МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна технологія, моделі і методи біматричної гри з урахуванням впливу «природи» як третього гравця»

Завідувач

випускаючої кафедри **Довбиш А. С.**

Керівник роботи **Шаповалов С. П.**

Студент групи ІНмз-01С **Мехедок О. П.**

СУМИ 2021

Сумський державний університет

(назва вузу)

Факультет *ЦЗДВФН* Кафедра *Комп'ютерних наук*

Спеціальність *«122 -Комп'ютерні науки»*

Затверджую:

зав.кафедрою ______________________

____

"________" ______________________ 20___р.

ЗАВДАННЯ

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Мехедок Олегу Петровичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *«Інформаційна технологія, моделі і методи біматричної гри з урахуванням впливу «природи» як третього гравця»*

затверджую наказом по інституту від "____" ________________ 20 ___ р. № _____________

2. Термін здачі студентом закінченого проекту (роботи) ________________________________

3. Вхідні данні до проекту (роботи) ____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми. Постановка задачі дослідження. 2) Інформаційний огляд. 3) Математична модель та вибір методу рішення 4) Розробка інформаційного та програмного забезпечення системи*

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

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання ____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	*Аналіз проблеми. Постановка задачі дослідження*		
2.	*Інформаційний огляд*		
3.	*Математична модель та вибір методу рішення*		
4.	*Розробка інформаційного та програмного забезпечення системи*		
5.	*Оформлення пояснювальної записки до дипломної роботи*		

Студент – дипломник ___________________________

(підпис)

Керівник проекту ___________________________

РЕФЕРАТ

Записка: 52 стр., 13 рис., 5 табл.. 4 додатки, 10 джерел інформації.

Об'єкт дослідження – задачі теорії ігор з урахуванням дії природи як третього гравця.

Мета роботи – адаптувати генетичний алгоритм для рішення біматричної гри з урахуванням дій «природи» в якості третього гравця.

Методи дослідження – методи математичного моделювання, генетичний алгоритм, програмування на алгоритмічній мові Java.

Результати – адаптовано генетичний алгоритм для рішення задач теорії ігор з урахуванням впливу природи. Проведена комп'ютерна реалізація та її тестування на конкретних завданнях.

БІМАТРИЧНА ЗАДАЧА, ГРА З ПРИРОДОЮ,
ГЕНЕТИЧНИЙ АЛГОРИТМ, КОМП'ЮТЕРНА РЕАЛІЗАЦІЯ
ТЕСТОВІ РОЗРАХУНКИ

ЗМІСТ

ВСТУП……………………………………………………………….. 5

1 ОГЛЯД ІСНУЮЧИХ АЛГОРИТМІВ ТА МЕТОДІВ РІШЕННЯ…..... 7

1.1 Моделі теорії ігор та їх розв’язання……………….. ………….……….. 7

1.2 Постановка задачі……………………………………………… 10

2 МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ ПРОБЛЕМИ ТА АЛГОРИТМ РІШЕННЯ... 11

2.1 Загальна математична модель ………...……………………….. 12

2.2 Генетичний алгоритм та його опис……..……… ………….……..... 14

2.3 Застосування алгоритму в рішенні біматричної гри з урахуванням впливу природи……..……………………………………… 18

3 КОМПЮТЕРНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ………………... 20

3.1 Опис комп’ютерної реалізації ……………………………… 20

3.2 Тестові розрахунки….………………………………………….. 23

ВИСНОВКИ.……………………………………………………... 28

СПИСОК ЛІТЕРАТУРИ..…………………………………………….. 29

ДОДАТОК А…………………...………………………………. 30

ДОДАТОК Б…………………...……………………………….. 32

ДОДАТОК В…………………..……………………………. 39

ДОДАТОК Г…………………...………………………….…. 45

ВСТУП

В умовах конкуренції виникає потреба узгодження дій фірм, об'єднань, міністерств та інших учасників проектів у випадках, коли їх інтереси не збігаються. У таких ситуаціях теорія ігор допомагає знаходити найкраще рішення для поведінки учасників, зобов'язаних погоджувати дії при зіткненні інтересів. Її можна розглядати як інструментарій, що послуговує підвищенню ефективності вибору стратегічної поведінки учасників конкуренції [1-7]. У деяких аспектах теорія ігор є першим наближенням для оптимальне прийняття рішень незалежними та конкуруючими сторонами в стратегічній обстановці.

Це має велике значення при вирішенні завдань в промисловості, сільському господарстві, на транспорті, в торгівлі, особливо при укладанні договорів з іноземними партнерами на будь-яких рівнях. Так, можна визначити науково обґрунтовані рівні зниження роздрібних цін і оптимальний рівень товарних запасів, вирішувати завдання екскурсійного обслуговування і вибору нових ліній міського транспорту, завдання планування порядку організації експлуатації родовищ корисних копалин в країні та ін. Використовуючи теорію ігор, можна викласти реальні сценарії для таких ситуацій, як цінова конкуренція та випуск продуктів (і багато інших), і передбачити їхні результати [1-3]. Ця взаємозалежність змушує кожного гравця враховувати можливі рішення або стратегії іншого гравця при формулюванні стратегії. Рішення гри описує оптимальні рішення гравців, які можуть мати подібні, протилежні або змішані інтереси, а також результати, які можуть бути результатом цих рішень.

Класичною стала задача конкуренції фірм на ринку, коли їх інтереси перетинаються й кожна із них своїми діями впливає на успіх конкурента. Метод теорії ігор можна застосовувати для розв’язання подібного роду конфліктів, при цьому кожний з учасників одержить рекомендації.

Кожного разу, коли у нас виникла ситуація з двома або більше гравцями, яка пов’язана з відомими виплатами чи кількісними наслідками, ми можемо використовувати теорію ігор, щоб допомогти визначити найбільш ймовірні результати. У бізнесі теорія ігор корисна для моделювання конкуруючої поведінки між

економічними агентами. Підприємства часто мають кілька стратегічних варіантів, які впливають на їх здатність отримувати економічний виграш. Наприклад, підприємства можуть зіткнутися з такими дилемами, як: відмовитися від існуючих продуктів чи розробити нові, знизити ціни порівняно з конкурентами чи застосувати нові маркетингові стратегії.

Зазвичай теорію ігор можна класифікувати як розділ математичного моделювання для вивчення конфліктних ситуацій. Це означає, що можна виробити оптимальні правила поведінки кожної сторони, що бере участь у вирішенні конфліктної ситуації й вказати їм план дій для досягнення максимальної корисності в умовах перебування в цій ситуації[3-7].

В економіці, наприклад, виявився недостатнім апарат математичного аналізу, що займається визначенням екстремумів функцій. З'явилася необхідність вивчення так званих оптимальних мінімаксних і максимінних рішень. Отже, теорію ігор можна розглядати як новий розділ оптимізаційного підходу, що дозволяє вирішувати нові завдання при прийнятті рішень.

Важливим аспектом, який мало вивчений при моделюванні конфліктних ситуацій засобами теорії ігор, залишається дослідження впливу «природи» на взаємодію сторін конфлікту. Під природою розуміється спроможна можливість покупців в придбанні продуктів конкуруючих фірм. В таких моделях природу можна додати в постановку задачі, як третього гравця, що грає не як розумний конкурент, а є конкурентом з невизначеними намірами, але маючими набір стратегій, що визначаються різними варіантами величини попиту. Саме такі моделі є предметом подальших досліджень.

1 ОГЛЯД ІСНУЮЧИХ АЛГОРИТМІВ ТА МЕТОДІВ РІШЕННЯ

1.1 Моделі теорії ігор та їх розв’язання

Теорія ігор - це теоретична основа для дослідження та аналізу будь-яких взаємин в конфліктних ситуаціях серед гравців-конкурентів. У деяких аспектах теорія ігор є наукою про стратегію або, принаймні, про оптимальне прийняття рішень незалежними та конкуруючими фірмами в стратегічній обстановці [1-3].

Рисунок 1. 1 – Представлення теорії ігор одним рисунком

Основними моделями теорії ігор, коли відсутні або заборонені домовленості між гравцями, є модель біматричної гри, що надає можливість оцінити який виграш зможе отримати кожний із гравців при обранні ними оптимальної поведінки (вибір розподілу стратегій, що приводить до максимізації прибутку).

Будь-яка теоретико-ігрова модель має відбивати, хто як конфліктує, і навіть, хто у якій формі зацікавлений у тому чи іншому результаті конфлікту. Сторони, що діють у конфлікті, називатимемо гравцями, а рішення, які здатні приймати гравці, - стратегіями.

Використовуючи теорію ігор, можна викласти реальні сценарії для таких ситуацій, як цінова конкуренція та випуск продуктів (і багато інших), і передбачити їхні результати.

Біматричні ігри двох осіб – це ігри з ненульовою сумою (за класифікацією теорії ігор [1-3]), в яких кожна гравець має скінчену кількість чистих стратегій.

Основна форма опису моделей таких ігор задається платіжною матрицею, що відображає платежі (виграші, корисність) кожного з учасників гри, в залежності від вибору ними стратегій дій з множини можливих.

Складові елементи біматричної моделі :$\langle N, S_N, U_N \rangle$, що представляють собою:

N – гравці або учасники гри, в подальшому приймемо N=2.

S_N – профілі стратегій гравців (набір альтернатив з непорожньої множини).

U_N – функції виграшів гравців: $U_N : X \times Y \rightarrow R$, де $X, Y \in S_N$

Основою розв’язання гри є пошук рівноваги – такого вибору профіля стратегій гравців, з якого кожному з учасників конфлікту не вигідно виходити, коли всі інші в ній залишаються.

Однією з найпривабливіших концепцій теорії ігор є поняття рівноваги Неша, за що Джон Форбс Неш отримав Нобелівську премію в сфері економіки.

Для гри $G_{bi} =\{S_i, U_i, i \in N \}$, стратегії $(s^*_1, s^*_2, \ldots s^*_N)$ утворюють рівновагу Неша, якщо для кожного гравця і стратегія s^*_i є його найкращою відповіддю-реакцією на набір стратегій s^*_{-i} інших гравців, тобто s^*_i розв’язує проблему

$$\max_{s_i \in S_i} U_i(s_1{}^*, \ldots, s_{i-1}{}^*, s_i, s_{i+1}{}^*, \ldots, s_n{}^*) \qquad (1.\ 1)$$

Теорема Неша. *В кожній скінченій біматричній грі існує рівновага в чистих або змішаних стратегіях.*

Відомим є факт, що не тільки рівновага Неша може визначатись в біматричних іграх. Існують і інші різновидності рівноваги. На рисунку 1. 2 представлені можливі ситуації рівноваг.

