МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ Аліна САВЧЕНКО
«___»_____2023 р.

# КВАЛІФІКАЦІЙНА РОБОТА
**(ПОЯСНЮВАЛЬНА ЗАПИСКА)**

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ МАГІСТР
ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ
«ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ПРОЕКТУВАННЯ»

**Тема: «Ботоферма для моделювання стратегій в психологічно-соціальній грі "Ультиматум"»**

Виконавець: Назарій СТІНЯНСЬКИЙ

Керівник: к.т.н. Олег ЗУДОВ

Нормоконтролер: к.т.н., доцент Олена ТОЛСТІКОВА

КИЇВ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних наук та технологій*
Кафедра *комп'ютерних інформаційних технологій*
Спеціальність *122 «Комп'ютерні науки»*
Освітньо-професійна програма *«Інформаційні технології проектування»*

ЗАТВЕРДЖУЮ:
завідувач кафедри КІТ
_____Аліна САВЧЕНКО
(підпис)
«____»_____2023 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи
Стінянського Назарія Юрійовича
(ПІБ випускника )

1. Тема роботи: «Ботоферма для моделювання стратегій в психологічно-соціальній грі "Ультиматум"» затверджена наказом ректора № 1976/ст від 29.09.2023р.
2. Термін виконання роботи: з 02 жовтня 2023 року по 31 грудня 2023 року.
3. Вихідні дані до роботи: застосунок на мові програмування Java для проведення досліджень гри "Ультиматум".
4. Зміст пояснювальної записки: 1. Аналіз гри "Ультиматум". 2. Функціонал та вибір інструментів для розробки. 3. Проектування та розробка застосунку. 4. Тестування ботоферми
5. Перелік обов'язкового ілюстративного матеріалу: 1. Ілюстрації до класів ботів 2. Ілюстрації до класів гри 3. Ілюстрації до класів RoundHandler. 4. Ілюстрації до тестів кожного бота окремо 5. Ілюстрації до інших тестів _____

6. Календарний план-графік

| № з/п | Завдання | Термін виконання | Підпис керівника |
|---|---|---|---|
| 1. | Аналіз предметної області та огляд аналогів. Написання 1 розділу, аналіз та поняття технології | 02.10.2023-16.10.2023 | |
| 2. | Вибір та опис використаних технологій. Написання 2 розділу, проектування застосунку | 17.10.2023-30.10.2023 | |
| 3. | Написання 3 та 4 розділів, розробка та тестування застосунку | 31.10.2023-14.11.2023 | |
| 4. | Загальне редагування та друк пояснювальної записки | 15.11.2023-20.11.2023 | |
| 5. | Проходження нормоконтролю, перепліт пояснювальної записки. | 16.11.2023-20.11.2023 | |
| 6. | Розробка тексту доповіді. Оформлення графічного матеріалу для презентації | 19.12.2023-22.12.2023 | |

7. Дата видачі завдання 02.10.2023 р.

Керівник кваліфікаційної роботи _____ Олег ЗУДОВ
(підпис керівника)

Завдання прийняв до виконання _____ Назар СТІНЯНСЬКИЙ
(підпис випускника)

# РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи на тему: «Ботоферма для моделювання стратегій в психологічно-соціальній грі “Ультиматум”» містить: 101 сторінку, 41 рисунок, 20 інформаційних джерел.

**Об'єкт дослідження** – розподілені системи штучного інтелекту

**Предмет дослідження** – симуляція психологічно-поведінкової гри на ботофермі

**Мета кваліфікаційної роботи** – розробити готову програму – ботоферму, для моделювання стратеній у грі “Ультиматум” і дослідити поведінкові моделі.

**Методи дослідження –** мова програмування Java, інтегроване середовище розробки IntellijIDEA**.**

Результати кваліфікаційної роботи рекомендується використовувати дослідникам, які цікавляться поведінковою психологією.

ЗАСТОСУНОК, БОТОФЕРМА, БОТ, UI, JAVA, МОДЕЛЬ, SWING, DESKTOP

**ЗМІСТ**

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

| | | |
|---|---|---|
| *SDK (Software development kit)* | – | Набір для розробки програмного забезпечення |
| *IDE (Integrated development. environment)* | – | Інтегроване середовище розробки |
| *API (Application programming interface)* | – | Програмний інтерфейс |
| *URL (Uniform Resource Locator* | – | Визначник місцезнаходження сайту в мережі Інтернет |

# ВСТУП

У сучасному суспільстві, яке невпинно розвивається та трансформується, вивчення психологічних та соціальних аспектів міжособистісних відносин є надзвичайно актуальним завданням. Однією з ключових складових цього дослідження є аналіз участі особистостей в соціальних іграх, де вони взаємодіють, вирішуючи різноманітні завдання та поставлені перед ними виклики.

Психологічно-соціальні дослідження представляють собою важливий напрямок наукової діяльності, спрямований на вивчення взаємодії між психічними та соціальними аспектами людської поведінки. Ця область науки досліджує, як індивіди та групи сприймають, взаємодіють та впливають один на одного в соціальному оточенні.

Психологічний компонент досліджень зосереджений на вивченні внутрішніх процесів, емоцій, мислення та інших аспектів психічного функціонування, які впливають на взаємодію в соціумі. З іншого боку, соціальний аспект охоплює вивчення структури суспільства, групової динаміки, культурних впливів та інших соціокультурних факторів.

Ці дослідження можуть охоплювати різноманітні теми, такі як міжособистісні відносини, соціальна дискримінація, вплив групової динаміки на прийняття рішень, аналіз соціокультурних варіацій у поведінці тощо. Вони надають можливість глибоко розуміти та пояснювати складні взаємозв'язки між психічними та соціальними процесами, що відкриває нові перспективи для розвитку сучасної психології та соціології.

Однією з таких ігор є "Ультиматум" - експериментальна гра, яка визначається не лише стратегічним мисленням, але й складними соціальними взаємодіями та психологічними реакціями учасників. У рамках цього дослідження розглядається можливість використання ботоферми - автоматизованого інструмента для проведення експерименту, який дозволяє

вивчати різноманітні аспекти соціально-психологічної гри "Ультиматум".

Метою даного дослідження є розкриття та аналіз психологічних та соціальних вимірів, які виникають під час участі в грі "Ультиматум", з використанням ботоферми як інструменту для автоматизації процесу проведення експерименту. Результати цього дослідження можуть внести важливий внесок у розуміння та аналіз взаємодії між особистісними та соціальними чинниками, що впливають на прийняття рішень у сфері міжособистісних відносин.

Дослідження психологічно-соціальної гри "Ультиматум" з використанням ботоферми відкриває шлях для нових можливостей у сфері дослідження поведінки та прийняття рішень, розширюючи наше розуміння людської натури та взаємодії в соціумі.

В ході дослідження буде використана ботоферма - програма, яка дозволить автоматизувати процеси взаємодії та спостереження за учасниками гри "Ультиматум". Це дозволить відтворити різні моделі поведінки гравців (ботів) та дослідити різні стратегії. Які стануть кращими? Які виявляться провальними? Чи залежатиме успіх стратегії від кількость егоїстичних гравців? Тощо.

Це дослідження стане цікавим та корисним досвідом і також допоможе зрозуміти як різні категорії людей можуть впливати один на одного.

