравця є, вочевидь, можливість його модифікації. Наприклад, зміна імені:

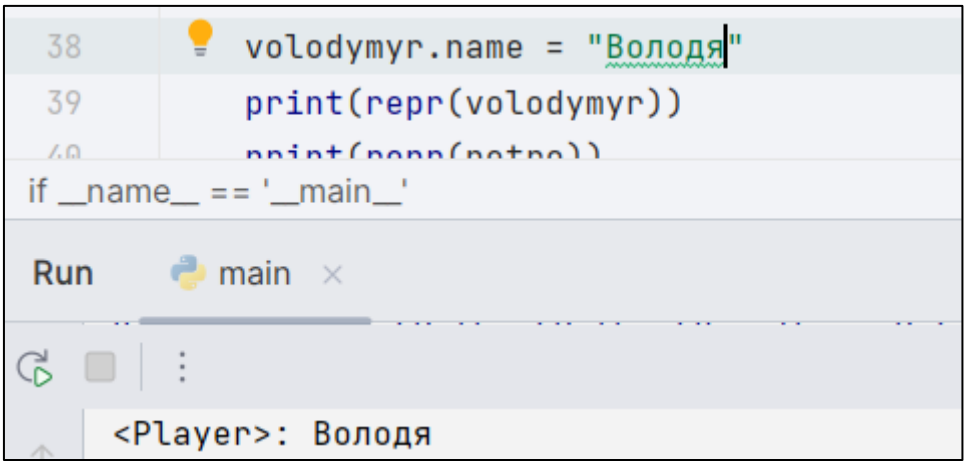


Рисунок 4.5 – Модифікація моделі гравця

Також, не менш важливою функцією є те, що користувач може визначити стратегію, якої буде дотримуватись гравець під час гри. Якщо ж стратегія не була задана, то за замовчуванням вона буде вважатись випадковою («**Випадкова**»):

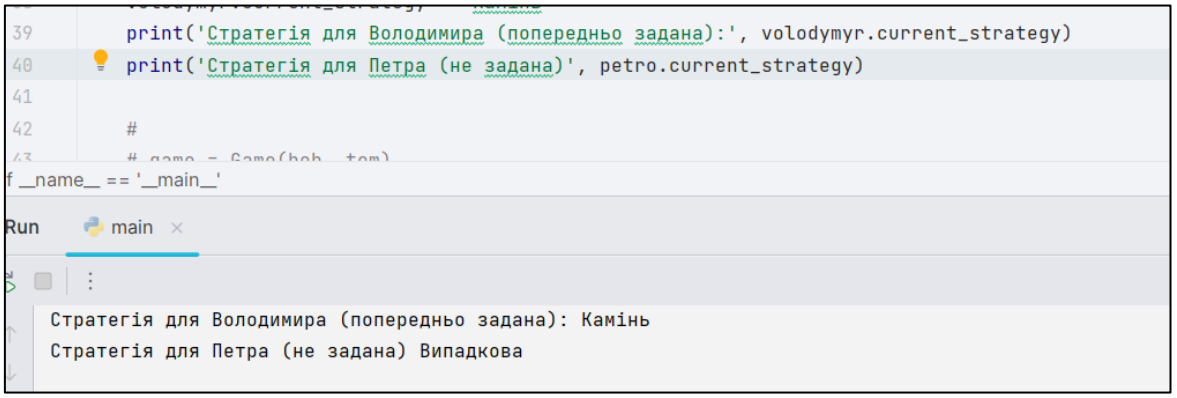


Рисунок 4.6 – Задавання стратегії гравцям

Причому, важливо розуміти, що якщо задати невалідну стратегію гравцю (тобто таку, як не входить в список його можливих стратегій – ці дані задаються при створенні гравця), то програма видасть помилку і запропонує правильні варіанти стратегії:

Рисунок 4.7 – Задавання невалідної стратегії гравцю

Останньою корисною функцією можна назвати метод, що визначає чи задана стратегія є випадковою. Насправді, цей метод більш потрібний для внутрішніх обрахунків та можливого подальшого розширення бібліотеки, але він все одно є відносно корисним:

Рисунок 4.8 – Визначення, чи стратегія гравця є випадковою

4.3 Створення гри

Наступним кроком буде створення гри, базуючись на її гравцях (та стратегіях, які мають ці гравці). Після цього є можливість задати назву гри (title). Взагалі, ця можливість передбачена і через конструктор класу Game, але шляхів є кілька:

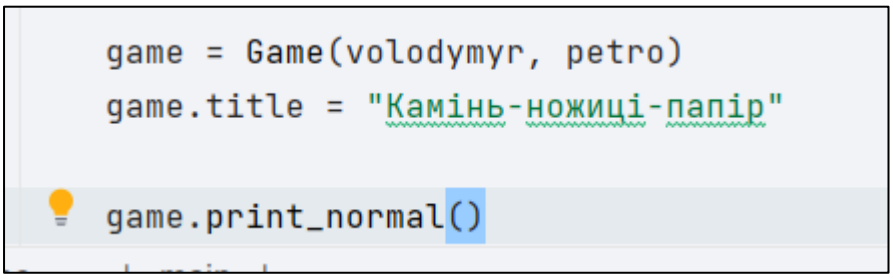


Рисунок 4.9 – Створення гри

Якщо не задати гри, то за замовчуванням буде відображатись назва “Game”.

4.4 Виведення гри на екран

Існує кілька методів, що по-різному відображають гру в консольному вікні виводу. Це методи:

__str__ - стандартний вивід, що застосується при виклику методу print()

__repr__ - класова репрезентація гри, скорочена форма для більш технічного виводу

print_normal - метод, що виводить гру в нормальній формі

4.4.1 __str__

Особливістю цього стилю виводу є те, що він виводить: назву гри, дві платіжні матриці для обох гравців (окремо для кожного), включно з описами назв цих стратегій та імен гравців.