Існують ігри з природою у яких є лише один учасник, що максимізує свій прибуток. Ігри з природою – математичні моделі, у яких вибір рішення залежить про об'єктивну реальність. Наприклад, купівельний попит, стан природи тощо. «Природа» – це узагальнене поняття не переслідує своїх цілей противника. У такому разі для вибору оптимальної стратегії використовують кілька критеріїв. Розрізняють два види завдань у іграх із природою:

- ❖ завдання прийняття рішень за умов ризику, коли відомі ймовірності, із якими природа приймає кожен із можливих станів;
- ❖ завдання про прийняття рішень в умовах невизначеності, коли немає можливості отримати інформацію про ймовірність появи станів природи.

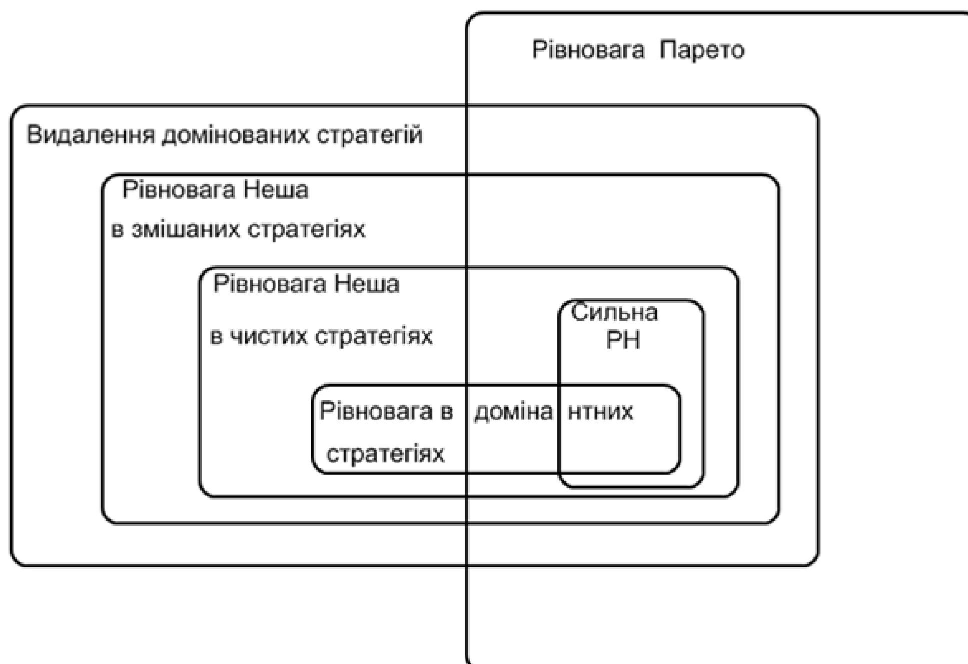


Рисунок 1. 2 – Види рівноваги у біматричній грі

Ігри з природою застосовуються для аналізу економічних ситуацій, оцінки ефективності прийнятих рішень та вибору найбільш кращих альтернатив, у яких цей вибір пов'язаний з сукупністю невизначених чинників довкілля, іменованих «природа». Тому термін «природа» характеризує якусь об'єктивну дійсність, яку слід розуміти буквально, хоча цілком можуть зустрічатися ситуації, в яких гравцем справді може бути природа (наприклад, погодні умови чи стихійні лиха).

Наприклад, деякій фірмі потрібно ухвалити план своїх дій про випуск на ринок для продажу товарів. Можливі ситуації на ринку: 1. Попит на ці товари відомий точно (буває крайнє рідко); 2.Відомим являється статистичний розподіл можливих значень попиту (буває найчастіше); 3. Фірма має інформацію лише

про межі, в яких знаходиться попит, але жодних навіть ймовірнісних міркувань про його майбутні значення немає.

У макроекономіці також використовується модель сукупного попиту-сукупної пропозиції, щоб зобразити, як кількість загального випуску та сукупний рівень цін можуть бути визначені в рівновазі.

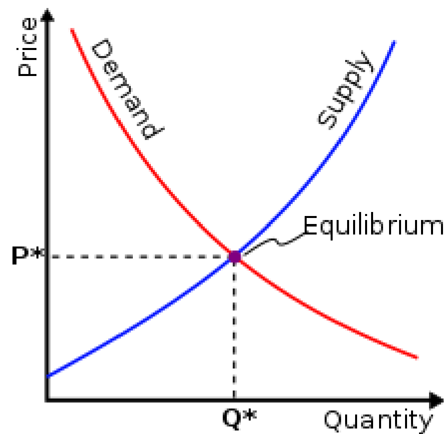


Рисунок 1. 3 – Точка рівноваги між попитом і пропозицією

З точки зору побудови єдиної моделі взаємовідношей між гравцями в моделях теорії ігор будемо розглядати гру з «природою», яка антагоністичну гру, в якій поведінка природи описується ситуацією 2. А торгова фірма в рамках цієї моделі обирає оптимальні стратегії з того, що відомий статистичний розподіл можливих значень попиту.

1.2 Постановка задачі

Прийдемо до наступної постановки задачі.

Дослідити вплив «природи» в якості третього гравця в результат біматричної гри двох гравців..

. Для досягнення мети провести наступні дослідження:

1. *Провести адаптацію генетичного алгоритму до рішення поставленого завдання.*
2. *Побудувати математичну модель та створити алгоритм розв'язку поставленого завдання.*
3. *Створити комп'ютерну реалізацію проведених досліджень та провести тестові випробування.*

4. *Проаналізувати одержані результати та зробити висновки.*

2 МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ ПРОБЛЕМИ ТА АЛГОРИТМ РІШЕННЯ

Розглянемо моделювання дій гравців, які мають на меті вибір оптимальних стратегій поведінки в умовах конфлікту. Стратегія - це сукупність правил, що визначають вибір варіанта дій при кожному особистому ході в залежності від ситуації, що склалася. Оптимальна стратегія гравця – стратегія, що забезпечує найкраще становище у цій грі, тобто. максимальний виграш. Якщо гра повторюється неодноразово і містить, крім особистих, випадкові ходи, оптимальна стратегія забезпечує максимальний середній виграш.

У грі беруть участь два суб'єкти, назвемо їх "гравець А" і "гравець В". В якості названих можуть бути як конкуруючі фірми, що змагаються за ринки збуту продукції, так і будь-які агенти господарської чи управлінської діяльності.

У кожному акті гри гравець А може прийняти одне з m рішень, гравець В – одне з n. Виникає ситуація у грі, що характеризується для гравців платіжними здобутками (виграшами). Величини a_{ij} – це корисність в цій ситуації гравця А, коли він застосовує в неї i-ту стратегію, а гравець В — j–ту, а b_{ij} все ж те саме, тільки стосовно гравця В. Сукупність таких ситуацій й виграші гравців складають платіжну матрицю гри: G_{bi} – матриця біматричної гри

$$G_{bi} = \begin{pmatrix} a_{11}, b_{11} & a_{12}, b_{12} & \dots & a_{1n}, b_{1n} \\ a_{21}, b_{21} & a_{22}, b_{22} & \dots & a_{2n}, b_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1}, b_{m1} & a_{m2}, b_{m2} & \dots & a_{mn}, b_{mn} \end{pmatrix} \quad (2.\ 1)$$

В цій матриці, подвійний елемент вказує на корисність (виграш, винагорода), яку одержує кожний з гравців в кожній із ситуацій, які виникають у грі.

Таку гру описує математична модель

$$U_A(A, S_A, S_B) = \sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij} p_i\, q_j,\ U_B(B, S_A, S_B) = \sum_{i=1}^{m}\sum_{j=1}^{n} b_{ij} p_i\, q_j,$$

$$\sum_{i=1}^{m} p_i = 1, \quad \sum_{j=1}^{n} q_j = 1 \qquad (2.\ 2)$$

В цій моделі присутні змішані стратегії гравців, що представляють собою ймовірності застосувань стратегій кожного гравця у грі, тобто відображують стратегії гравців на ймовірнісне поле: $S_A(p_1,p_2,\ldots p_m)$, $S_B(q_1,q_2,\ldots q_n)$.

Стратегії $S*_A(p*_1,p*_2,\ldots p*_m)$, $S*_B(q*_1,q*_2,\ldots q*_n)$ визначають рівноважну ситуацію у грі, якщо одночасно виконуються нерівності

$$U_A(A,S^*_A,S^*_B) \geq U_A(A,S_A,S^*_B);\ U_B(B,S^*_A,S^*_B) \geq U_B(B,S^*_A,S_B) \quad (2.\ 3)$$

Гру з природою представимо як антагоністичну гру, що задається матрицею гри

$$Gp = \begin{pmatrix} a\hat{}_{11} a\hat{}_{12} \ldots a\hat{}_{1n} \\ a\hat{}_{21} a\hat{}_{22} \ldots a\hat{}_{2n} \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ a\hat{}_{m1} a\hat{}_{m2} \ldots a\hat{}_{mn} \end{pmatrix} \quad (2.\ 4)$$

в якій елементи $a\hat{}_{ij}$ виграші гравця А, що застосовує стратегію A_i, а природа в цей час находилась в стані j.

Таку гру описує математична модель

$$U(Gp,S_A,S_B) = \sum_{i=1}^{m}\sum_{j=1}^{n} a\hat{}_{ij} p\hat{}_i\, q\hat{}_j, \quad \sum_{i=1}^{m} p\hat{}_i = 1, \quad \sum_{j=1}^{n} q\hat{}_j = 1 \quad (2.\ 5)$$

Додаткові умови, що випливають з теореми про активні стратегії

$$U(Gp,S_A,S^*_B) \leq U(Gp,S^*_A,S^*_B) \leq U(Gp,S^*_A,S^*_B) \quad (2.\ 6)$$

Представлена модель вимагає аналізу та конкретизації різних варіантів взаємовідносин між гравцями та природою.

2.1 Загальна математична модель

В грі приймають участь три гравці: гравець А грає з В біматричну гру, а третій гравець – природа грає матричну антагоністичну гру як з гравцем А, так і з гравцем В. Загальна математична модель складається з сукупності співвідношень (2.1)-(2.6). Якщо грати біматричну гру без впливу природи, то потрібно розв’язати (2.1-2.3), Якщо грати кожному з гравців тільки з природою, то рішення задач (2.4)-(2.6) опише цю ситуацію.

На рисунку 2.1 показані моделі різних інформаційних станів гри та показані варіанти взаємодій.