# РОЗДІЛ 1
# АНАЛІЗ ГРИ “УЛЬТИМАТУМ”

## 1.1. Ультиматум як слово

Для того, аби цілком зрозуміти суть гри “Ультиматум” та суть дослідження, яке проводиться в цій кваліфікаційній роботі, слід гарно розібратися в самому визначенні слова “Ультиматум”, адже не спроста відомий економіст, та лауреат Нобелівської премії Рауль Кардосо вибрав саме таку назву своїй грі.

З визначення, слово Ультиматум означає рішучу, категоричну вимогу з можливими погрозами. Це слово походить від латинського - ultimatum, від ultimus - останній, крайній.

Можна також сказати, що ультиматум — це вимога або пропозиція, висунута іншій стороні, як правило, більш впливовою організацією або особою, з очікуванням остаточного рішення або дії.

Ультиматум може бути висунутий у багатьох різних ситуаціях, зокрема в особистих стосунках, бізнесі, політиці та міжнародній дипломатії. Наприклад, батьки можуть поставити дитині ультиматум, щоб вона припинила певну поведінку або зазнала покарання. Розуміння поняття ультиматуму може допомогти людям орієнтуватися в таких ситуаціях і знаходити взаємовигідні рішення.

### 1.2. Перемовини

Процес переговорів також є предметом дослідження теорії ігор. Як і в кожній моделі теорії ігор, у ньому є гравці (сторони переговорів), стратегії (їх пропозиції) і виграші (результат переговорів).

Як правило, в процесі переговорів учасники мають дійти згоди, як поділити певну додаткову вартість. Якщо переговори невдалі, то їх виграші менші (або взагалі нульові).

Простим прикладом переговорів є продаж товару. Уявимо собі Єгипет, «музей» каміння, товар без цінників і продавець намагається продати вам вазу. Продавець знає, що ціна вази 20$ — це нижня границя можливого (озвучена ціна звичайно буде вп'ятеро більша). Ви, припустимо, захотіли купити товар, але точно не готові платити більше 60$ (тобто при купівлі за меншу ціну різниця буде Вашим виграшем).

Таким чином під час переговорів потрібно досягти згоди як розділити 40$ можливого прибутку від ухвалення угоди. Ці 40$ — додаткова вартість, яка виникає від кооперації; якщо угода не ухвалюється, кожен з гравців отримує 0.

Подібна властивість характерна для будь-яких переговорів. Інша властивість полягає у тому, що це не гра з нульовою сумою. Якщо додаткова вартість з'являється, гравці намагаються її поділити, і всі їх стратегії спрямовані на отримання більшої частки. Це може здаватися схожим на некооперативні ігри, але всі сторони знають, що якщо угода не буде ухвалена, то ніхто нічого не отримає. Ця взаємно шкідлива альтернатива, якої вони намагаються уникнути і це породжує можливість застосування погроз, обіцянок та зобов'язань, що робить переговори цікавою грою, складною для дослідження.

Розглянемо ще один приклад. У силіконовій долині зустрілося два підприємці: Енді і Білл. Енді виробляє чіп, який він продає за 900$, Білл розробив програмний пакет, який він може продавати за 100$. Після зустрічі

вони розуміють, що після невеликої доробки вони можуть створити єдиний модуль, який буде коштувати 3000$. Тобто разом вони створюють додаткові 2000$, і вони очікують продавати велику кількість модулів кожен рік. Єдина перешкода для початку запуску виробництва — це як ділити отримані з продажу кожного модуля 3000$. Якщо подумати, то це непроста задача.

Наприклад, Білл може почати з того, що без його програми чіп — це звичайна мікросхема без особливих властивостей і тому він має отримати 2100$, а Енді — свої 900$. У відповідь Енді скаже, що без його чіпа програма Білла — це просто алгоритми, які не надто ефективні на звичайному харді, тому він має отримати 2900$, а Білл свої 100$. Сторонній спостерігач може резонно запропонувати поділити додатковий виграш пропорційно, але і тут все непросто.

Білл запропонує ділити навпіл (оскільки один без одного не можуть створити продукт), тобто 1100$ собі і 1900$ — Енді. Енді запропонує ділити прибуток пропорційно початковому вкладу у вартість продукту (1/10 та 9/10), тобто 2700$ йому і 300$ — Біллу.

Очевидних причин взяти за основу той чи інший варіант немає, тому так може продовжуватись досить довго. Кінцеве рішення буде залежати від впертості, терпіння і винахідливості сторін.

### 1.3. Гра “Ультиматум”

Тепер варто дізнатись що ж собою являє сама гра “Ультиматум”. Які її правила, цінність, історія тощо.

Ультиматум - це класична гра поділу, яка використовується в дослідженнях переважно в експериментальній економіці для вивчення неегоїстичних переваг. Вона є найпоширенішим інструментом досліджень в

експериментальній економіці. Це і не дивно, адже правила дужу прості хоча гра може дослідити суть будь-яких перемовин.

У грі беруть участь два гравці: А та В . Гравцю А видається певна сума грошей. Потім йому пропонується поділити цю суму між собою та гравцем В у будь-якій пропорції, після чого гравець В може або прийняти частку, запропоновану гравцем А , і тоді угода відбувається, або відмовитись. У другому випадку обидва гравці позбавляються виграшів і залишаються ні з чим. При цьому вся інформація про гру та її правила відома обом учасникам заздалегідь.

А і дійсно, будь-які перемовини зводяться до пропозиції та її прийняття або ж відхилення. У випадку, якщо угоду було відхилено - ніхто не отримає вигоди. В цьому полягає вся сутність гри “Ультиматум”.

### 1.4. Раціональність та справедливість

У грі, згідно з класичною теорією максимізації вигоди, яку б частку більше за нуль гравець А не пропонував, гравцю В завжди вигідно погоджуватися, адже в іншому випадку виграші обох учасників дорівнюють нулю. Тому гравцеві А вигідніше пропонувати мінімально можливу частку і максимізувати власний виграш, а другому гравцеві прийняти цю частку і отримати вигоду більше за нуль. Дане твердження правильно з причиною у тому, що обидва агента раціональні і максимізують свій виграш.

Проте насправді безліч проведених експериментів показало, перший гравець пропонує гравцю В у середньому частку 30-40 % від початкової виданої суми. Водночас пропозиції часток менше 20 % найчастіше відкидаються гравцем В

При чому експерименти проводились при різних умовах з різними людьми з різних країн, з різним становищем, доходами та іншим. І результати дійсно дещо відрізнялись. Тому окрім раціональності існує й інший не мало важливий фактор - справедливість. Якщо гравець В вважатиме, що угода занадто не справедлива - він її скоріше відхилить не зважаючи на раціональне рішення - прийняти її.

### 1.5. Експерименти

Багато експериментів було проведено з грою «Ультиматум». Враховуючи різні варіанти, дослідники вивчали поведінки людей за невеликих змін стандартної гри.

Наприклад, були вивчені поділи гри у закритих племенах та спільнотах у статті Henrich et al. (2001), і з'ясувалося, що відмінності товариств щодо господарської організації та ступеня інтеграції ринків значно впливають на рішення про поділ. Так, чим вищий ступінь інтеграції ринків і віддача від співпраці (того, як взаємодія з іншими людьми важлива в економічному плані) у племені, тим вище запропонована частка.