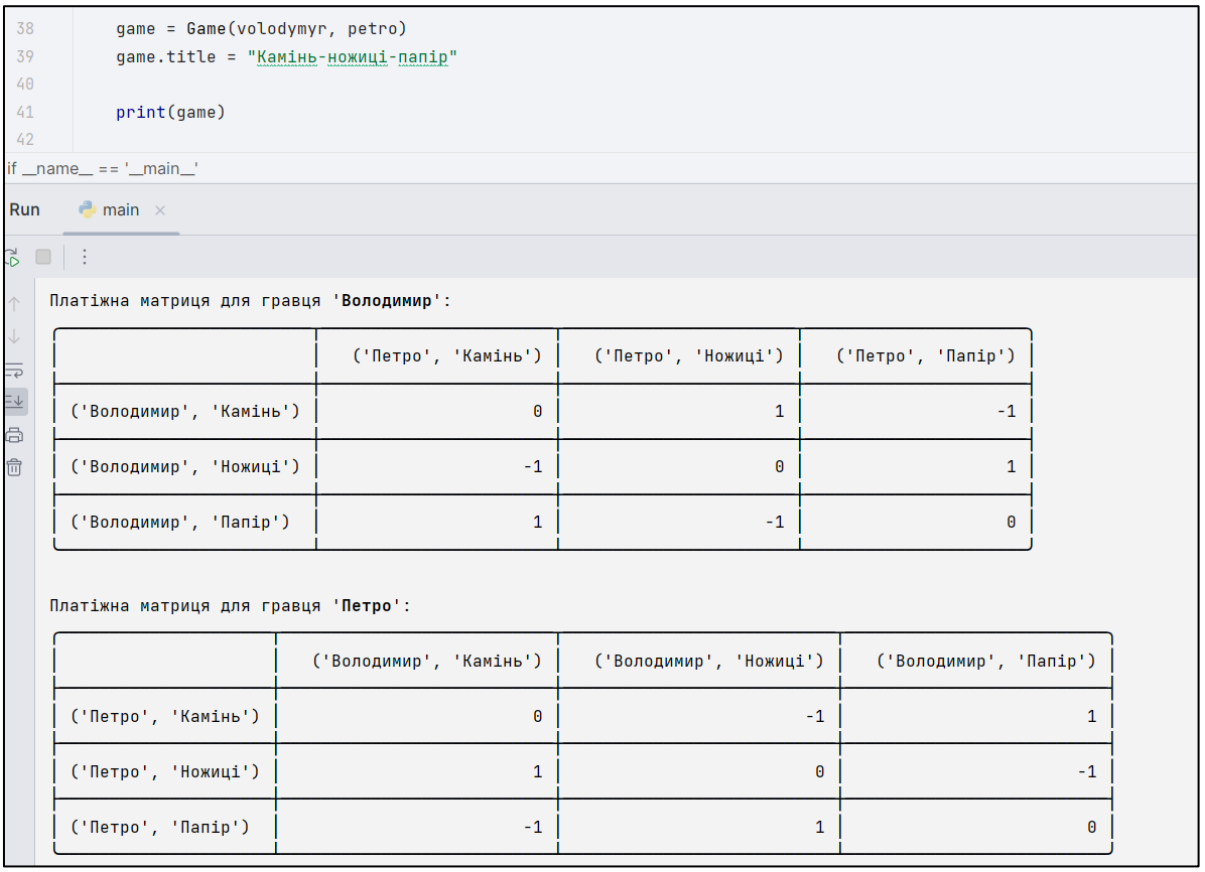


Рисунок 4.10 – Вивід гри в str форматі

4.4.2 __repr__

Перевантажений метод repr дозволяє виводити гру в форматі більш технічної, короткої форми (лише назва класу та самої гри):

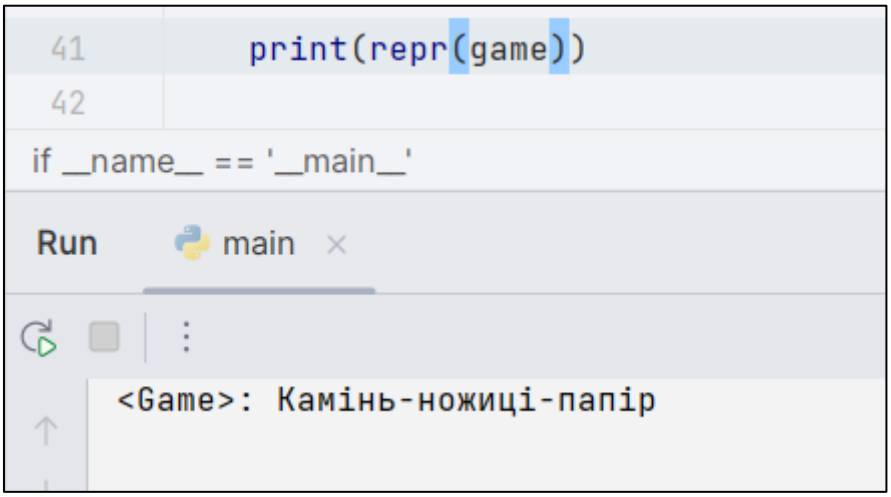


Рисунок 4.11 – Вивід гри в repr форматі

4.4.3 print_normal

Метод print_normal виводить гру в нормальній формі, включаючи імена гравців, їх стратегії та саму назву гри. Сам формат трохи схожий на str вивід, лише з однією відмінністю: тут виводиться спільна платіжна матриця, що дозволяє більш компактно вивчати гру:

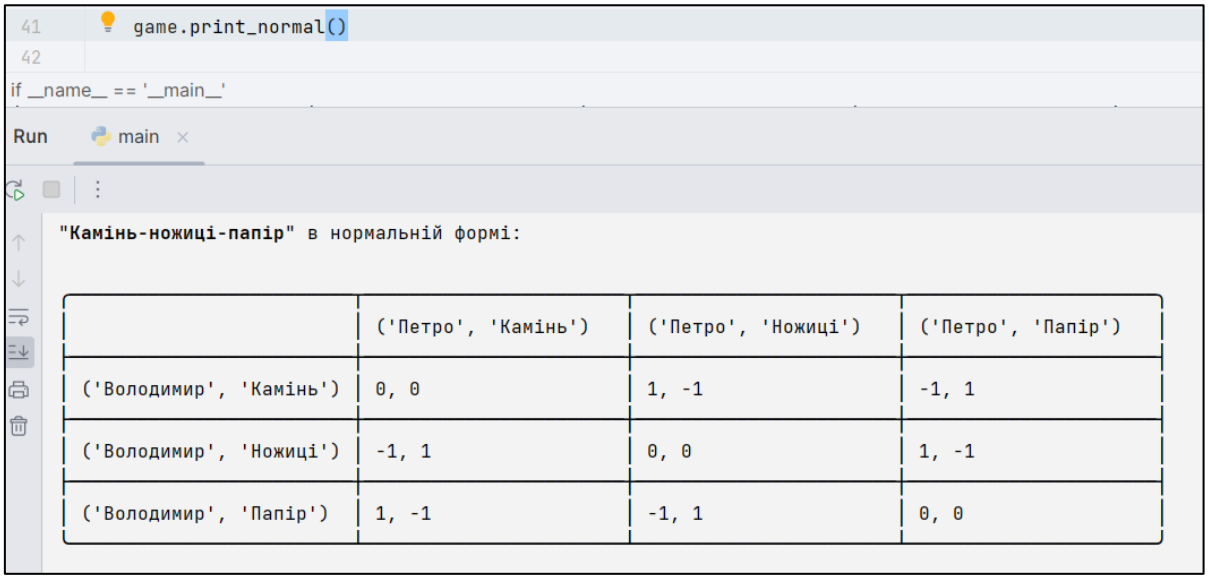


Рисунок 4.12 – Вивід гри в нормальній формі

4.5 Обчислення результатів та метрик гри

4.5.1 Метод is_zero_sum - Детермінування антагоністичних ігор

Одним з найпростіших функцій для роботи з класом Game є метод is_zero_sum, який перевіряє чи гра є антагоністичною (тобто грою з нульовою сумою):

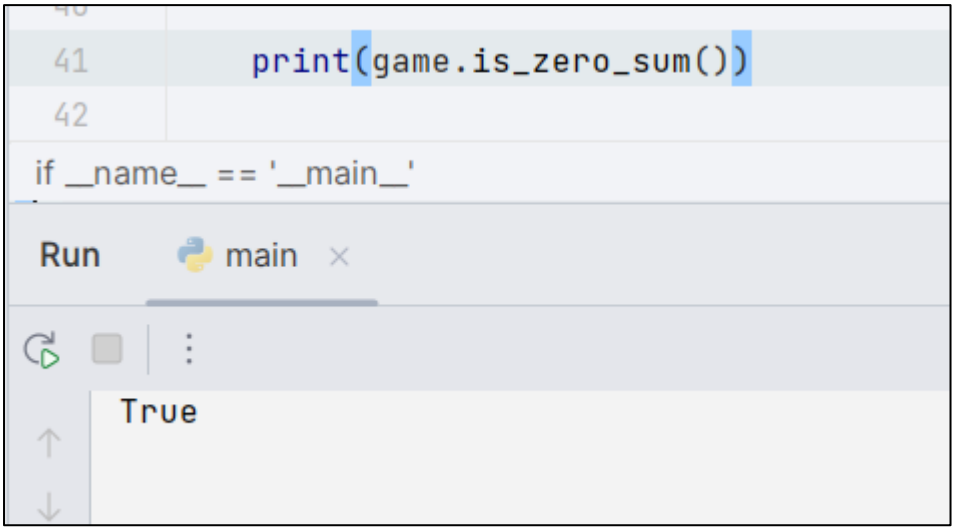


Рисунок 4.13 – Приклад антагоністичної гри

Взагалі, гра «Камінь-ножиці-папір» є антагоністичною, тому результат True є правильним і очікуваним. Якщо ж змінити якісь ваги при створенні гравців, то можна отримати зворотній результат:

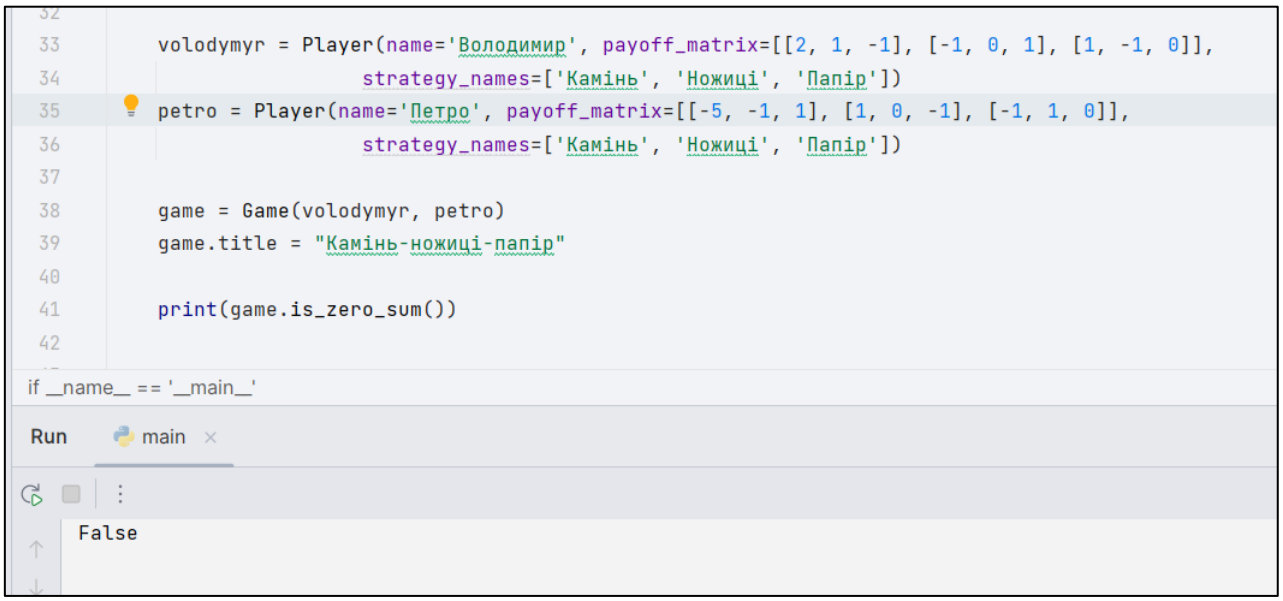


Рисунок 4.14 – Приклад не антагоністичної гри

4.5.2 Метод play - обрахунок результатів гри

Метод play дозволяє зіграти уявну гру, з заданими стратегіями гравців. Ці стратегії можуть бути як конкретні та визначати якусь явну стратегію гравця, так і випадкова, що буде підтримувати ймовірнісний результат гри.

Для початку, спробуємо задати гравцям якусь стратегію та провести гру:

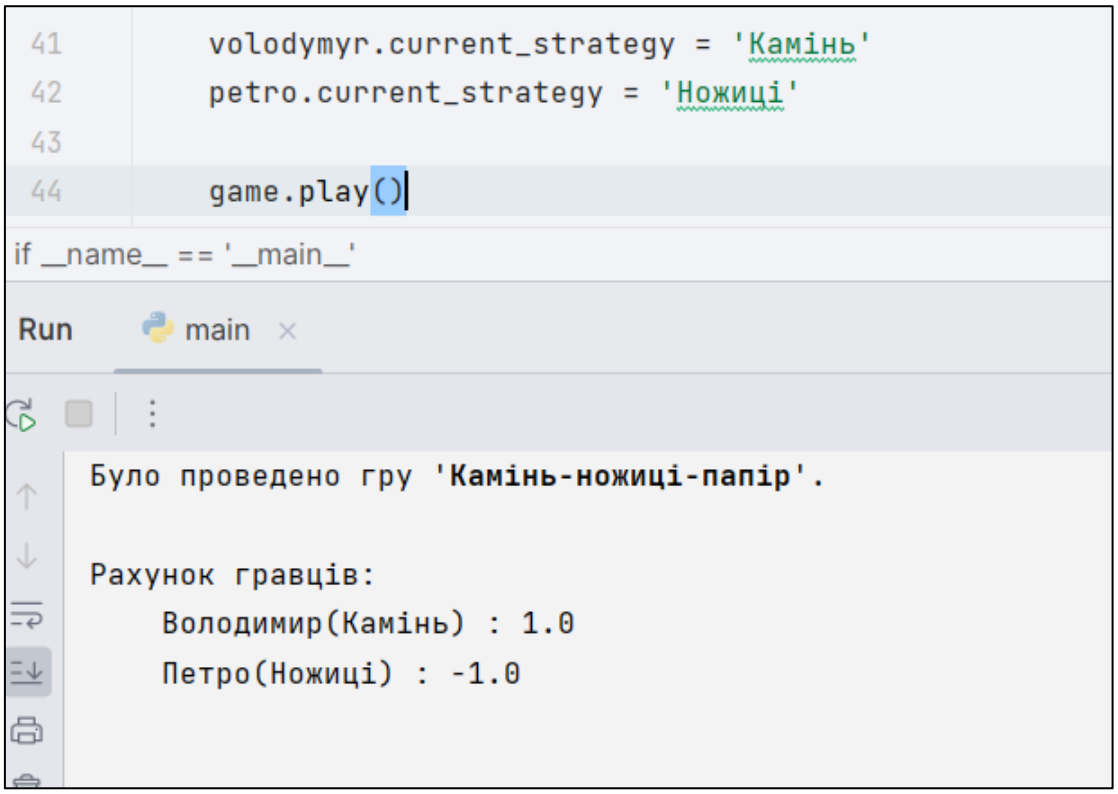


Рисунок 4.15 – Симуляція гри «Камінь-ножиці-папір»

Як можна спостерігати на скріншоті вище, кожен з гравців отримав якийсь результат гри. Оскільки Володимир зіграв камінь, а Петро – ножиці, то перший гравець переміг, отримавши 1 очко. В той же час, Петро отримав -1 очко, що демонструє його поразку. Якби обидва гравці зіграли одну і ту ж стратегію (наприклад, обидва гравці обрали «камінь»), то їх результатом послугувала б нічия – тобто обидва гравці мали б по 0 очок.