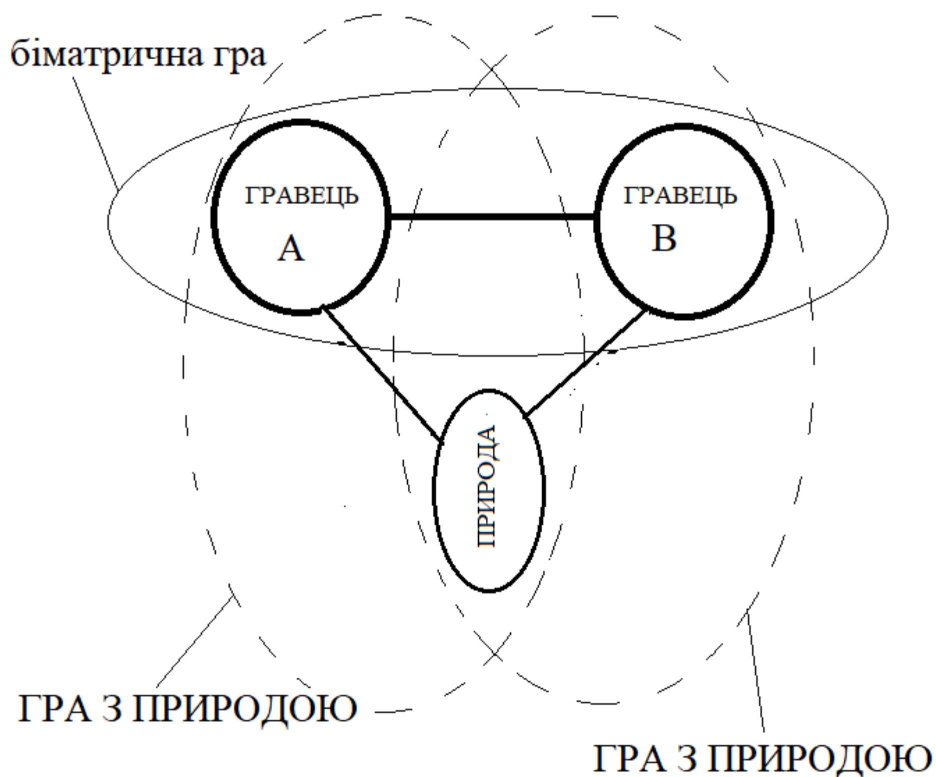


Рисунок 2.1 – Інформаційні стани гри

На рисунку 2.2 представлено біматричну гру в просторі дій гравців (по осі 0x - A , 0y – B). Точка рівноваги Неша R як перетин ліній рівня виграшів. Координати P і Q – максимальні можливі виграші гравців відповідно A і B.

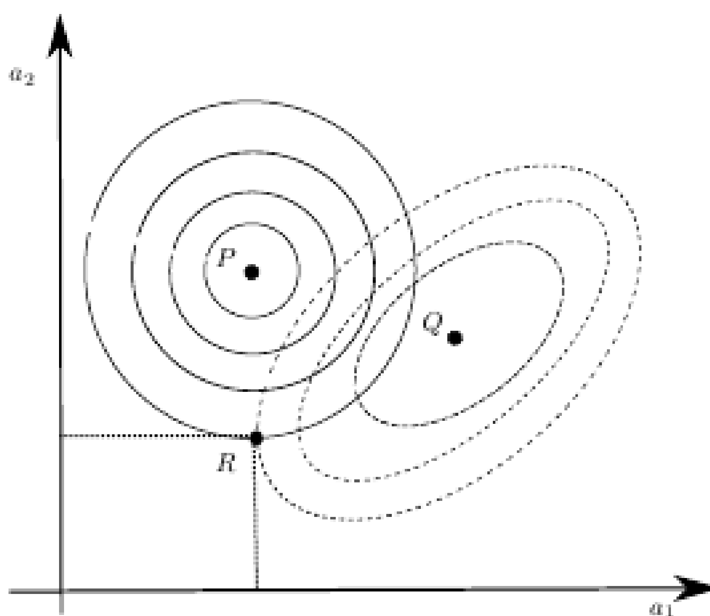


Рисунок 2. 2 – Точка рівноваги Неша R в просторі дій гравців

Важливою властивістю змішаної стратегії рівноваги Неша є те кожна дія на підтримку рівноважної змішаної стратегії будь-якого гравця є найкращою відповіддю і приносить цьому гравцеві таку ж виплату.

2.2 Генетичний алгоритм та його опис

Генетичний алгоритм належить до класу натхненних природою алгоритмів й заснований на теорії еволюції в природі, сучасні основи якої були закладені Чарльзом Дарвіном, а також іншими менш відомими дослідженнями, такими як Жан-Батист де Ламарк. Відповідно до загального розуміння теорії еволюції, форми життя зазнають постійної (само)оптимізації, намагаючись пристосуватися до мінливого середовища проживання [5, 6].

Генетичннй алгоритми - адаптивні методи пошуку, які останнім часом часто використовуються для вирішення задач функціональної оптимізації. Вони засновані на генетичних процесах біологічних організмів: біологічні популяції розвиваються протягом кількох поколінь, згідно законів природного відбору і за принципом "виживає найбільш пристосований"[5].

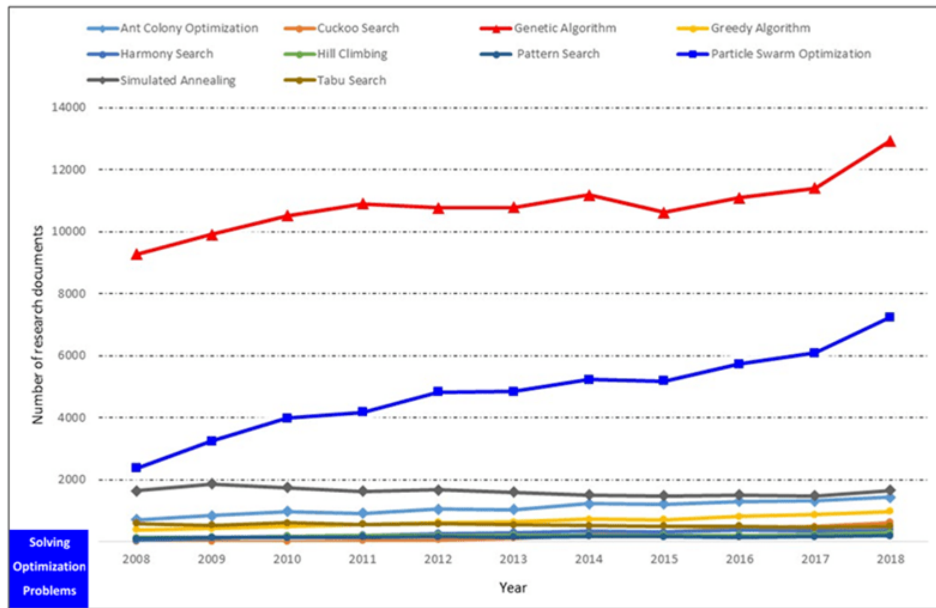
В даний час під терміном "генетичні алгоритми" ховається не одна модель, а досить широкий клас алгоритмів, часом мало схожих один від одного. Дослідники експериментували з різними типами уявлень, операторів кросовера і мутації, спеціальних операторів, і різних підходів до відтворення та відбору.

Хоча модель еволюційного розвитку, застосовувана в ГА, сильно спрощена в порівнянні зі своїм природним аналогом, тим не менше ГА є досить потужним засобом і може з успіхом застосовуватися для широкого класу прикладних задач, включаючи ті, які важко, а іноді і зовсім неможливо, вирішити іншими алгоритмами.

Генетичний алгоритм (GA) є одним із найпопулярніших алгоритмів стохастичної оптимізації, які часто використовуються для вирішення складних великомасштабних задач оптимізації в різних областях [5, 6].

Про популярність генетичного алгоритму говорить хоча би ця діаграма, що представлена на рисунку 2.3

Popularity of Genetic Algorithm



(Scopus database, accessed on 30th September 2019)

Рисунок 2. 3 – Шкала популярності алгоритмів

Традиційним вважається ГА, представлений за псевдокодом та на схемі [6].

ПОЧАТОК / * генетичний алгоритм * /

Створити початкову популяцію

Оцінити пристосованість кожної особини

зупинення: = FALSE

ПОКИ НЕ зупинення ВИКОНУВАТИ

ПОЧАТОК / * створити популяцію нового покоління * /

ПОВТОРИТИ (розмір_популяціх / 2) РАЗ

ПОЧАТОК / * цикл відтворення * /

Вибрати дві особини з високою пристосованістю з попереднього покоління для схрещування

Схрестити вибрані особини і отримати двох нащадків

Оцінити пристосованості нащадків

Помістити нащадків в нове покоління

КІНЕЦЬ

ЯКЩО популяція зійшлася ТО зупинення: = TRUE

КІНЕЦЬ

КІНЕЦЬ

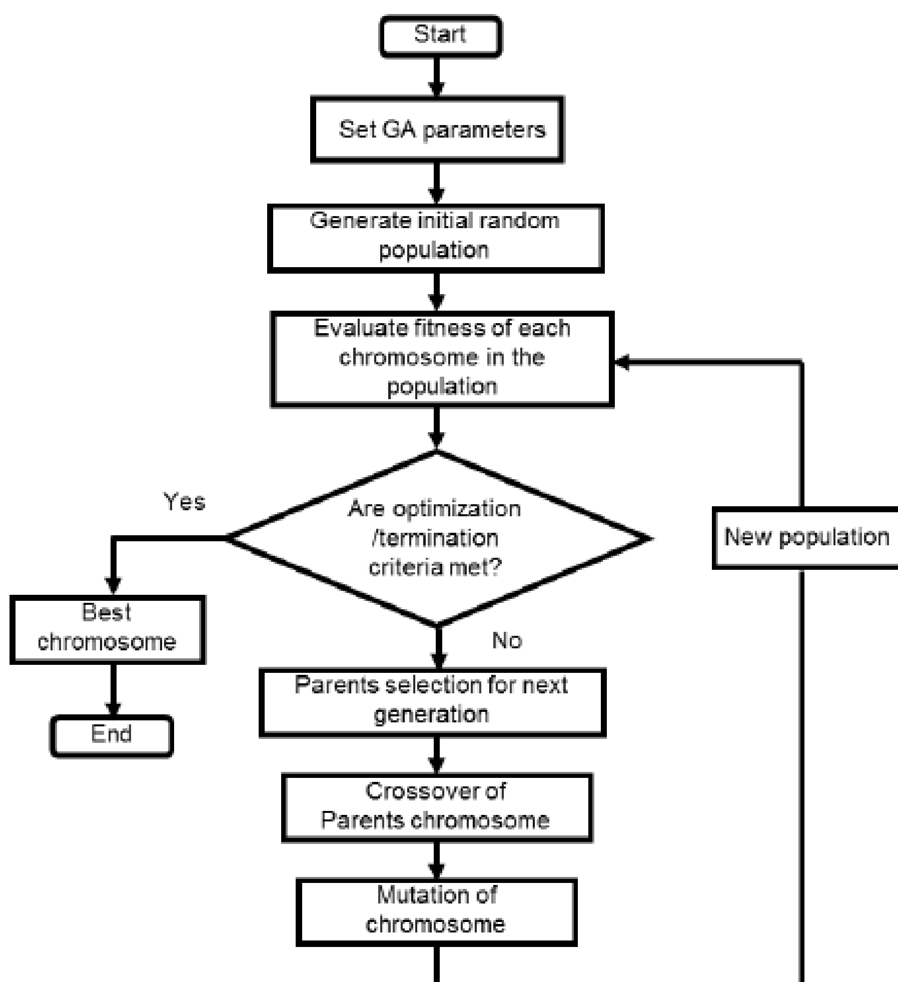


Рисунок 2. 4 – Класична схема генетичного алгоритму

Основними блоками генетичного алгоритму являються.