В Індонезії у спільноті Ламелара основним видом діяльності є китобійний промисел , для якого необхідна спільна робота принаймні 7-8 осіб, тому середній розмір пропонованої частки становив 58% суми. Тобто необхідність великої кооперації призводить до необхідності спільного розподілу надлишків.

У свою чергу, в племені Мачигуенга сім'ї економічно незалежні і рідко беруть участь у діяльності, яка потребує допомоги ззовні, тому частка розподілу склала 26%, причому відмов за результатами дослідження практично не було, враховуючи, що 75% усіх часток були нижчими за 30%.

Також є підстави вважати, що величина загальної суми і відповідно розподілу в абсолютному вираженні мають значення. Чим вища пропозиція в

абсолютній величині, тим нижча ймовірність відмови, навіть якщо частка щодо початкової суми така сама. Проведення експерименту в Індії з варіантами 20, 200, 2000, 20000 рупій показало значну варіацію в поведінці гравців зі збільшенням ставок: за найбільшої ставки відмову вибрав лише один учасник з 24-х; за найменшої ставки відмов було 36 %.

Один з цікавих експериментів із грою ультиматум провівся у Лабораторії дослідження поведінки в Університеті Гарварда, який допоміг вченим розглядати вплив соціального контексту на прийняття рішень учасниками гри.

Назва Експерименту: "Вплив соціального статусу на стратегії у грі ультиматум."

Мета Експерименту: Вивчення того, як соціальний статус учасників впливає на їхні стратегії та прийняття рішень у грі ультиматум.

Методологія:

Учасники: У експерименті взяли участь 100 студентів різних соціальних груп.

Розділення Груп: Учасників було розділено на дві групи: високий соціальний статус та низький соціальний статус. Це визначалося на основі анкетування та попереднього дослідження про соціальні аспекти їхнього життя.

Гра Ультиматум: Учасники грали в гру ультиматум, де один гравець (пропонент) мав можливість розділити певний ресурс з іншим гравцем (респондентом). Респондент міг прийняти або відхилити пропозицію.

Контрольні Змінні: В експерименті враховувалися різні фактори, такі як інформація про соціальний статус партнера, рівень довіри, чесності та сприйняття справедливості.

Результати:

Вплив Соціального Статусу: Учасники з високим соціальним статусом були більш схильні до вищих пропозицій і мали більш високий рівень впливу на рішення респондентів.

Стратегії в Залежності від Груп: Гравці з високим соціальним статусом виявилися більш стратегічними, розглядаючи не лише свої вигоди, але і те, як їхні пропозиції впливають на рішення респондентів.

Психологічні Аспекти: Студенти з низьким соціальним статусом часто відхиляли низькі пропозиції, навіть якщо це означало, що вони отримають менше, ніж інші.

Висновки:

Цей експеримент підкреслив важливість соціального контексту в грі ультиматум. Учасники не просто максимізували свій виграш, але також враховували соціальний статус та його вплив на рішення партнера. Експеримент надав цінний внесок у розумінні соціальних аспектів взаємодії в умовах обмежених ресурсів та конфлікту інтересів.

### 1.6. Різновиди гри

Окрім стандартних правил гри "Ультиматум", існують різноманітні модифікації та варіації, які дозволяють досліджувати різні аспекти людської поведінки, взаємодії та стратегій. Давайте розглянемо кілька цікавих різновидів гри:

#### 1.6.1. Багатоосібний “Ультиматум”

Багатоосібний Ультиматум розширює стандартну гру "Ультиматум" на випадок, коли бере участь троє або більше гравців. Цей варіант гри дозволяє досліджувати більш складні соціальні динаміки та взаємодії. Існує кілька причин, чому багатоосібний ультиматум може бути цікавим для досліджень:

1 Більш Складні Стратегії:

У багатоосібних сценаріях гравці можуть розвивати більш складні стратегії взаємодії. Поділ грошей може включати більше динаміки, такі як обговорення, голосування чи об'єднання сил для спільного прийняття рішення.

2 Соціальні Взаємодії:

Багатоосібний ультиматум створює більше можливостей для соціальних взаємодій та визначення внутрішніх динамік групи. Гравці можуть брати до уваги не лише свої індивідуальні інтереси, але і інтереси групи.

3 Розвиток Колективних Рішень:

Гра стимулює розробку механізмів прийняття рішень в колективі. Учасники можуть працювати над тим, щоб досягти спільного консенсусу чи прийняти групове рішення.

4 Спостереження за Впливом Групи на Поведінку:

Дослідження багатоосібного ультиматуму може допомогти вивчити, як впливає групова динаміка на поведінку і прийняття рішень. Це може бути корисно для розуміння колективних процесів у реальних ситуаціях.

5 Специфікація Поведінки в Групі:

Багатоосібний ультиматум може виявити особливості взаємодії в групі, такі як лідерство, конфлікти чи співпраця, що може бути корисно для розуміння динаміки колективної поведінки.

Дослідження багатоосібного ультиматуму може мати практичне застосування в області управління, командної роботи, організаційної психології та інших сферах, де важливо розуміти, як групи приймають колективні рішення та вирішують конфлікти.

### 1.6.2. Конкурентний “Ультиматум”

"Конкурентний Ультиматум" представляє собою унікальну модифікацію гри, де пропонується ідея не одного, а багатьох пропонентів. Така структура гри дозволяє створити конкурентну обстановку, де респондент може обирати між пропозиціями від кількох гравців. Давайте розглянемо деталі цієї концепції:

Багато Пропонентів: У цій версії гри існує багато осіб, які виступають у ролі пропонентів. Кожен з них може внести свою пропозицію щодо розподілу ресурсів або ендавменту.

Обмежені Вибори Респондента: Респондент, той, хто приймає чи відхиляє пропозиції, обирає з пропозицій всіх пропонентів. Це дозволяє йому (респондентові) вибирати найбільш вигідні для себе варіанти.

Заохочення до Конкуренції: Умови гри сприяють конкуренції між пропонентами. Кожен з них старається внести пропозицію, яка буде більш вигідною для респондента, з метою отримати його підтримку.

Максимізація Доходу Респондента: Зазвичай, при наявності багатьох пропонентів, респонденту пропонують значну частку від загального фонду, спонукуючи його взяти участь в грі та прийняти одну з пропозицій.

Важкі Вибори Респондента: Респондент стикається з важким вибором, оскільки йому потрібно визначитися, яку пропозицію прийняти. Він може аналізувати пропозиції, оцінювати їхню вигідність і вибирати ту, яка найбільше відповідає його інтересам.

Ця версія гри створює захопливий сценарій, де конкуренція між пропонентами визначає зміст гри, а респондент має можливість максимізувати свій виграш, обираючи оптимальну пропозицію.

### 1.6.3. “Ультиматум” з чайовими

"Ультиматум з чайовими" - це варіація класичної гри в ультиматум, в якій введено елемент чайових. В цій грі гравці мають можливість домовлятися про додаткові вигоди чи винагороди, які будуть включені в угоду разом з основним розподілом ресурсів чи ендавменту.

Основна ідея полягає в тому, що гравець А, який вносить пропозицію, може включити в неї чайові або додаткові вигоди для гравця В. Це може бути, наприклад, додаткова частина ресурсів, особливі привілеї чи інші бонуси. Гравець А виражає своє довір'я гравцеві В, роблячи ставку саме на чайові.