Для демонстрації роботи програми з випадковими стратегіями гравців, я використаю приклад іншої класичної гри, що називається “Battle of Sexes”. Коротка суть її така – хлопець та дівчина планують свої справи на вечір і хочуть обрати чим зайнятись – сходити в кіно чи на мюзикл. Кожен з них не знаючи вибору іншого обирає як провести час, і в залежності від їх виборів формується наступна платіжна матриця:

```
44      victor = Player(name='Віктор', payoff_matrix=[[2, 0], [0, 1]],
45                      strategy_names=['Кіно', 'Мюзикл'])
46      lyudmyla = Player(name='Людмила', payoff_matrix=[[1, 0], [0, 2]],
47                      strategy_names=['Кіно', 'Мюзикл'])
48
49      game = Game(victor, lyudmyla)
50      game.title = "Суперечка про проведення вечора"
51      game.print_normal()
52
if __name__ == '__main__'
```

main

"Суперечка про проведення вечора" в нормальній формі:

	('Людмила', 'Кіно')	('Людмила', 'Мюзикл')
('Віктор', 'Кіно')	2, 1	0, 0
('Віктор', 'Мюзикл')	0, 0	1, 2

Рисунок 4.16 – Модель суперечки про проведення вечора

Як можемо бачити, найгірший випадок для обох з пари – це коли обидва вони обирають різні варіанти, отже в їх інтересах обрати щось спільне.

Просимулювавши цю гру, можна спостерігати ймовірне математичне очікування кожного з гравців:

	('Людмила', 'Кіно')	('Людмила', 'Мюзикл')
('Віктор', 'Кіно')	2, 1	0, 0
('Віктор', 'Мюзикл')	0, 0	1, 2

Було проведено гру 'Суперечка про проведення вечора'.

Рахунок гравців:
    Віктор(Випадкова) : 0.5
    Людмила(Мюзикл) : 1.0

Рисунок 4.17 – Симулювання суперечки про проведення вечора

Як можна бачити, стратегіями для пари я задав наступні: Віктор обиратиме випадково, а Людмила буде дотримуватись свого наміру сходити на мюзикл. В

результаті, при таких вхідних даних, можна очікувати, що потенційне задоволення Людмили в середньому вдвічі вище, ніж у Віктора (1 проти 0.5).

4.5.3 Метод find_dominant_strategies – Пошук домінуючих стратегій для гравців

Наступний метод, find_dominant_strategies, виводить на екран потенційні домінуючі (тобто ті, що принесуть найбільший результат) стратегії для кожного гравця.

Важливо зазначити, що якщо існує кілька домінуючих комбінацій для гравця, то буде запропоновано обрати ту з них, яка принесе найбільший результат при домінуючій стратегії суперника (припускаємо, що суперник є теж раціональним і намагається грати домінуючу стратегію).

Для прикладу використаємо ту ж саму суперечку Віктора та Людмили.

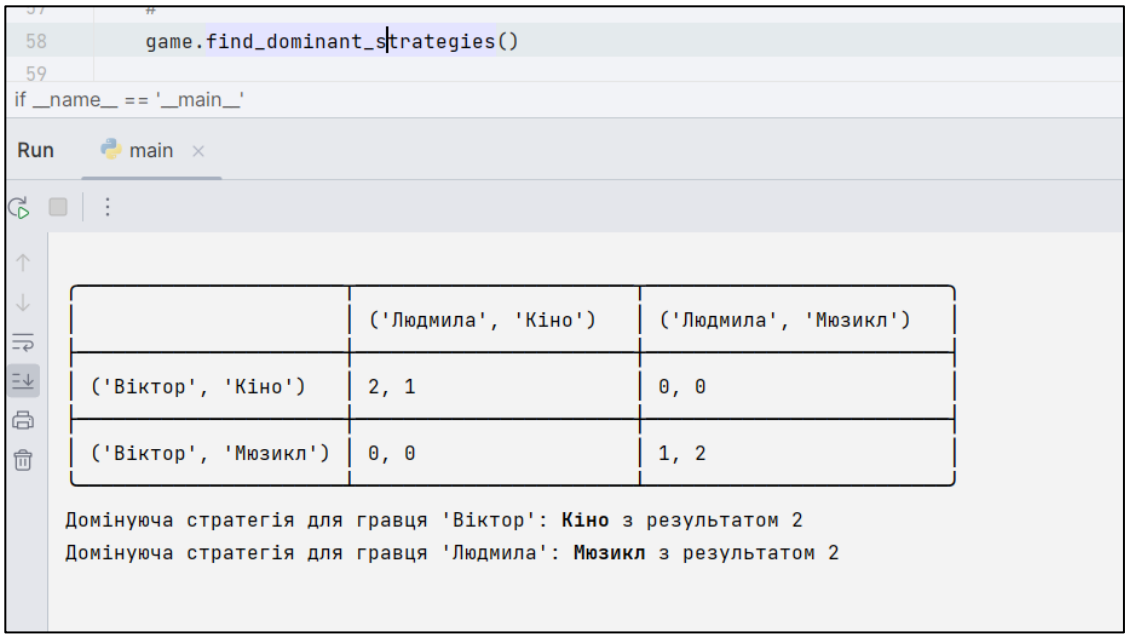


Рисунок 4.18 – Пошук домінуючої стратегії для гравців

Результатом гри є сума всіх можливих результатів, якщо гравець зіграє стратегію. Тобто, в нашому випаду, це число 2 для обох гравців.

4.5.4 Метод nash_equilibrium – Пошук рівноваги Неша

Метод **nash_equilibrium** можна вважати головним, що підтримує бібліотека GameTheoryPy.

Сам метод може працювати в 2 режимах – з форматованим виводом та без. За своєю природою, метод, як зрозуміло з назви, шукає всі рівноваги Неша, що притаманні грі. Значення стратегій, за яких відбувається рівновага, повертаються у вигляді масива пар старатегій. Також, метод може працювати в 2 режимах – з форматованим виводом поверх резунльтату та без виводу.