1. *Створення початкової популяції.*

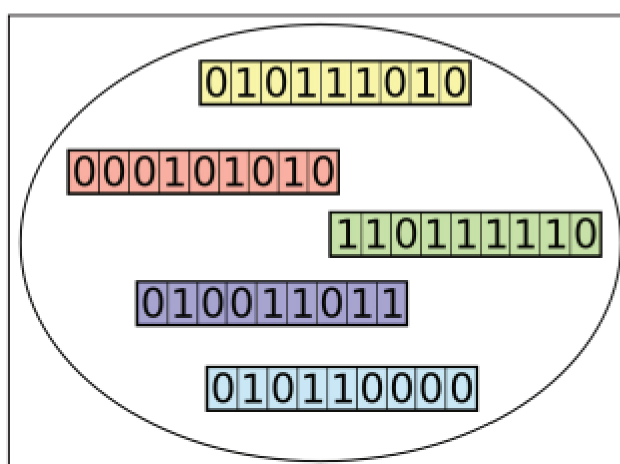


Рисунок 2. 5 – Початкова популяція хромосом

ГА працює над сукупністю, що складається з деяких рішень, де розмір популяції (розмір популяції) є кількістю рішень. Кожне рішення називається індивідуальним. Кожен окремий розчин має хромосому. Хромосома представлена як набір параметрів (особливостей), що визначає індивіда. Кожна хромосома має набір генів. Кожен ген представлений якимось чином, наприклад, у вигляді рядка з 0 і 1, як показано на рисунку 2.5.

2. *Функція пристосованості (Fitness функція).*

Для відбору найкращих особин використовується фітнес-функція. Результатом функції пристосованості є значення пристосованості, що представляє якість рішення. Чим вище значення придатності, тим вище якість рішення. Відбір найкращих особин на основі їхньої якості застосовується для створення так званого пулу, де особина вищої якості має вищу ймовірність бути обраною в пул для схрещування.

3. *Операція схрещування (Crossover (recombination))*

На основі відібраних особин для схрещування підбирають «кращих» батьків. Схрещування можна проводити за різними схемами. Один з варіантів представлений на рис. 2.6.

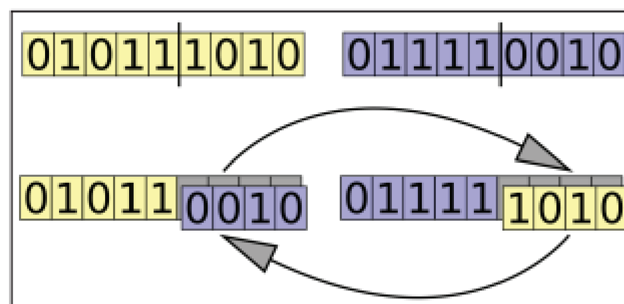


Рисунок 2. 6 – Операція схрещування хромосом

4. *Мутація.*

Наступним оператором варіації є мутація. Для кожного потомства виберіть кілька генів і змініть його значення. Мутація залежить від представлення хромосоми, але вам вирішувати, як застосувати мутацію. Якщо кодування є двій-

ковим (тобто простір значень кожного гена має лише два значення 0 і 1), переверніть значення бітів одного або кількох генів. Приклад мутації показано на рисункі 2.7

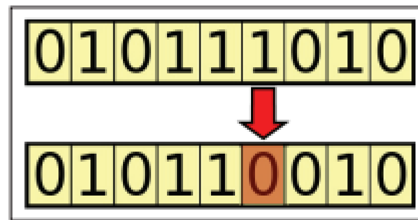


Рисунок 2. 7 – Операція мутації хромосом

5. *Умова припинення.*

Цей процес генерації повторюється до тих пір, поки не буде досягнута умова припинення. Поширені умови припинення:

- Знайдено рішення, яке задовольняє мінімальним критеріям;
- Досягнута фіксована кількість поколінь;
- Призначений бюджет (час на обчислення/гроші) досягнуто;
- Придатність рішення з найвищим рейтингом досягла або досягла такого плато, що послідовні ітерації більше не дають кращих результатів.

2.3 Застосування алгоритму в рішенні біматричної гри з урахуванням впливу природи

Адаптуємо генетичний алгоритм під рішення поставленого завдання.

Блок № 1. Створення початкової популяції.

Кожна генеруєма особина (хромосома) має набір генів, роль яких будуть виконувати змішані стратегії гравців - $S_A(p_1, p_2, p_3,\dots,p_n)$; $S_B(q_1,q_2,q_3,\dots,q_n)$.

При генерації будемо притримуватись властивості

$$\sum_{i=1}^{m} p_i = 1, \quad \sum_{j=1}^{n} q_j = 1$$

Блок №2. Fitness функція

Функція пристосованості являє собою виграші гравців

$$U_A(A, S_A, S_B) = \sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}p_i\, q_j, U_B(B, S_A, S_B) = \sum_{i=1}^{m}\sum_{j=1}^{n} b_{ij}p_i\, q_j,$$

За цими функціями відбираються 60% кращих для схрещування та 40% інших для мутації.

Блок №3. Схрещування.

Схрещування відбувається за наступною процедурою.

Відбираються дві хромосоми (нагадуємо, що вони мають гени, що складаються з набору змішаних стратегій), вибирається точка розділу у хромосом, й батьківські особи міняються частинами. Виникають нащадки, яким передаються гени батьків.

Блок № 4. Мутація.

У інших 40% хромосом, які не схрещувались, обирається ген, що має найбільше значення й переноситься до нащадку на те ж саме місце. До інших генів проводиться процедура мутація шляхом додавання до одного з них (вибирається випадково) визначеного числа (0,05), а від іншого це число віднімається і вони оновлені переходять до нащадкової особини.

Блок № 5. Блок призупинення алгоритму.

Умову останова алгоритму виберемо за принципом призначення визначеного числа зміни поколінь при виконанні алгоритму.

Якщо умова ще не сталася, переходимо до схрещування на мутації і знову нове покоління направляємо на селекцію.

3 КОМПЮТЕРНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Опис комп'ютерної реалізації

Український профільний ресурс DOU.UA провів в 2021 році чергове щорічне опитування про мови програмування - https://habr.com/ru/post/543346/

На рис. 3.1 представлено результати цього опитування по рейтингу мов програмування для комерційного використання:

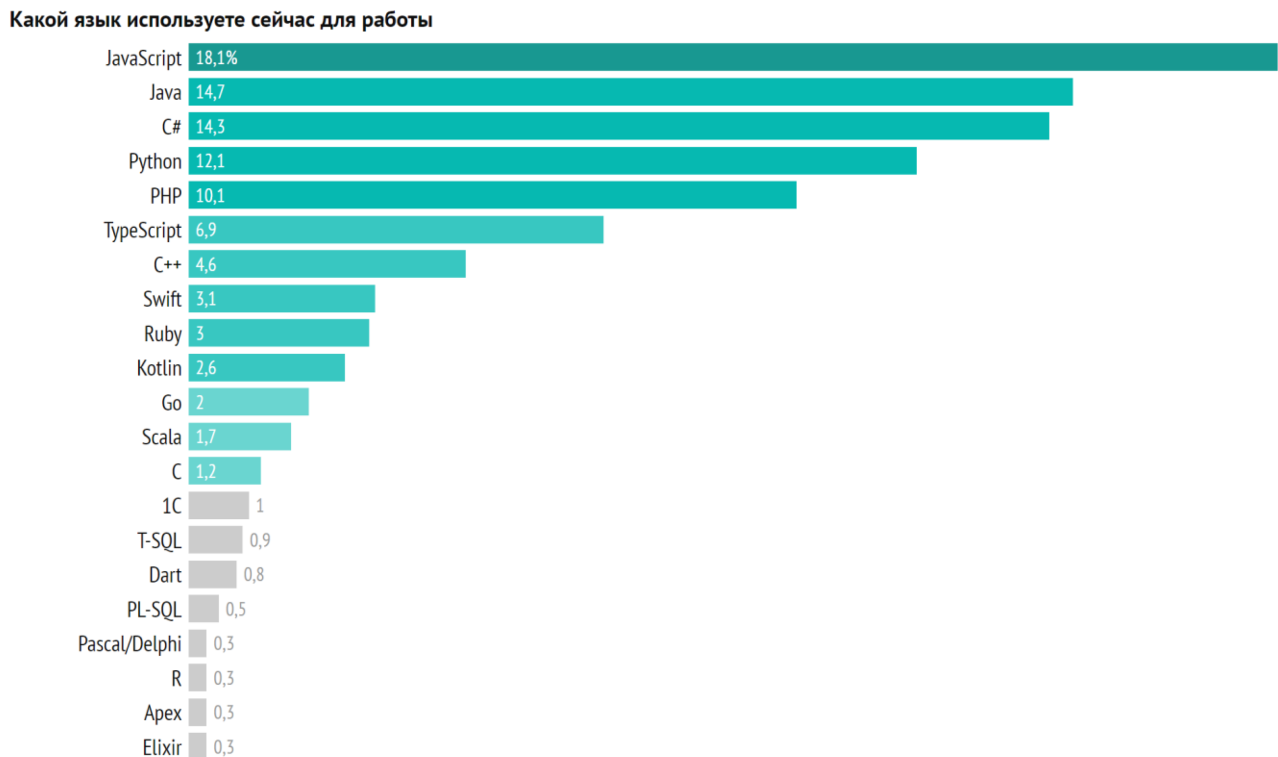


Рисунок 3.1 – Результати опитування комерційного використання мов програмування

З гістограми явну перевагу має Java – об'єктно-орієнтована мова, заснована на класах, та розроблена так, щоб бути легкою в реалізації. Це мова програмування загального призначення, призначена для того, щоб користувачі-програмісти могли писати один раз, запускати в будь-якому місці (WORA), що означає, що скомпільований код Java може працювати на всіх платформах, які підтримують Java, без необхідності перекомпіляції.