Гравець В, у свою чергу, має вигоду від того, що гравець А довіряє йому, і це може спонукати його до того, щоб прийняти пропозицію. Однак на його совісті залишається виконати умови чайових, тобто здійснити обіцяні додаткові вигоди.

Цей елемент чайових робить гру більш складною і залежною від взаємного довір'я гравців. Вони повинні враховувати не тільки основний розподіл ресурсів, але й можливі наслідки та виконання домовлених чайових. Таким чином, винагорода стає більш справедливою або збалансованою завдяки елементу довіри в цій версії ультиматуму.

### 1.6.4. Зворотній “Ультиматум”

"Зворотний Ультиматум" - це гра, яка передбачає визначення розподілу ресурсів або ендавменту між двома учасниками, проте в цьому випадку є певна специфіка. У цій версії гри пропонент (той, хто вносить пропозицію)

має право пропонувати будь-яку кількість розділів ендавменту. Головна особливість полягає в тому, що гра завершується лише тоді, коли респондент (той, хто відповідає на пропозицію) приймає пропозицію або відмовляється від участі.

Якщо респондент приймає пропозицію, то розділ ендавменту реалізується відповідно до запропонованого варіанту. У випадку відмови респондента, той, хто вносить пропозицію, отримує трохи менше половини початкового ендавменту або фонду, який був визначений на початку гри.

Таким чином, учасники гри мають стратегічно розглядати свої пропозиції та взаємодіяти так, щоб досягти максимально вигідного результату для себе в умовах цього зворотного ультиматуму.

### 1.6.5. “Ультиматум” з неповною інформацією

Гра "Ультиматум з неповною інформацією" відрізняється тим, що один із гравців має додаткову інформацію про розмір ресурсів або "пирога", який потрібно розділити. Це може бути інформація про загальний обсяг ресурсів, можливо, їхню ціну або щось інше, що впливає на цінність ресурсу.

Наприклад, якщо гравець В знає, що ставка в цій грі є надто великою або важкою для гравця А, то він може використовувати цю інформацію для своєї вигоди. Знання того, що ставка велика, може підштовхнути гравця В до прийняття найменш вигідного для гравця А розподілу ресурсів. Гравець В може спрямовувати гру в такий спосіб, щоб забезпечити собі максимальний виграш, враховуючи обставини.

Ця додаткова інформація надає можливість більш виваженим стратегіям та може впливати на рішення гравців у напрямку, який більше вигідний для того, хто має додаткову інформацію. Гра стає більш заплутаною, оскільки

розглядається не лише сам розподіл ресурсів, але й вплив додаткової інформації на рішення гравців.

### 1.6.6. Диктатор

Гра "Диктатор" є однією з форм гри в ультиматум, де гравець А має абсолютну владу над прийняттям рішення щодо розподілу ресурсів, і гравець В позбавлений можливості впливати на рішення.

В даній версії гри гравець В нічого не може зробити, крім як прийняти або відхилити пропозицію, яку робить гравець А. Гравець А може вирішити взяти собі усе (100% ресурсів чи ендавменту), і гравець В нічого не може протистояти цьому рішенню.

Ця версія гри дозволяє досліджувати, наскільки альтруїстичним чи егоїстичним є гравець А. Якщо гравець А обирає розділ, що не враховує інтереси гравця В, то це може свідчити про високий рівень егоїзму. З іншого боку, якщо гравець А вирішує розділити ресурси чи ендавмент більш справедливо, то це може вказувати на більш альтруїстичний підхід.

Ця модель гри дозволяє вивчати психологічні та етичні аспекти прийняття рішень в умовах абсолютної влади одного з учасників.

## 1.7. Середні показники

Середні показники, які випливають із результатів проведених наукових досліджень гри "Ультиматум", відкривають цікаві аспекти щодо стратегій та прийняття рішень учасниками:

Пропозиція 50/50, що передбачає рівний розподіл ресурсів, визнається практично всіма, аж на 100%, гравцями. Це свідчить про широке прийняття ідеї справедливості та рівності серед учасників гри.

Пропозиція 70/30 викликає зацікавленість меншої кількості гравців, і лише близько 40% з них готові прийняти такий розподіл ресурсів. Це вказує на те, що більшість учасників уникатимуть сценаріїв, де вони отримують меншу частку виграшу.

Цікаво, що пропозиції 80/20 та 70/30 мають схожий рівень прийняття серед гравців. Це може вказувати на те, що деякі учасники сприймають ці варіанти як еквівалентні з точки зору вигоди, або є важливим впливом інших факторів.

Значно менший відсоток гравців, лише 8-10%, приймає пропозицію 90/10. Це може свідчити про високий рівень вимогливості та несхвалення учасниками варіантів, де вони отримують менше значущу частку ресурсів.

Отже, аналіз середніх показників зазначених пропозицій в грі "Ультиматум" надає важливі відомості про тенденції та переваги учасників щодо розподілу ресурсів, а також відображає ступінь їхньої чутливості до різних пропозицій з точки зору справедливості та вигоди.