Для демонстрації використаємо суперечку Віктора та Людмили, описану вище:

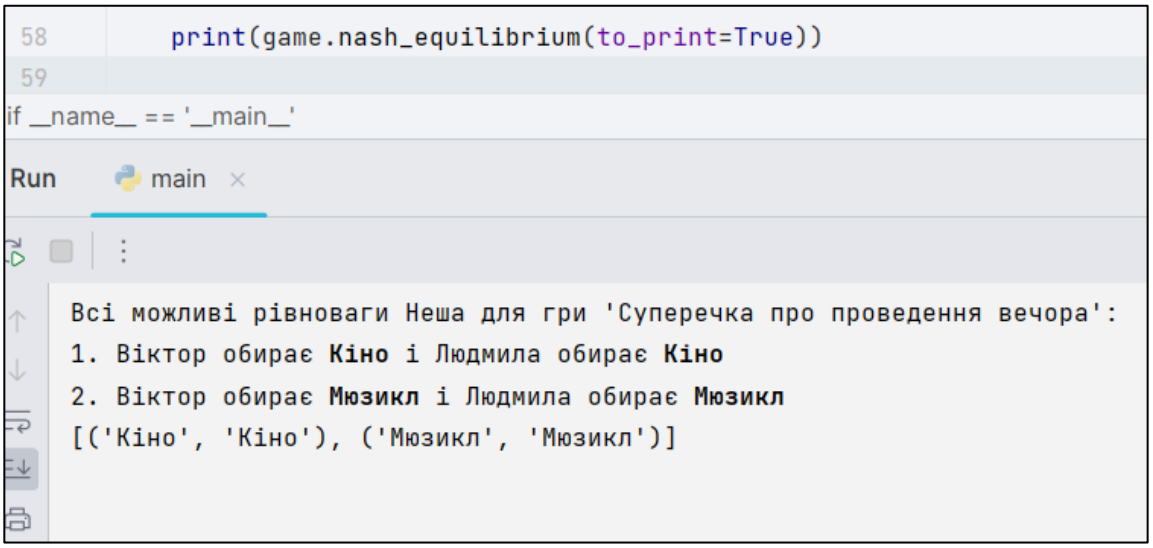


Рисунок 4.19 – Пошук рівноваги Неша

За режим «виводу» відповідає параметр to_print. З його значенням False результатом методу був би лише повернутий масив.

4.6 Публікація пакету на PyPI

Важливим є також публікація пакету на ресурсі PyPI. Як можна бачити, було створено новий зовнішній модуль, який можна використовувати в інших проектах.

Версія: 0.0.1, тобто найперша бета-версія продукту.

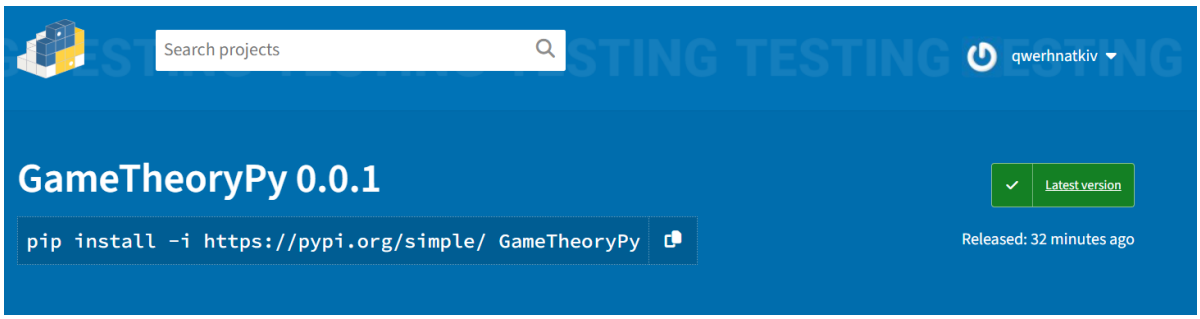


Рисунок 4.20 – Пакет GameTheoryPy на PyPI

Також, разом з пакетом було додано її опис та документацію (яка посилається на github документацію).

Крім того, доступні посилання на домашню сторінку та баг трекер (github репозиторій)