Синтаксис Java подібний до C і C++, але має менше засобів низького рівня, ніж будь-який з них. Середовище виконання Java надає динамічні можливості (такі як відображення та модифікація коду під час виконання), які зазвичай недоступні в традиційних скомпільованих мовах.

Рейтинг мов програмування зумовив вибір саме Java для комп'ютерної реалізації досліджень магістерської роботи.

Вхідними даними реалізації є адаптація генетичного алгоритму для розв'язання біматричної гри з урахуванням «природи» та описана в п. 2.3.

Математичні моделі надані в п.

Комп'ютерна реалізація представлена в додатках А, Б, В, Г з використанням класів та програмним кодом-шаблоном в якому реалізовані функції та методи.

Додаток А – реалізовано class Main, в якому описуються вхідні матриці проекту та описуються які ігри будуть аналізуватися. Нижче приведено фрагмент з заданням вхідних матриць

```
public class Main {
    public static void main(String[] args) {
        double[][] valuesA = new double[][]{
                {2, 1, 4},
                {3, 1, 2},
                {1, 0, 3}
        };
        double[][] valuesB = new double[][]{
                {1, 1, 2},
                {4, 2, 3},
                {3, 2, 1}
        };
        double[][] valuesNature = new double[][]{
                {8, 4, 2},
                {2, 8, 4},
                {1, 2, 8}
        };
```

Додаток Б – реалізовано class GeneticAlgorithmBimatrixGame. В цьому додатку вирішується генетичний алгоритм розрахунку біматричної гри без втручання природи. Нижче представлено фрагмент, що задає початкову популяцію

```
public List<double[]> launch() {
    /* Population of 100 chromosomes */
    List<Chromosome> chromosomes = createChromosomes(100);
    /* 50 generations */
    System.out.println("Bimatrix game:");
    double[] Pi = new double[0];
    double[] Ki = new double[0];
```

Додаток В – реалізовано class GeneticAlgorithmZeroSumGame. В цьому додатку вирішується генетичний алгоритм розрахунку гри з природою.

Додаток Г – реалізовано class GeneticAlgorithmTwoPlayersAndNatureGame.

В цьому додатку вирішується генетичний алгоритм розрахунку гри трьох гравців одночасно: гравців А та В, та «Природи», що своїми діями впливає на результат біматричної гри.

Результати розрахунків представлені наступним фрагментом коду

```
    double[] strategyA = new double[3];
    double[] strategyB = new double[3];
    System.out.println("strategyAgainstNature: " +
Arrays.toString(strategyAgainstNature));
    System.out.println("Player A: " + Arrays.toString(bimatrixStrategy.get(0)));
    System.out.println("Player B: " + Arrays.toString(bimatrixStrategy.get(1)));
    for (int i = 0; i < 3; i++) {
        strategyA[i] = 0.6 * bimatrixStrategy.get(0)[i] + 0.4 *
strategyAgainstNature[i];
        strategyB[i] = 0.6 * bimatrixStrategy.get(1)[i] + 0.4 *
strategyAgainstNature[i];
    }
    System.out.println("Strategy of player A: " + Arrays.toString(strategyA));
    System.out.println("Strategy of player B: " + Arrays.toString(strategyB));
```

3.2 Тестові розрахунки

Для проведення тестових розрахунків, змоделюємо гру, як економічну ситуацію наступним чином.

Постановка задачі. Дві торговельні фірми А і В надають послуги населенню в продажі товарів трьох видів П1, П2, П3. Їх взаємозв'язки (гра) описується моделлю біматричної гри з матрицею

Таблиця 3.1 Матриця виграшів гравців у біматричній грі

Стратегії гравця В		**П1**	**П2**	**П3**
Стратегії Гравця А	**П1**	**2,1**	**1,1**	**4,2**
	П2	**3,4**	**1,2**	**2,3**
	П3	**1,3**	**0,2**	**3,1**

Тобто кожний з гравців має у біматричній грі між собою по три стратегії, а саме продажу продукції П1, або П2, або П3. Прибуток від продажу в умовних одиницях указаний в матриці через кому, перше число відповідає фірмі А, друге фірмі В. Наприклад, випускаючи на продаж одночасно продукцію типу П1 отримаємо за матрицею результат 2, 1, що буде означати, що в цьому випадку А одержить 2 умовних одиниці прибутку, а В отримає 1 у. о.

Але одержання прибутків фірм залежить ще від того, який попит на продукцію існує в цей час на ринку продажі й яка купівельна спроможність у населення. Ці взаємини задаються матрицею гри з «природою»:

Таблиця 3.2 Матриця виграшів гравців у грі з природою

Стратегії «природи»		**М1**	**М2**	**М3**
Стратегії гравця А	**П1**	**8**	**2**	**1**
	П2	**4**	**8**	**2**
	П3	**2**	**4**	**8**

Кожна з торгових фірм при розробці варіантів плану продажу товарів на майбутньому ярмарку з урахуванням мінливої кон'юнктури ринку та попиту покупців має однакові можливості, що описуються саме матрицею 3.2. Але природа виступає як гравець, що грає непередбачувано. Така ж саме матриця приймається й для опису доходів гравця В.

Припустимо, що ми не маємо інформації про ймовірність появи станів природи, але нам відомо, що цей стан може бути в трьох варіантах (3-х стратегіях) – М1, М2, М3. Й прибутки фірми в залежності від кожного стані природи нам відомі, вони саме й задані матрицею 3.2.

Приймемо такий підхід до прийняття рішень (вибору альтернативи) є запровадження гіпотез про поведінку середовища. Введемо гіпотеза має дозволяти кожної альтернативи чисельно оцінити пов'язані з нею наслідки, отже, і порівняти будь-які дві альтернативи. Однією з найважливіших гіпотез такого типу є гіпотеза антагонізму. Отже гра з природою є антагоністичною грою.

Завдання дослідження.

Дослідити вплив природи на біматричну гру двох гравців. Для цього провести порівняння декількох варіантів розвитку подій.

1. Розрахувати біматричну гру між гравцями без урахування впливу природи.
2. Розрахувати модель гри, коли спочатку гравці розігрують біматричну гру між собою, та одержують результати гри Rbi, а потім кожний з них грає з природою, та одержує результат Rp. Одержані результати ігор усереднюємо за формулою, та одержуємо вихідний результат R:

$$R = 0,6^* Rbi + 0,4^* Rp \qquad (3.1)$$

3. Третю модель вирахуємо в представлені того, що гра відбувається одночасно трьох гравців, причому одна з них біматрична, а дві інші антагоністичні. В такій грі функцію пристосовності в генетичному алгоритмі визначимо за формулою

$$F = 0,6^*Fbi + 0,4Fp \qquad (3.2)$$

4. Проаналізувати результати розрахунків моделей та зробити висновки щодо впливу природи на біматричну гру.

Результати досліджень.

За алгоритмом розв’язку (див. 2.3) складено програму на ЕОМ (додаток Б, В, Г), та одержані наступні результати (рис. 3. 3 та табл. 3.3, строки виділені жовтим кольором)

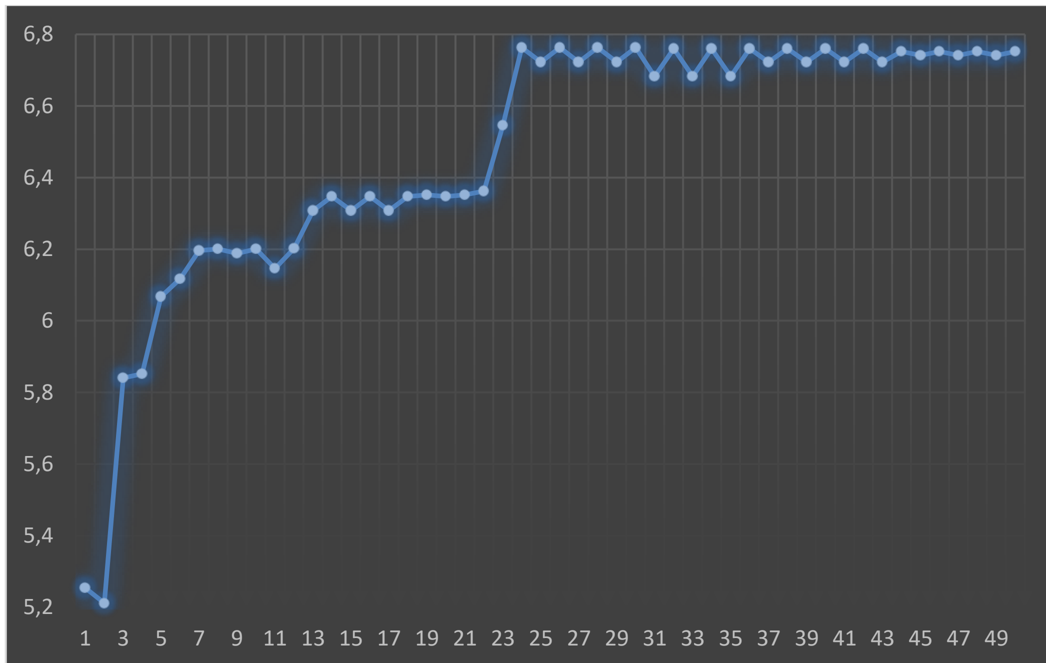


Рисунок 3. 2 – Результати зміни функції пристосованості від кількості поколінь для біматричної гри, без урахування впливу «природи»

Таблиця 3.3 Результати вибору стратегій гравців А та В в моделі 2.