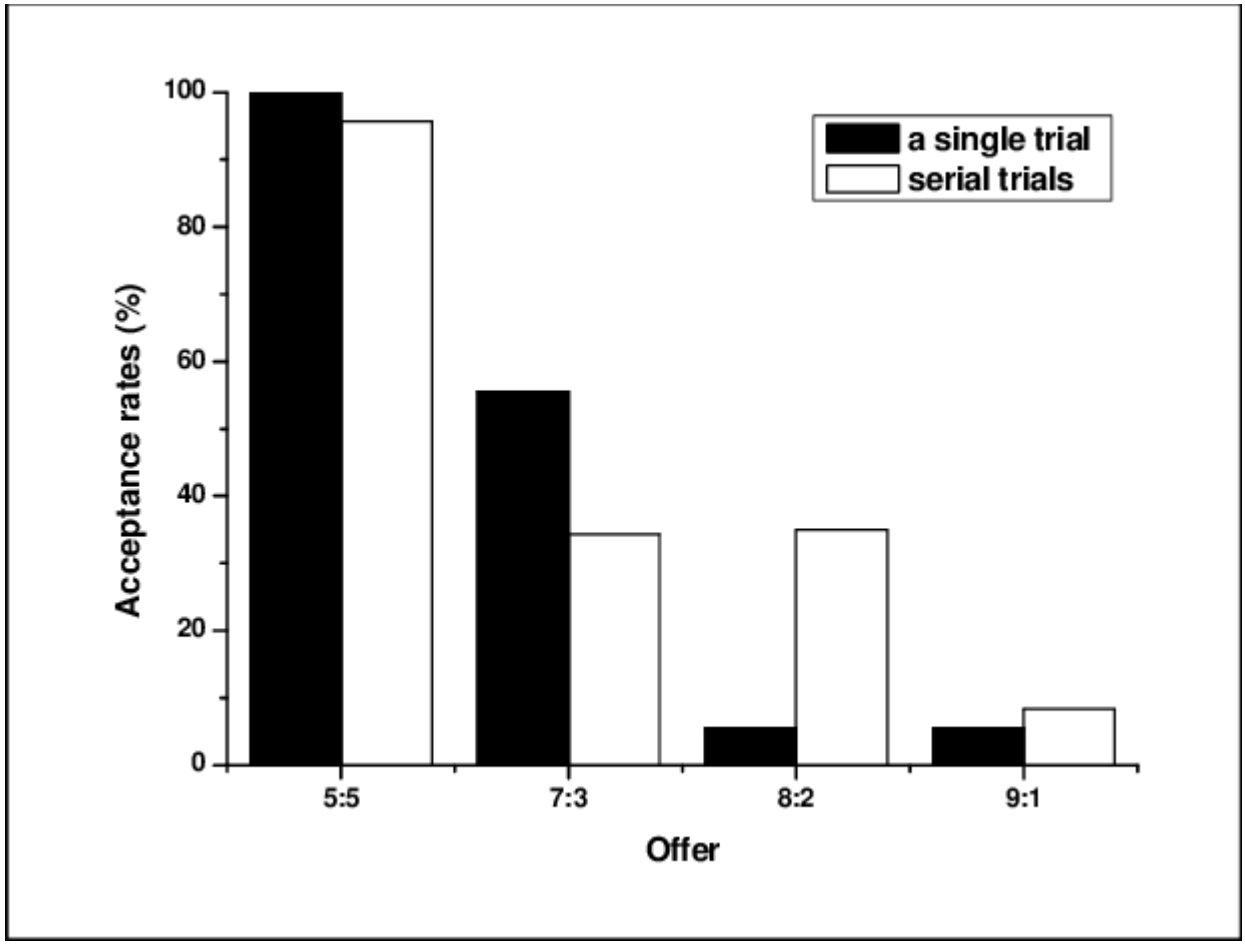



Рис. 1.1. Тестування на грі “Ультиматум”

## ВИСНОВКИ ДО РОЗДІЛУ 1

Гра ультиматум, в усіх своїх різновидностях та варіаціях, виступає як цікавий експериментальний інструмент для вивчення людського прийняття рішень, взаємодії та соціальних відносин. На підставі аналізу різних моделей гри, можна сформулювати кілька обширних висновків:

Стратегії та Раціональність: В ультиматумі виокремлюються стратегії гравців, їхні раціональні та ірраціональні рішення. Гравці намагаються максимізувати свій виграш, але в той же час, емоції, довіра та інші соціальні чинники можуть впливати на їхні рішення.

Довіра та Репутація: В грі ультиматум, особливо в моделях з чайовими чи неповною інформацією, важливою стає роль довіри та репутації. Гравці, які здобувають довіру партнера, можуть досягти більш вигідних угод та розподілу ресурсів.

Справедливість та Егоїзм: Варіанти гри засвідчують важливі аспекти соціальної справедливості та егоїзму. Вибір гравцем розділу може відображати його альтруїстичні або, навпаки, егоїстичні настанови.

Культурні та Індивідуальні Різниці: Результати гри можуть варіюватися в залежності від культурних та індивідуальних особливостей гравців. Врахування цих аспектів дозволяє краще розуміти різноманіття людського поведінку в умовах ультиматуму.

Інформаційна Стратегія: Гра ультиматум із неповною інформацією підкреслює важливість інформаційних стратегій в прийнятті рішень. Гравці, які можуть здобути більше інформації, мають перевагу в угодах.

Динаміка Взаємодії: Гра в ультиматум розглядається як динамічний процес, де взаємодія гравців може еволюціонувати внаслідок різних факторів, таких як досвід, вивчення стратегій опонентів та змінюючіся умови гри.

Загалом, гра в ультиматум слугує не лише інструментом для тестування економічних теорій, але й дозволяє досліджувати глибокі аспекти людської поведінки, раціональності та соціальних відносин. Врахування різних контекстів та аспектів гри веде до більш глибокого розуміння механізмів, що лежать в основі прийняття рішень в умовах обмежених ресурсів та конфлікту інтересів.

# РОЗДІЛ 2
# ФУНКЦІОНАЛ ТА ВИБІР ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ

## 2.1. Опис вимог до застосунку

Мета застосунку полягає у вивченні різних стратегій, характерів та поведінок гравців у грі "Ультиматум". Це досягається шляхом збору даних про раунди гри та їх подальшого аналізу. Основний акцент робиться на дослідженні взаємодії між гравцями з різними стратегіями та характерами, щоб визначити оптимальні шляхи досягнення цілей гри.

Для цього необхідно наступне:

1. Програма, яка б легко масштабувалась задля додавання нових стратегій та нових ботів з різними поведінками.
2. Візуалізація результатів або за допомогою виводу в консоль, або краще за допомогою ui програми.

Вимоги:

- можливість створювати ботів
- можливість надавати ботам свою унікальну поведінку, приймати самостійні рішення, керувати рівнями прийняття та розподілу.
- можливість грати в різні модифікації гри, як от диктатор, тощо.
- можливість проводити тестування на різних групах ботів. Залучати різних ботів до різних груп.
- можливість легко переглянути результати в ui, якщо можливо. Або в консолі.

- швидкодія
- можливість оброблення багатьох ботів і багатьох раундів
- паралельність
- оптимізація ресурсів

Цей розділ створений для детального розгляду вимог та обґрунтувань щодо функціональності програмного продукту, який має допомагати в дослідженні стратегій та поведінки гравців у грі "Ультиматум". У подальших розділах буде надано конкретний план щодо реалізації цих вимог.

### 2.2. Вибір мови програмування

Основні характеристики мови програмування, на які треба звернути увагу конкретно в моєму випадку - це:

- можливість розширення
- швидкодія
- паралельність
- зручність
- можливість створення ui програм

Давайте тепер розглянемо кожну характеристику окремо.

#### 2.2.1. Розширення

Тема розширення програм є ключовою у сучасному програмуванні, оскільки вона визначає гнучкість та легкість вдосконалення програмного забезпечення. У цьому контексті розглядатимуться різні мови програмування, з наголосом на Java, та їхні можливості щодо розширення функціональності програм.

Java, як мова програмування, відома своєю платформенною незалежністю, що дозволяє розробляти програми, які можна виконувати на різних операційних системах. Проте, однією з ключових переваг Java є його велика розширюваність. Розширення програм на Java відбувається за допомогою використання різноманітних механізмів, які надають можливість легко впроваджувати нові функції та вдосконалювати існуючий код.

Один з ефективних методів розширення програм на Java - використання інтерфейсів та абстракцій. Інтерфейси дозволяють визначити набір методів, які повинен реалізувати будь-який клас, що реалізує даний інтерфейс. Це створює можливість для введення нових класів, які відповідають вимогам інтерфейсу, тим самим розширюючи функціональність програми без необхідності модифікації існуючого коду.

Ще одним механізмом розширення програм на Java є використання абстракцій. Абстракції дозволяють визначити загальний функціонал, приховуючи деталі реалізації. Це створює можливість створювати нові класи, які розширюють абстракцію та надають нові можливості програмі.

Патерни проектування також грають важливу роль у розширенні програм на Java. Вони визначають типові прийоми розв'язання конкретних проблем та дозволяють легко впроваджувати ці рішення в існуючий код. Використання патернів дозволяє забезпечити гнучкість та розширюваність програмного забезпечення.

Крім того, у Java існує можливість використання динамічного завантаження класів, яке дозволяє додавати новий код під час виконання програми. Це робить можливим розширення програми без її перезапуску, що є важливою перевагою для додатків з великою кількістю користувачів.