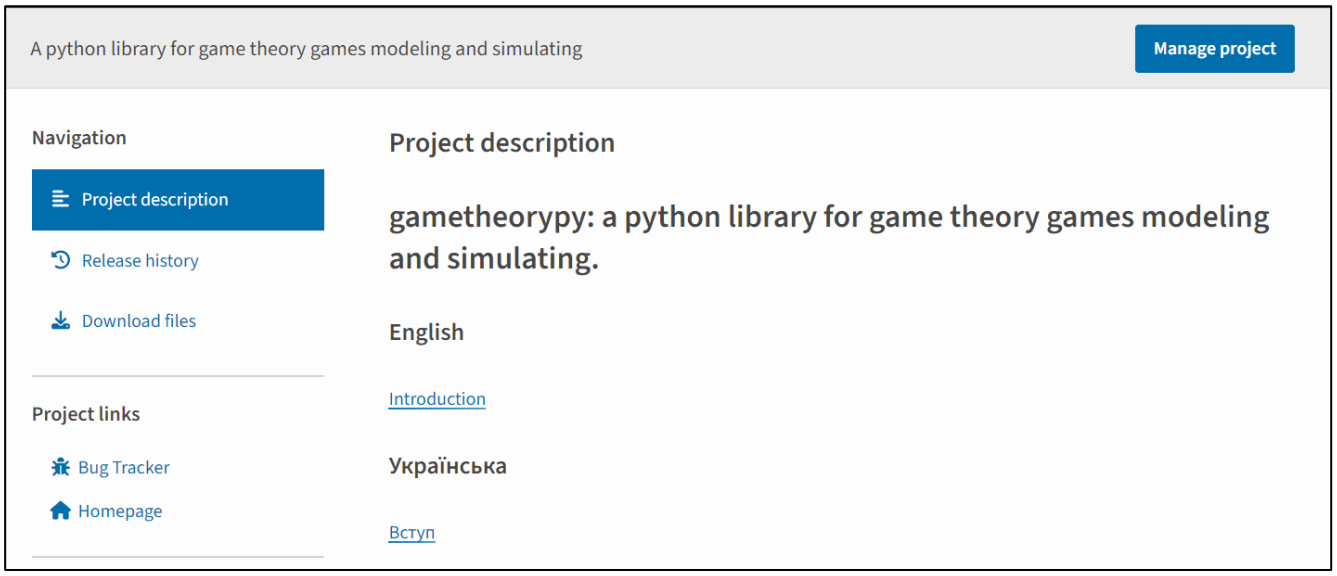


Рисунок 4.21 – Опис проекту

Мною було додано інші додаткові відомості про пакет. В першу чергу, я обрав ліцензію MIT, що надає чіткий дозвіл користувачам повторно використовувати код для будь-яких цілей, іноді навіть якщо код є частиною власного програмного забезпечення.

Також, підтримується адреса автора (мене) для особистих контактів, вимоги по Python версії та специфіка операційної системи (мульти-платформенний продукт).

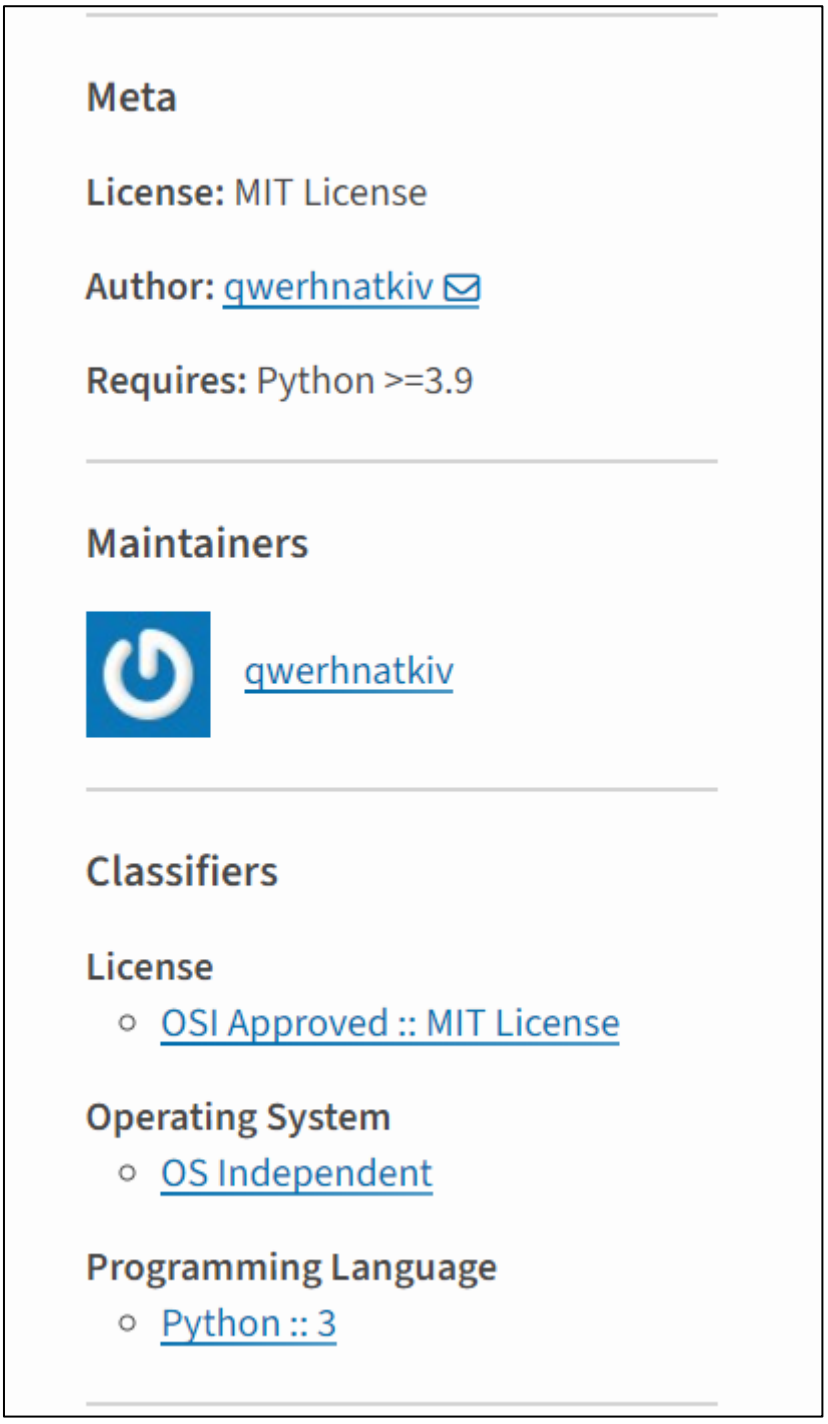


Рисунок 4.21 – Додаткові відомості

4.7 Порівняння з іншими подібними бібліотеками

Для повної оцінки проведеної роботи та результатів цієї роботи важливо порівняти готовий продукт з вже існуючими пакетами Python. Провівши короткий аналіз, я прийшов до висновку, що основними джерелами для роботи з теорією ігор в середовищі Python є бібліотеки **NashPy** та **Gambit**.