Стратегія проти природи	0,446653	0,234722	0,318624
Гравець А	0,008	0,985	0,007
Гравець В	0,9	0	0,1
Природа * 0,4	0,178661	0,093889	0,12745
Гравець А * 0,6	0,0048	0,591	0,0042
Гравець А з урахуванням природи	0,183461	0,684889	0,13165
Природа * 0,4	0,178661	0,093889	0,12745
Гравець В * 0,6	0,54	0	0,06
Гравець В з урахуванням природи	0,718661	0,093889	0,18745

На рисунку 3. 3 приведений графік функції пристосованості в залежності від кількості поколінь, що відбулися в генетичному алгоритмі, коли гравці грають тільки біматричну гру. Виділяється той факт, що функція функція має тільки одно плато, якого генетичний алгоритм досяг вже в 25 поколінні.

Результати застосувань стратегій близькі до точки рівноваги, що може відбутися в чистих стратегіях, що теж опосередковано вказує на достовірність комп'ютерного випробування.

На рисунку 3.4 приведений графік функції пристосованості в залежності від кількості поколінь, що відбулися в генетичному алгоритмі. Цікавим є те, що функція має три виражених плато, причому першого (f=7,2) вона досягла приблизно 9-10 поколінні, біля другого плато (f=7,3) у функції відбулися коливання з 19 по 35 покоління, а третього (f=7,5) досягнуто в сороковому поколінні.

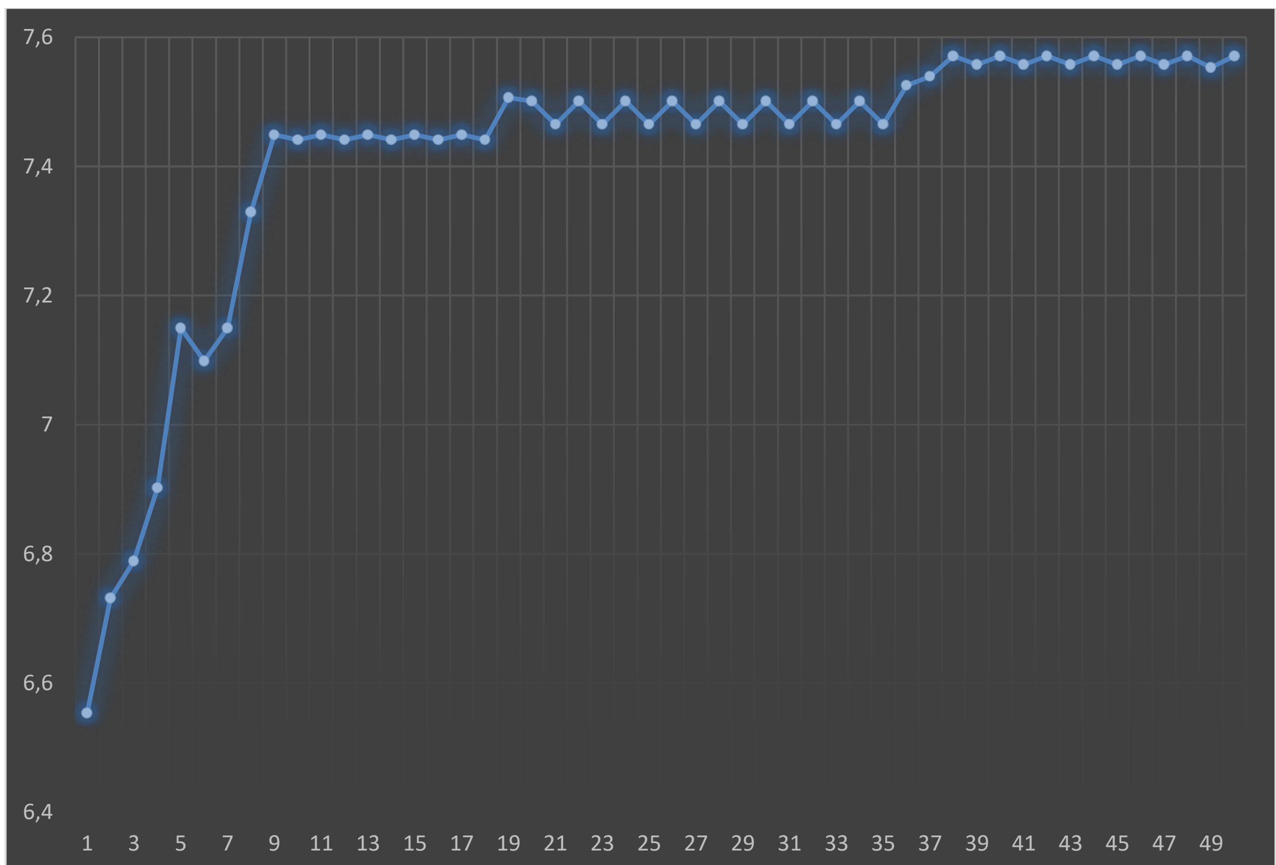


Рисунок 3. 3 – Результати зміни функції пристосованості від кількості поколінь для гри, коли всі три сторони грають одночасно

Таблиця 3.4 Результати вибору стратегій гравців А та В в моделі 3.

Стратегія проти природи	0,446653	0,234722	0,318624
Гравець А	0,008	0,985	0,007
Гравець В	0,9	0	0,1
Гра трьох гравців			
Природа	0,178661	0,093889	0,12745
Гравець А	0,088	0,911	0,001
Гравець В	0,923	0,01	0,067

Вплив природи в біматричну гру відбувається як в моделі 2, так і в моделі 3. З'єднаємо розв'язки по вибору оптимальних стратегій гравців А і В за всіма трьома моделями в одній таблиці див. табл. 3.5)

Таблиця 3. 5 Узагальнена таблиця розв'язків

Тип моделі для розрахунків	Набори змішаних стратегій гравця А	Набори змішаних стратегій гравця В
Модель № 1	0,008; 0,985; 0, 007	0,9; 0; 0,1
Модель № 2	0,183; 0, 685; 0,132	0,719; 0,094; 0,187
Модель № 3	0,088; 0,911; 0,001	0,923; 0,01; 0,067

ВИСНОВКИ

В магістерській роботі проведені дослідження по розробці інформаційної технології для рішення біматричних ігор з урахуванням дії природи, як третього гравця. Результати досліджень:

1. Адаптовано генетичний алгоритм для рішення поставленого завдання.
2. Проведено порівняльний аналіз розв’язання біматричних ігор без урахування впливу природи та з урахування впливу природи на біматричну гру.
3. Створено комп’ютерну реалізацію за тестовим прикладом та отримані результати роботи.

Цікавим є той факт, що вплив природи є суттєвим, та змінює оптимальні стратегії гравців в конфліктних ситуаціях.

СПИСОК ЛІТЕРАТУРИ

1. Bonanno G. Game theory. 2nd Edition. – CreateSpace Independent Publishing Platform, 2018. – 592 p.
2. Peters H. Game Theory. A Multi-Leveled Approach. Second Edition. – Springer-Verlag Berlin Heidelberg, 2015. – 494 p.
3. Диксит А. Стратегические игры. Доступный учебник по теории игр / А. Диксит, С. Скит, Д. Рейли-младший. – Москва: Манн, Иванов и Фербер (МИФ), 2017. – 880 с.
4. Dr. Sea-shon Chen The Influence of Nature on Outcomes of Three Players Game [Електронний ресурс] – Режим доступу до ресурсу: http://www.jgbm.org/page/2%20%20Sea-shon%20Chen.pdf
5. Бойко Н. І., Михайлишин В. Ю. Ефективність застосування генетичних алгоритмів для пошуку оптимізованих рішень// Інформаційні системи та мережі, 2016, №854. – с. 249-257.
6. Вирсански Э. Генетические алгоритмы на Python / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2020. – 286 с.
7. Сивоконь В. В., Шаповалов С. П. Адаптація чисельних методів до розв'язання задач теорії ігор// Матеріали та програма науково-технічної конференції «Інформатика, математика, автоматика (ІМА -2020)». – Суми, СумДУ, 2020. – с. 53.
8. Game theory// Stanford School of Engineering [Електронний ресурс] – Режим доступу до ресурсу: https://online.stanford.edu/courses/soe-ycs0002-game-theory.
9. Fischer M On the connection between symmetric NN-player games and mean field games // Ann. Appl. Probab. 27 (2) 757 - 810, April 2017 [Електронний ресурс] – Режим доступу до ресурсу: https://doi.org/10.1214/16-AAP1215.
10. Min Fan, Ping Zou, Shao-Rong Li, Chin-Chia Wu A Fast Approach to Bimatrix Games with Intuitionistic Fuzzy Payoffs// Te Scientific World Journal, Volume 2014, Article ID 121245, 6 p.

ДОДАТОК А

```
import java.util.Arrays;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        double[][] valuesA = new double[][]{
                {2, 1, 4},
                {3, 1, 2},
                {1, 0, 3}
        };

        double[][] valuesB = new double[][]{
                {1, 1, 2},
                {4, 2, 3},
                {3, 2, 1}
        };

        double[][] valuesNature = new double[][]{
                {8, 4, 2},
                {2, 8, 4},
                {1, 2, 8}
        };

        GeneticAlgorithmBimatrixGame geneticAlgorithmBimatrixGame = new GeneticAlgorithmBimatrixGame(valuesA, valuesB);
        List<double[]> bimatrixStrategy = geneticAlgorithmBimatrixGame.launch();

        GeneticAlgorithmZeroSumGame geneticAlgorithmZeroSumGame = new GeneticAlgorithmZeroSumGame(valuesNature);
```

```
        double[] strategyAgainstNature = geneticAlgorithmZeroSumGame.launch();


        double[] strategyA = new double[3];
        double[] strategyB = new double[3];
        System.out.println("strategyAgainstNature: " +
Arrays.toString(strategyAgainstNature));
        System.out.println("Player A: " + Arrays.toString(bimatrixStrategy.get(0)));
        System.out.println("Player B: " + Arrays.toString(bimatrixStrategy.get(1)));
        for (int i = 0; i < 3; i++) {
            strategyA[i] = 0.6 * bimatrixStrategy.get(0)[i] + 0.4 *
strategyAgainstNature[i];
            strategyB[i] = 0.6 * bimatrixStrategy.get(1)[i] + 0.4 *
strategyAgainstNature[i];
        }


        System.out.println("Strategy of player A: " + Arrays.toString(strategyA));
        System.out.println("Strategy of player B: " + Arrays.toString(strategyB));


    }
}
```