Загалом, Java надає розробникам ряд інструментів та підходів, які сприяють легкості розширення програм. Інтерфейси, абстракції, патерни

проектування та динамічне завантаження класів - це всі засоби, які сприяють гнучкому та ефективному вдосконаленню програмного забезпечення.

Крім використання мови програмування Java, слід розглянути й інші підходи та техніки, що сприяють легкості розширення програм. Важливим аспектом є використання модульного програмування, яке дозволяє розділити програму на невеликі, самостійні модулі. Це сприяє покращенню та доповненню окремих частин програми без необхідності зміни інших.

Мови програмування, такі як Python, також володіють потужними механізмами для розширення програм. У Python, наприклад, існують динамічні типи даних та динамічна зміна об'єктів, що дозволяють на льоту додавати нові властивості та методи до існуючих класів. Це робить мову Python ефективною для створення розширюваних програм.

Іншим важливим аспектом є використання відкритих стандартів та API (інтерфейсів програмування застосунків). Використання стандартів дозволяє інтегрувати різні компоненти та бібліотеки, що в подальшому спрощує розширення програм шляхом додавання нових компонентів, які відповідають стандартам.

Зокрема, веб-розробка використовує різноманітні фреймворки та бібліотеки, які дозволяють легко розширювати функціональність веб-додатків. Наприклад, у JavaScript існують фреймворки, такі як React або Angular, які дозволяють створювати компоненти та модулі, які можна легко впроваджувати та масштабувати.

Іншим підходом до розширення програм є використання плагінів. Плагіни дозволяють динамічно додавати новий функціонал до програми, роблячи її більш гнучкою та адаптивною до змінних вимог. Такий підхід добре підтримується у ряді мов програмування та фреймворків.

У заключення, легкість розширення програм визначається комплексом підходів, які використовуються у програмуванні. Від використання інтерфейсів

та абстракцій у Java до динамічних типів у Python та використання модульного програмування та відкритих стандартів, всі ці методи допомагають створювати програми, які легко розширюються та адаптуються до змін вимог.

Я все-ж віддаю перевагу мові програмування java в цьому аспекті, адже вона має ООП концепцію, що дійсно спрощує розширення.

### 2.2.2. Швидкодія

Мови програмування відіграють ключову роль у розробці програмного забезпечення, забезпечуючи зручний і ефективний спосіб створення програм. Однією з найбільш поширених та впливових мов програмування є Java. Її виразна швидкодія та універсальність роблять її популярним вибором серед розробників.

Java володіє вражаючою швидкодією завдяки своєму використанню віртуальної машини Java (JVM). JVM є інтерпретатором байт-коду, що дозволяє виконувати Java-програми на різних платформах безпосередньо, без необхідності перекомпіляції. Це забезпечує високу переносимість програм та дозволяє розробникам ефективно взаємодіяти з різними операційними системами.

Однією з ключових особливостей швидкодії Java є використання технології Just-In-Time (JIT) компіляції. Коли програма запускається, код JIT компілюється безпосередньо в машинний код для конкретної платформи, що дозволяє підвищити продуктивність та забезпечити оптимальну швидкодію виконання.

У порівнянні з іншими мовами програмування, Java відзначається ефективним управлінням пам'яттю. Вона використовує автоматичне управління пам'яттю, що дозволяє розробникам уникнути багатьох проблем, пов'язаних з витоками пам'яті та некоректними вказівниками.

Крім того, Java підтримує багато потоків, що сприяє паралельному виконанню завдань та поліпшує швидкодію програм в умовах багатозадачності. Висока швидкодія потоків у Java робить її відмінним вибором для розробки програм, які вимагають швидкого та ефективного взаємодії з великою кількістю одночасних завдань.

Таким чином, швидкодія Java обумовлюється не лише використанням віртуальної машини та JIT компіляцією, але і управлінням пам'яттю, підтримкою багатьох потоків та іншими технічними аспектами, які роблять її відмінною мовою програмування для різних видів розробок.

### 2.2.3. Паралельність

Паралельність у програмуванні відіграє ключову роль у досягненні ефективності та швидкодії виконання завдань. Розглянемо цей аспект у різних мовах програмування, звертаючи увагу на особливості реалізації багатопоточності в Java.

Java є однією з мов програмування, яка активно підтримує багатозадачність та паралельність. Одним із ключових інструментів для цього є вбудована підтримка многозадачності за допомогою потоків. Потоки у Java дозволяють виконувати кілька завдань паралельно, що особливо корисно в умовах, коли програма повинна обробляти багато подій або виконувати різні завдання одночасно.

Один зі способів створення потоків у Java - це використання класу Thread. Розробники можуть успадковувати цей клас та реалізовувати метод run(), де визначається виконавчий код для потоку. Однак існує й інший підхід, який полягає в реалізації інтерфейсу Runnable та передачі його екземпляра об'єкту класу Thread.

Java також надає інструменти для координації потоків та управління їхнім виконанням. Синхронізація та монітори дозволяють уникати конфліктів при

одночасному доступі до ресурсів з боку різних потоків. Крім того, в Java присутній пакет java.util.concurrent, який містить високорівневі конструкції для роботи з паралельністю, такі як ExecutorService та ThreadPoolExecutor.

Загалом, Java виявляється потужним інструментом для реалізації багатозадачних та паралельних програм. Вона надає зручні засоби для створення та керування потоками, а також високорівневі конструкції для оптимального використання ресурсів системи.

Паралельність в Java також підтримується концепцією фреймворку Fork/Join, який забезпечує велику гнучкість та ефективність при роботі з паралельними завданнями. Цей фреймворк вперше був представлений в Java 7 і дозволяє автоматично розподіляти завдання між різними потоками.

Однією з основних ідей Fork/Join є розбиття великого завдання на менші підзавдання, які можуть виконуватися паралельно. Коли підзавдання виконуються, їх результати об'єднуються для отримання остаточного результату. Цей підхід дозволяє автоматично використовувати всі доступні ресурси системи та ефективно розділяти навантаження між потоками.

Крім того, у Java 8 та пізніших версіях було введено новий підхід до паралельної обробки даних за допомогою стрімів та лямбда-виразів. Стріми надають зручний спосіб виконання операцій на колекціях даних паралельно. Можливість використання лямбда-виразів дозволяє дуже ефективно виражати операції, що виконуються паралельно над елементами колекції.

Варто зазначити, що правильне використання паралельності у Java вимагає уважного контролю над синхронізацією та уникненням гонок даних, щоб уникнути неправильної обробки та збереження консистентності даних.

Отже, завдяки різноманітним інструментам та конструкціям, Java є мовою програмування, яка дозволяє розробникам легко та ефективно реалізовувати паралельні програми, забезпечуючи високу продуктивність та оптимальне використання ресурсів системи.

Паралельність у Java додатково підсилюється використанням реактивного програмування, особливо за допомогою фреймворку Project Reactor. Реактивне програмування є парадигмою, яка активно використовується для розробки високопродуктивних та масштабованих систем.

Project Reactor включає в себе імплементацію реактивного стилю програмування для мови Java. Основною перевагою реактивного програмування є здатність асинхронно реагувати на події та працювати з потоками даних, використовуючи концепції витоку (flux) та одиночного елементу (mono).