NashPy - це Python бібліотека, що підтримується з 2016 року. Основні

особливості бібліотеки NashPy включають наступне:

а) Робота з платіжними матрицями

б) Робота з деревами та розширеними формами ігор

в) Обрахунок рівноваги Неша трьома різними алгоритмами

г) Генерування домінуючих стратегій та відповідей на зіграні ходи

д) Підтримка моделі динаміки реплікаторів, що дозволяє симулювати еволюційні процеси (включно з мутаціями)

Gambit - це широкомасштабний та глобальний набір інструментів для роботи з об'єктами теорії ігор. Вперше бібліотека заявила про себе в середині 1980-х років, ще не мові BASIC. Після цього, історично, вона переписувалась на C, C++ (з підтримкою грантів), та згодом з'явився окремий інтерфейс для мови Python. Було б довго перераховувати все, що підтримується фреймворком Gambit, але основним та унікальним функціоналом є:

а) Інтерактивний, кросплатформенний графічний інтерфейс

б) Інструменти командного рядка для обчислення рівноваг ігор

в) Розширюваність і широкі можливості для доповнення

г) Підтримка роботи з файлами, читання/запис даних об'єктів теорії ігор

д) Пошук рівноваг Неша кількома способами

Взагалі, важко порівнювати «інді-бібліотеку» з масивними інструментами, які надаються бібліотеками вище, але все ще пакет GameTheoryPy має свої очевидні переваги та недоліки.

4.6.1 Переваги

— GameTheoryPy – єдина Python бібліотека про теорію ігор, що підтримує українську мову

— Простота роботи з системою та модифікації її об'єктів

— Унікальна вбудована система форматування, з підтримкою pretty-print в різних форматах та забезпечення не лише технічних результатів в вигляді python-об'єків, а й показ результатів на кожному кроці в консоль

— Широкі можливості для налаштування системи, включно з назвами гри, гравців, їх стратегій тощо

— 100% покриття документацією українською та англійською мовами
— Добре спроектована архітектура системи, що дозволить її розширювати надалі

4.6.2 Недоліки

— Відсутність підтримки читання даних з файлу та запису в файл
— Підтримка лише одного алгоритму пошуку рівноваг Неша, що може відобразитись на продуктивності (особливо враховуючи, що бібліотека написана на мові Python без підтримки NumPy обрахунків)
— Відсутність підтримки деревоподібних ігор
— Відсутність явного графічного інтерфейсу (як забезпечує бібліотека Gambit), а лише підтримка виводу в консоль
— Відсутність вбудованого набору ігор, які можна змоделювати без додаткового вводу користувача

Примітка. Більшість з проблем, описаних вище, будуть вирішені з плином часу, ходом розвитку бібліотеки та її можливостей.

ВИСНОВКИ

Отже, підбивши висновки розробки бібліотеки GameTheoryPy, можна стверджувати, що було виконано всі поставлені цілі. Результатом виконання дипломної роботи стала Python бібліотека, що дозволяє зручно і швидко працювати з основними поняттями теорії ігор та їх програмними моделями. Фреймворк наділений простим та інтуїтивним API, що тільки покращує досвід користувача з ним.

Застосування бібліотек Pandas, tabulate та подекуди NumPy дозволило пришвидшити розробку і зробити її оптимальнішою, адже в межах цих пакетів є багато корисних функцій та об’єктів, що дозволяють розвивати подібний проект швидше (до прикладу, для моделювання платіжних матриць використовувався Pandas DataFrame).

Також, завдяки дотримуванню принципів SOLID та основ проектування можна сказати, що система є відкритою до розширення і додавання нового функціоналу без великих затрат ресурсів. В майбутньому це може допомогти з розвитком бібліотеки і додаванням до вже наявного функціоналу таких можливостей, як:

— Готовий набір основних ігор, що розглядаються в навчальних прикладах. Це можуть бути як і прості приклади («Камінь-ножиці-папір», «Підкидання монетки», «Дилема в’язня» тощо), так і більш складні екземпляри («Задача про переговори», «Задача про поділ торта», «Суспільні блага» та ін. [6])
— Забезпечення роботи з файловою структурою
— Підтримка деревоподібних послідовних ігор за допомогою бібліотеки BinaryTree
— Алгоритм ітеративного видалення домінуючих стратегій для пошуку рівноваги
— Підтримка інших алгоритмів пошуку рівноваги Неша для забезпечення вищої продуктивності продукту
— Моделювання послідовних ходів гравців та інтерактивні реакції на них інших гравців. Можливо залучити нові властивості гравців, такі як

«раціональність», що дозволить з ймовірнісним підходом моделювати відповіді суперників

— Інтерактивний інтерфейс

— Розширення документації до ресурсу ReadTheDocs.io

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Економічні проблеми сталого розвитку / Сумський Державний Університет. – 2018 – 352с.
2. Теорія ігор: курс лекцій, навчальний посібник / Київський Політехнічний Інститут ім. Ігоря Сікорського. – 2022 – 225с.
3. В.О. Корнієнко. МОДЕЛЮВАННЯ ПРОЦЕСІВ У ПОЛІТИКО-КОМУНІКАТИВНОМУ ПРОСТОРІ [Електронний Ресурс] / В.О. Корнієнко, С.Г.Денисюк, А.А. Шиян. – 2008 – С. 8–16. – Режим Доступу: https://fies.vntu.edu.ua/pmba/stf/teach/books/Mod_pol.pdf
4. Robert Martin. Clean Code: A Handbook of Agile Software Crafrmanship 1 st edition. – 2008 – 464c.
5. Pandas: Documentation [Electronic Resource], 2023 – Available From: https://pandas.pydata.org/docs/
6. Wikipedia: List of games in game theory [Electronic Resource], 2023 – Available From: https://en.wikipedia.org/wiki/List_of_games_in_game_theory