ДОДАТОК Б

```
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.*;

public class GeneticAlgorithmBimatrixGame {
    /* Values of player A */
    private double[][] valuesA;

    /* Values of player B */
    private double[][] valuesB;

    public GeneticAlgorithmBimatrixGame(double[][] valuesA, double[][] valuesB) {
        this.valuesA = valuesA;
        this.valuesB = valuesB;
    }

    public List<double[]> launch() {
        /* Population of 100 chromosomes */
        List<Chromosome> chromosomes = createChromosomes(100);
        /* 50 generations */
        System.out.println("Bimatrix game:");
        double[] Pi = new double[0];
        double[] Ki = new double[0];
        for (int i = 0; i < 50; i++) {
            chromosomes.sort(Collections.reverseOrder());
            System.out.println("Fitness value: " + chromosomes.get(0).getFitness()
                    + ", P values: " + Arrays.toString(chromosomes.get(0).getGenesP())
                    + ", K values: " + Arrays.toString(chromosomes.get(0).getGenesK()));
```

```
            Pi = Arrays.copyOf(chromosomes.get(0).getGenesP(),
chromosomes.get(0).getGenesP().length);
            Ki = Arrays.copyOf(chromosomes.get(0).getGenesK(),
chromosomes.get(0).getGenesK().length);
            crossoverPopulation(chromosomes);
            mutatePopulation(chromosomes);
        }
        return Arrays.asList(Pi, Ki);
    }

    /* Creates population of chromosomes */
    private List<Chromosome> createChromosomes(int number) {
        List<Chromosome> chromosomes = new LinkedList<>();
        for (int i = 0; i < number; i++) {
            chromosomes.add(createChromosome());
        }
        return chromosomes;
    }

    /* Creates chromosome */
    private Chromosome createChromosome() {
        double[] genesP = createGenes();
        double[] genesK = createGenes();
        double fitness = calculateFitness(genesP, genesK);
        return new Chromosome(genesP, genesK, fitness);
    }

    /* Creates genes */
    private double[] createGenes() {
        double[] randomNumbers = new double[3];
```

```
        double sum = 0;
        Random r = new Random();
        for (int i = 0; i < 3; i++) {
            double randomNumber = r.nextDouble();
            randomNumbers[i] = randomNumber;
            sum += randomNumber;
        }
        /* Each generated number is divided by sum of these numbers to get numbers
whose sum is 1 */
        return new double[]{randomNumbers[0] / sum, randomNumbers[1] / sum,
randomNumbers[2] / sum};
    }



    /* Calculates fitness value for genes */
    private double calculateFitness(double[] genesP, double[] genesK) {
        double fitness = 0;
        for (int i = 0; i < genesP.length; i++) {
            for (int j = 0; j < genesK.length; j++) {
                fitness += this.valuesA[i][j] * genesP[i] * genesK[j];
                fitness += this.valuesB[i][j] * genesP[i] * genesK[j];
            }
        }

        return fitness;
    }

    /* Crossover of population */
    private void crossoverPopulation(List<Chromosome> chromosomes) {
        /* Take top 60 chromosomes for crossover */
```

```
        for (int i = 0; i < 60; i += 2) {
            crossoverChromosomes(chromosomes.get(i), chromosomes.get(i + 1));
        }
    }

    /* Crossover of chromosomes */
    private void crossoverChromosomes(Chromosome firstChromosome,
Chromosome secondChromosome) {
        List<double[]> crossoverGenesP =
crossoverGenes(firstChromosome.getGenesP(), secondChromosome.getGenesP());
        if (crossoverGenesP.size() != 0) {
            firstChromosome.setGenesP(crossoverGenesP.get(0));
            secondChromosome.setGenesP(crossoverGenesP.get(1));
        }

        List<double[]> crossoverGenesK =
crossoverGenes(firstChromosome.getGenesK(), secondChromosome.getGenesK());
        if (crossoverGenesK.size() != 0) {
            firstChromosome.setGenesK(crossoverGenesK.get(0));
            secondChromosome.setGenesK(crossoverGenesK.get(1));
        }
        if (crossoverGenesP.size() != 0 || crossoverGenesK.size() != 0) {
            firstChromosome.setFitness(calculateFitness(firstChromosome.getGenesP(),
firstChromosome.getGenesK()));

secondChromosome.setFitness(calculateFitness(secondChromosome.getGenesP(),
secondChromosome.getGenesK()));
        }
    }

    /* Crossover of genes */
```

```
    private List<double[]> crossoverGenes(double[] firstGenes, double[]
secondGenes) {
        /* i - gen to swap between gene sets */
        for (int i = 0; i < firstGenes.length; i++) {
            /* j - gene to leave as is */
            for (int j = 0; j < secondGenes.length; j++) {
                if (i != j) {
                    /* Is crossover possible */
                    if (firstGenes[i] + secondGenes[j] < 1 && firstGenes[j] +
secondGenes[i] < 1) {
                        /* Swap of gene i */
                        double temp = firstGenes[i];
                        firstGenes[i] = secondGenes[i];
                        secondGenes[i] = temp;

                        /* Index of last gene */
                        int thirdIndex = 3 - i - j;
                        /* Calculate value of last gene */
                        firstGenes[thirdIndex] = round(1 - firstGenes[i] - firstGenes[j], 3);
                        secondGenes[thirdIndex] = round(1 - secondGenes[i] -
secondGenes[j], 3);

                        return Arrays.asList(firstGenes, secondGenes);
                    }
                }
            }
        }

        /* Return empty value if crossover is impossible */
        return Collections.emptyList();
```

```
    }

    /* Mutation of population */
    private void mutatePopulation(List<Chromosome> chromosomes) {
        /* Take bottom 40 chromosomes for mutation */
        for (int i = 60; i < chromosomes.size(); i++) {
            mutateChromosome(chromosomes.get(i));
        }
    }

    /* Mutation of chromosome */
    private void mutateChromosome(Chromosome chromosome) {
        chromosome.setGenesP(mutateGenes(chromosome.getGenesP()));
        chromosome.setGenesK(mutateGenes(chromosome.getGenesK()));
        chromosome.setFitness(calculateFitness(chromosome.getGenesP(),
chromosome.getGenesK()));
    }

    /* Mutation of genes */
    private double[] mutateGenes(double[] genes) {
        List<Integer> indexes = new LinkedList<>(Arrays.asList(0, 1, 2));
        /* Gene to mutate */
        int geneToMutate = indexes.get(getRandomNumber(0, indexes.size()));
        indexes.remove(new Integer(geneToMutate));
        /* Gene to leave as is */
        int geneToLeaveAsIs = indexes.get(getRandomNumber(0, indexes.size()));
        double newGeneValue;
        /* Generate gene new value */
        while (true) {
```

```
            Random random = new Random();
            newGeneValue = round(random.nextDouble(), 3);
            if (newGeneValue + genes[geneToLeaveAsIs] < 1) {
                break;
            }
        }
        /* Index of last gene */
        int thirdIndex = 3 - geneToMutate - geneToLeaveAsIs;
        genes[geneToMutate] = newGeneValue;
        /* Calculate value of last gene */
        genes[thirdIndex] = round(1 - newGeneValue - genes[geneToLeaveAsIs], 3);
        return genes;
    }

    private double round(double value, int places) {
        if (places < 0) {
            throw new IllegalArgumentException();
        }

        BigDecimal bd = new BigDecimal(Double.toString(value));
        bd = bd.setScale(places, RoundingMode.HALF_UP);
        return bd.doubleValue();
    }

    private int getRandomNumber(int min, int max) {
        Random random = new Random();
        return random.nextInt(max - min) + min;
    }
}
```

ДОДАТОК В

```
import java.util.*;



public class GeneticAlgorithmZeroSumGame {
    /* Values of nature */
    private double[][] natureValues;

    public GeneticAlgorithmZeroSumGame(double[][] natureValues) {
        this.natureValues = natureValues;
    }

    public double[] launch() {
        /* Population of 100 chromosomes */
        List<Chromosome> chromosomes = createChromosomes(100);
        /* 30 generations */
        System.out.println("Zero-sum game:");
        double[] Xi = new double[0];
        double v = 0;
        for (int i = 0; i < 30; i++) {
            Collections.sort(chromosomes);
            System.out.println("Fitness value: " + chromosomes.get(0).getFitness()
                    + ", P values: " + Arrays.toString(chromosomes.get(0).getGenesP()));
            Xi = Arrays.copyOf(chromosomes.get(0).getGenesP(),
chromosomes.get(0).getGenesP().length);
            v = 1 / chromosomes.get(0).getFitness();
            crossoverPopulation(chromosomes);
            mutatePopulation(chromosomes);
        }
```

```
    /* Calculate strategy */
    for (int i = 0; i < 3; i++) {
        Xi[i] = v * Xi[i];
    }

    return Xi;
}

/* Creates population of chromosomes */
private List<Chromosome> createChromosomes(int number) {
    List<Chromosome> chromosomes = new LinkedList<>();
    for (int i = 0; i < number; i++) {
        chromosomes.add(createChromosome());
    }

    return chromosomes;
}

/* Creates chromosome */
private Chromosome createChromosome() {
    double[] genesP = createGenes();
    double fitness = calculateFitness(genesP);

    return new Chromosome(genesP, fitness);
}

/* Creates genes */
private double[] createGenes() {
```

```
    double[] genes = new double[3];
    Random r = new Random();
    do {
        for (int i = 0; i < 3; i++) {
            genes[i] = r.nextDouble();
        }
    } while (!validateGeneratedGenes(genes));

    return genes;
}

/* Validates gene for compliance with the condition a*x1 + b*x2 + c*x3 >= 1 */
private boolean validateGeneratedGenes(double[] genes) {
    for (int column = 0; column < this.natureValues[0].length; column++) {
        double sum = 0;
        for (int row = 0; row < this.natureValues.length; row++) {
            sum += this.natureValues[row][column] * genes[row];
        }
        if (sum < 1) {
            return false;
        }
    }

    return true;
}

/* Calculates fitness value for chromosome */
private double calculateFitness(Chromosome chromosome) {
    return calculateFitness(chromosome.getGenesP());
```

```
    }

    /* Calculates fitness value for genes */
    private double calculateFitness(double[] genesP) {
        return Arrays.stream(genesP).sum();
    }

    /* Crossover of population */
    private void crossoverPopulation(List<Chromosome> chromosomes) {
        /* Take top 60 chromosomes for crossover */
        for (int i = 0; i < 60; i += 2) {
            crossoverChromosomes(chromosomes.get(i), chromosomes.get(i + 1));
        }
    }

    /* Crossover of chromosomes */
    private void crossoverChromosomes(Chromosome firstChromosome,
Chromosome secondChromosome) {
        List<double[]> crossoverGenesP =
crossoverGenes(firstChromosome.getGenesP(), secondChromosome.getGenesP());
        if (crossoverGenesP.size() != 0) {
            firstChromosome.setGenesP(crossoverGenesP.get(0));
            secondChromosome.setGenesP(crossoverGenesP.get(1));
            firstChromosome.setFitness(calculateFitness(firstChromosome));
            secondChromosome.setFitness(calculateFitness(secondChromosome));
        }
    }

    /* Crossover of genes */
```

```
   private List<double[]> crossoverGenes(double[] firstGenes, double[]
secondGenes) {
      for (int i = 0; i < firstGenes.length; i++) {
         /* i - gen to swap between gene sets */
         double[] firstGenesRes = Arrays.copyOf(firstGenes, firstGenes.length);
         double[] secondGenesRes = Arrays.copyOf(secondGenes,
secondGenes.length);
         double temp = firstGenesRes[i];
         firstGenesRes[i] = secondGenesRes[i];
         secondGenesRes[i] = temp;
         /* Validate new genes */
         if (validateGeneratedGenes(firstGenesRes) &&
validateGeneratedGenes(secondGenesRes)) {
            firstGenes = Arrays.copyOf(firstGenesRes, firstGenesRes.length);
            secondGenes = Arrays.copyOf(secondGenesRes, secondGenesRes.length);
            /* If genes are valid then return them */
            return Arrays.asList(firstGenes, secondGenes);
         }
      }

      /* Return empty value if crossover is impossible */
      return Collections.emptyList();
   }

   /* Mutation of population */
   private void mutatePopulation(List<Chromosome> chromosomes) {
      /* Take bottom 40 chromosomes for mutation */
      for (int i = 60; i < chromosomes.size(); i++) {
         mutateChromosome(chromosomes.get(i));
      }
```

```
    }

    /* Mutation of chromosome */
    private void mutateChromosome(Chromosome chromosome) {
        chromosome.setGenesP(mutateGenes(chromosome.getGenesP()));
        chromosome.setFitness(calculateFitness(chromosome));
    }

    /* Mutation of genes */
    private double[] mutateGenes(double[] genes) {
        double newGeneValue;
        Random random = new Random();
        double[] genesRes;
        /* Generate value until new gene is valid */
        do {
            genesRes = Arrays.copyOf(genes, genes.length);
            int geneToMutate = getRandomNumber(0, 3);
            newGeneValue = random.nextDouble();
            genesRes[geneToMutate] = newGeneValue;
        } while (!validateGeneratedGenes(genesRes));

        return genesRes;
    }

    private int getRandomNumber(int min, int max) {
        Random random = new Random();
        return random.nextInt(max - min) + min;
    }
}
```