Однією з ключових переваг Project Reactor є можливість створення реактивних потоків даних, які можуть легко адаптуватися до паралельної обробки. Комбінування реактивного програмування з багатозадачністю Java, зокрема з використанням потоків, дозволяє розробникам створювати високопродуктивні та ефективні реактивні застосунки.

Додатково, Project Reactor забезпечує розширені можливості обробки помилок та керування винятками в асинхронному середовищі. Це дозволяє створювати надійні та стабільні реактивні системи, які можуть ефективно взаємодіяти з великою кількістю подій та даних одночасно.

Отже, використання Project Reactor та реактивного програмування у Java доповнює підтримку паралельності, роблячи мову більш гнучкою та підходящою для сучасних вимог до розробки додатків, які потребують високої продуктивності та відмінного керування асинхронністю.

### 2.2.4. Зручність

Зручність використання мови програмування — це ключовий аспект, який визначає ефективність та задоволення розробників під час створення програмного забезпечення. Розглянемо цей аспект в контексті різних мов програмування, зосереджуючи увагу на Java.

Java відзначається високим рівнем зручності, що робить її однією з найпопулярніших мов програмування у світі. Перш за все, її синтаксис є досить простим та легким для розуміння, що дозволяє новачкам швидко освоювати основи програмування.

Однією з важливих переваг Java є її платформенна незалежність. Це означає, що програми, написані на Java, можуть виконуватися на різних платформах без змін вихідного коду. Це робить Java ідеальним вибором для розробки крос-платформених додатків, оскільки розробники можуть писати код один раз і використовувати його на різних операційних системах.

Другорядність мови підтримки допомагає полегшити розробку та управління проектами. Велика кількість готових бібліотек та фреймворків в спільноті Java забезпечує розробникам засоби для швидкої імплементації функціональності без необхідності написання великої кількості коду "з нуля".

Платформа Java також має величезну базу знань, оскільки вона існує протягом багатьох років. Це означає, що розробники можуть легко отримати доступ до великої кількості документації, ресурсів та форумів, що полегшує розв'язання проблем та навчання нових концепцій.

Окрім того, інтегроване середовище розробки (IDE), таке як IntelliJ IDEA або Eclipse, робить написання коду на Java ще зручнішим. Завдяки автоматичному доповненню коду, інструментам рефакторингу та підтримці версій, розробники можуть прискорити процес розробки та зменшити кількість помилок.

Додатково, Java володіє великою увагою до безпеки та надійності, що робить її привабливою для розробки великих та критичних застосунків. Механізми управління пам'яттю та автоматичні перевірки на витоки пам'яті допомагають уникнути багатьох типових помилок.

Java також відома своєю підтримкою об'єктно-орієнтованого програмування (ООП), що сприяє створенню чистого та організованого коду.

Інкапсуляція, успадкування та поліморфізм дозволяють розробникам створювати гнучкі та легко розширювані програми.

Важливим фактором є також велика спільнота розробників, яка активно допомагає одне одному, ділиться досвідом і розробленими бібліотеками. Це створює позитивне середовище для вирішення проблем, обговорення найкращих практик та швидкого отримання відповідей на питання.

У контексті зручності Java також активно використовується для розробки веб-застосунків, завдяки фреймворкам як Spring і Apache Struts. Ці фреймворки пропонують високорівневі конструкції та шаблони, що спрощують розробку веб-додатків та забезпечують високий рівень зручності для програмістів.

Отже, Java виступає в ролі потужного та зручного інструменту для розробки різноманітних програм, починаючи від мобільних додатків і закінчуючи великомасштабними системами. З врахуванням зазначених переваг, розробники можуть швидко та ефективно втілювати свої ідеї в життя, користуючись різноманіттям інструментів та можливостей, які пропонує мова програмування Java.

### 2.2.5. Можливість створення UI програм

Розглядаючи можливості створення користувацького інтерфейсу (UI) в різних мовах програмування, зосередимо увагу на особливостях, які пропонує Java для вирішення завдань, пов'язаних з розробкою графічного інтерфейсу користувача.

Java надає декілька зручних інструментів для розробки інтерфейсів, що дозволяє розробникам створювати візуально привабливі та функціональні додатки. Однією з найбільш використовуваних бібліотек для роботи з графічним інтерфейсом в Java є JavaFX.

JavaFX є сучасною бібліотекою для розробки багатофункціональних, стильних та відзначених високою продуктивністю інтерфейсів. Завдяки мові

опису інтерфейсу FXML, розробники можуть описувати структуру UI у вигляді розмітки, що полегшує розробку та підтримку коду.

JavaFX надає можливості для використання стандартних елементів управління, таких як кнопки, поля введення, таблиці та інші. Однак відзначається також можливістю створення власних елементів управління та їхньої настройки за допомогою стилів CSS, що дозволяє досягти індивідуального та сучасного дизайну.

Окрім JavaFX, існують також інші бібліотеки та фреймворки, які використовуються для розробки UI в Java, наприклад, Swing та AWT. Вони надають високий рівень абстракції та спрощують процес створення вікон, діалогових вікон та елементів управління.

Однією з вагомих переваг Java в контексті UI є її крос-платформенність. Додатки, написані на Java, можуть запускатися на різних операційних системах без значних змін у вихідному коді, що робить її ідеальним вибором для розробки платформонезалежних програм.

Додатково до JavaFX, важливим аспектом створення користувацького інтерфейсу в Java є використання бібліотек, таких як Apache Pivot, SWT (Standard Widget Toolkit) та Java's Abstract Window Toolkit (AWT). Кожна з цих бібліотек має свої особливості та вигоди в контексті розробки UI.

Apache Pivot відомий своєю простотою та гнучкістю. Він надає широкий набір готових елементів управління та має високий рівень абстракції, що полегшує розробку. Бібліотека SWT, в свою чергу, активно використовується у проекті Eclipse та володіє низьким рівнем абстракції, що дозволяє розробникам більше контролю над графічним інтерфейсом.

AWT, включений в Java стандартно, є дещо менш потужним порівняно з більш сучасними бібліотеками, але все ще використовується у простих аплікаціях та прототипах.

Java також має підтримку для розробки мобільних користувацьких інтерфейсів за допомогою платформи JavaFX Mobile та більш нових інструментів для створення мобільних додатків, таких як JavaFX Mobile SDK.

Загалом, зручність створення користувацького інтерфейсу в Java полягає у доступності різноманітних інструментів та бібліотек, які надають розробникам велику свободу та можливість вибору залежно від вимог конкретного проекту. Це робить Java привабливою мовою для створення не лише десктопного, а й мобільного та веб-інтерфейсу, дозволяючи розробникам легко втілювати свої ідеї в користувацькі та естетичні програми.

### 2.2.8. Інші особливості Java

Java, як мова програмування, є винятковою у своїй універсальності та здатності задовольняти різноманітні потреби розробників завдяки багатству бібліотек, фреймворків та інших інструментів. Однією з визначальних особливостей Java є її крос-платформенність, яка дозволяє виконувати програми на різних операційних системах без потреби внесення змін у вихідний код.

Однією з найважливіших бібліотек є Java Standard Edition (SE), яка включає в себе базовий набір класів та інтерфейсів для створення різноманітних програм. У цьому контексті, Java Collections Framework надає широкий спектр структур даних, таких як списки, масиви, мапи, що дозволяє розробникам ефективно управляти та обробляти даними.