ДОДАТОК Г

```
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.*;

public class GeneticAlgorithmTwoPlayersAndNatureGame {
    /* Values of player A */
    private double[][] valuesA;

    /* Values of player B */
    private double[][] valuesB;

    /* Values of nature */
    private double[][] valuesNature;

    public GeneticAlgorithmTwoPlayersAndNatureGame(double[][] valuesA,
double[][] valuesB, double[][] valuesNature) {
        this.valuesA = valuesA;
        this.valuesB = valuesB;
        this.valuesNature = valuesNature;
    }

    public List<double[]> launch() {
        /* Population of 100 chromosomes */
        List<Chromosome> chromosomes = createChromosomes(100);
        /* 50 generations */
        System.out.println("Two players VS Nature:");
        double[] Pi = new double[0];
        double[] Ki = new double[0];
```

```
        for (int i = 0; i < 50; i++) {
            chromosomes.sort(Collections.reverseOrder());
            System.out.println("Fitness value: " + chromosomes.get(0).getFitness()
                    + ", P values: " + Arrays.toString(chromosomes.get(0).getGenesP())
                    + ", K values: " + Arrays.toString(chromosomes.get(0).getGenesK()));
            Pi = Arrays.copyOf(chromosomes.get(0).getGenesP(),
chromosomes.get(0).getGenesP().length);
            Ki = Arrays.copyOf(chromosomes.get(0).getGenesK(),
chromosomes.get(0).getGenesK().length);
            crossoverPopulation(chromosomes);
            mutatePopulation(chromosomes);
        }

        return Arrays.asList(Pi, Ki);
    }

    /* Creates population of chromosomes */
    private List<Chromosome> createChromosomes(int number) {
        List<Chromosome> chromosomes = new LinkedList<>();
        for (int i = 0; i < number; i++) {
            chromosomes.add(createChromosome());
        }

        return chromosomes;
    }

    /* Creates chromosome */
    private Chromosome createChromosome() {
        double[] genesP = createGenes();
        double[] genesK = createGenes();
```

```
        double fitness = calculateFitness(genesP, genesK);

        return new Chromosome(genesP, genesK, fitness);
    }

    /* Creates genes */
    private double[] createGenes() {
        double[] randomNumbers = new double[3];
        double sum = 0;
        Random r = new Random();
        for (int i = 0; i < 3; i++) {
            double randomNumber = r.nextDouble();
            randomNumbers[i] = randomNumber;
            sum += randomNumber;
        }
        /* Each generated number is divided by sum of these numbers to get numbers
whose sum is 1 */
        return new double[]{randomNumbers[0] / sum, randomNumbers[1] / sum,
randomNumbers[2] / sum};
    }

    /* Calculates fitness value for genes */
    private double calculateFitness(double[] genesP, double[] genesK) {
        double playersSum = 0;
        double natureSum = 0;
        for (int i = 0; i < genesP.length; i++) {
            for (int j = 0; j < genesK.length; j++) {
                playersSum += this.valuesA[i][j] * genesP[i] * genesK[j];
                playersSum += this.valuesB[i][j] * genesP[i] * genesK[j];
```

```
            natureSum += this.valuesNature[i][j] * genesP[i] * 0.33;
            natureSum += this.valuesNature[i][j] * genesK[i] * 0.33;
         }
      }

      return 0.6 * playersSum + 0.4 * natureSum;
   }

   /* Crossover of population */
   private void crossoverPopulation(List<Chromosome> chromosomes) {
      /* Take top 60 chromosomes for crossover */
      for (int i = 0; i < 60; i += 2) {
         crossoverChromosomes(chromosomes.get(i), chromosomes.get(i + 1));
      }
   }

   /* Crossover of chromosomes */
   private void crossoverChromosomes(Chromosome firstChromosome,
Chromosome secondChromosome) {
      List<double[]> crossoverGenesP =
crossoverGenes(firstChromosome.getGenesP(), secondChromosome.getGenesP());
      if (crossoverGenesP.size() != 0) {
         firstChromosome.setGenesP(crossoverGenesP.get(0));
         secondChromosome.setGenesP(crossoverGenesP.get(1));
      }

      List<double[]> crossoverGenesK =
crossoverGenes(firstChromosome.getGenesK(), secondChromosome.getGenesK());
      if (crossoverGenesK.size() != 0) {
         firstChromosome.setGenesK(crossoverGenesK.get(0));
```

```
            secondChromosome.setGenesK(crossoverGenesK.get(1));
        }

        if (crossoverGenesP.size() != 0 || crossoverGenesK.size() != 0) {
            firstChromosome.setFitness(calculateFitness(firstChromosome.getGenesP(),
firstChromosome.getGenesK()));

secondChromosome.setFitness(calculateFitness(secondChromosome.getGenesP(),
secondChromosome.getGenesK()));
        }
    }

    /* Crossover of genes */
    private List<double[]> crossoverGenes(double[] firstGenes, double[]
secondGenes) {
        /* i - gen to swap between gene sets */
        for (int i = 0; i < firstGenes.length; i++) {
            /* j - gene to leave as is */
            for (int j = 0; j < secondGenes.length; j++) {
                if (i != j) {
                    /* Is crossover possible */
                    if (firstGenes[i] + secondGenes[j] < 1 && firstGenes[j] +
secondGenes[i] < 1) {
                        /* Swap of gene i */
                        double temp = firstGenes[i];
                        firstGenes[i] = secondGenes[i];
                        secondGenes[i] = temp;

                        /* Index of last gene */
                        int thirdIndex = 3 - i - j;
```

```
                    /* Calculate value of last gene */
                    firstGenes[thirdIndex] = round(1 - firstGenes[i] - firstGenes[j], 3);
                    secondGenes[thirdIndex] = round(1 - secondGenes[i] -
secondGenes[j], 3);

                    return Arrays.asList(firstGenes, secondGenes);
                }
            }
        }
    }

    /* Return empty value if crossover is impossible */
    return Collections.emptyList();
  }

  /* Mutation of population */
  private void mutatePopulation(List<Chromosome> chromosomes) {
    /* Take bottom 40 chromosomes for mutation */
    for (int i = 60; i < chromosomes.size(); i++) {
      mutateChromosome(chromosomes.get(i));
    }
  }

  /* Mutation of chromosome */
  private void mutateChromosome(Chromosome chromosome) {
    chromosome.setGenesP(mutateGenes(chromosome.getGenesP()));
    chromosome.setGenesK(mutateGenes(chromosome.getGenesK()));
    chromosome.setFitness(calculateFitness(chromosome.getGenesP(),
chromosome.getGenesK()));
  }
```

```
/* Mutation of genes */
private double[] mutateGenes(double[] genes) {
   List<Integer> indexes = new LinkedList<>(Arrays.asList(0, 1, 2));
   /* Gene to mutate */
   int geneToMutate = indexes.get(getRandomNumber(0, indexes.size()));
   indexes.remove(new Integer(geneToMutate));
   /* Gene to leave as is */
   int geneToLeaveAsIs = indexes.get(getRandomNumber(0, indexes.size()));
   double newGeneValue;
   /* Generate gene new value */
   while (true) {
      Random random = new Random();
      newGeneValue = round(random.nextDouble(), 3);
      if (newGeneValue + genes[geneToLeaveAsIs] < 1) {
         break;
      }
   }
   /* Index of last gene */
   int thirdIndex = 3 - geneToMutate - geneToLeaveAsIs;
   genes[geneToMutate] = newGeneValue;
   /* Calculate value of last gene */
   genes[thirdIndex] = round(1 - newGeneValue - genes[geneToLeaveAsIs], 3);

   return genes;
}

private double round(double value, int places) {
   if (places < 0) {
```

```
            throw new IllegalArgumentException();
        }

        BigDecimal bd = new BigDecimal(Double.toString(value));
        bd = bd.setScale(places, RoundingMode.HALF_UP);
        return bd.doubleValue();
    }

    private int getRandomNumber(int min, int max) {
        Random random = new Random();
        return random.nextInt(max - min) + min;
    }
}
```