Одним з найбільш важливих фреймворків для розробки веб-додатків у Java є Spring Framework. Spring вирізняється своєю гнучкістю та модульністю, дозволяючи розробникам вибирати та використовувати лише ті компоненти, які необхідні для конкретного проекту. Він забезпечує високий рівень абстракції для управління конфігурацією, безпекою, обробкою подій та багатьма іншими аспектами розробки.

Hibernate — це ще одна важлива технологія у світі Java, яка вирішує завдання роботи з базами даних. Використовуючи концепції відображення об'єктів на реляційні дані, Hibernate дозволяє розробникам працювати з базами даних більш абстрактно, спрощуючи роботу з SQL-запитами та забезпечуючи високий рівень переносу коду між різними Системами управління базами даних.

У сфері розробки мобільних додатків для Android платформи, Java використовується як основна мова програмування, де вона набула особливого значення завдяки своїй стабільності та широким можливостям розробки.

Однією з особливостей Java також є її активна спільнота розробників. Онлайн-форуми, блоги, конференції та інші ресурси роблять Java платформою, де розробники можуть ділитися досвідом, отримувати поради та вирішувати проблеми разом з іншими членами спільноти.

Таким чином, Java особлива своєю універсальністю, крос-платформенністю та розширеним спектром інструментів, які надають розробникам всі необхідні засоби для того, щоб створювати різноманітні та ефективні програми у різних областях розробки.

У сфері обробки масивних об'ємів даних Java пропонує Apache Hadoop — фреймворк для обробки та аналізу великих об'ємів даних в розподіленому середовищі. Це робить Java ефективним вибором для розробки систем обробки Big Data.

Ще однією ключовою особливістю Java є її вбудована підтримка мережевого програмування. ServerSocket та Socket класи дозволяють створювати мережеві застосунки, що взаємодіють через TCP або UDP протоколи. Це робить Java відмінним вибором для розробки серверів, клієнтських додатків та інших мережевих застосунків.

У сфері веб-розробки, окрім вже згаданого Spring Framework, Java пропонує JavaServer Faces (JSF) та Apache Struts. JSF — це фреймворк, який

дозволяє розробникам побудовувати високопродуктивні веб-інтерфейси на основі компонентів. З іншого боку, Apache Struts дозволяє створювати масштабовані веб-додатки на основі паттерну проектування Model-View-Controller (MVC).

Однією з важливих переваг Java є інтегроване середовище розробки (IDE) — IntelliJ IDEA та Eclipse, які надають широкий набір інструментів для підтримки розробки, таких як автодоповнення коду, дебаггінг, аналіз коду та інші. Це робить процес розробки в Java більш комфортним та продуктивним.

### 2.2.9. Висновок

Java - ідеально підходить для моєї задачі, тому буде використана саме ця мова програмування для написання ботоферми.

## ВИСНОВКИ ДО РОЗДІЛУ 2

У другому розділі було розглянуто функціонал для ботоферми та вибір інструментів за допомогою яких можна досягти необхідних вимог для розробки ПЗ до моєї теми кваліфікаційної роботи.

Java, з своєю винятковою універсальністю та багатоманіттям інструментів, визначається як одна з найбільш важливих та впливових мов програмування. Її крос-платформенність, широкий набір бібліотек і фреймворків роблять її зручним інструментом для розв'язання різних завдань в різних сферах розробки.

Java активно використовується в розробці великих, масштабних проектів, в тому числі систем управління базами даних, веб-додатків, мобільних додатків та багатьох інших.

# РОЗДІЛ 3
# ПРОЕКТУВАННЯ ТА РОЗРОБКА ЗАСТОСУНКУ

## 3.1. GOF Патерни

Часто в ООП програмуванні використовують різні стандартні підходи до написання коду. Такі підходи ще називають патернами проектування.

Патерни проектування є ефективним інструментом для вирішення типових проблем у програмуванні та розробці програмного забезпечення. Одними з найбільш визнаних і широко використовуваних є GOF патерни, що їх розробила "Банда Чотирьох" — Еріх Гамма, Річард Хелм, Ральф Джонсон і Джон Вліссідес.

Банда Чотирьох вперше представила GOF патерни у своїй книзі "Design Patterns: Elements of Reusable Object-Oriented Software", виданій у 1994 році. Ця книга стала біблією для багатьох програмістів і розробників програмного забезпечення, оскільки надала чіткі та ефективні рішення для численних проблем проектування.

GOF патерни дозволяють розробникам створювати гнучкі, легко зрозумілі та підтримувані системи. Вони широко використовуються в областях програмування, де важлива гнучкість та розширюваність. GOF патерни поділяються на 3 види:

1 Породжувальні патерни
2 Структурні патерни
3 Поведінкові патерни

Давайте розглянемо кожен з них. Також розглянемо деякі приклади до кожного з видів.

### 3.1.1. Породжувальні патерни (Creational Patterns)

Породжувальні патерни (Creational Patterns) — це одна з категорій патернів проектування, які вирішують завдання створення об'єктів та їх подальшої ініціалізації. Ці патерни надають механізми для створення об'єктів таким чином, щоб програма була більш гнучкою та незалежною від структури її об'єктів. Основні цілі породжувальних патернів включають у себе:

- забезпечення гнучкості:

Патерни дозволяють створювати об'єкти без прив'язки до їх конкретних класів, роблячи систему більш гнучкою та витривалою до змін.

- сховище деталей створення:

Патерни дозволяють сховати деталі створення об'єктів, роблячи їхню реалізацію невидимою для клієнтського коду.

- забезпечення єдиної точки доступу:

Деякі породжувальні патерни, такі як "Одиночка" (Singleton), гарантують, що клас має лише один екземпляр та надає глобальний доступ до нього.

Основні породжувальні патерни включають у себе:

- абстрактна фабрика (Abstract Factory):

Надає інтерфейс для створення сімейства пов'язаних або взаємозалежних об'єктів без їх конкретної специфікації.

- будівельник (Builder):

Розділяє процес конструювання об'єкта від його представлення, дозволяючи створювати різні представлення об'єкта.

- фабричний метод (Factory Method):

Визначає інтерфейс для створення об'єктів, але залишає вибір їхнього конкретного класу нащадку.

- одиночка (Singleton):

Гарантує, що клас має лише один екземпляр і надає глобальний доступ до нього.

- прототип (Prototype):

Дозволяє створювати нові об'єкти шляхом копіювання існуючих об'єктів, щоб уникнути складного процесу їхнього створення.

Ці патерни надають програмістам стандартні рішення для вирішення проблем, пов'язаних із створенням об'єктів, та сприяють покращенню структури та гнучкості програмного забезпечення.

### 3.1.2. Структурні патерни (Structural Patterns)

Структурні патерни (Structural Patterns) — це одна з категорій патернів проектування, які спрямовані на створення більш ефективних та гнучких структур програмного забезпечення шляхом визначення способів композиції класів та об'єктів.

Основні мети структурних паттернів включають:

- композиція об'єктів:

Структурні патерни дозволяють складати об'єкти та класи в більш складні структури, спрощуючи їх використання та розширення.

- заміна об'єктів:

Ці патерни забезпечують можливість замінювати один об'єкт іншим, не змінюючи при цьому структуру коду.

- забезпечення гнучкості:

Структурні патерни спрощують роботу з різними частинами системи, роблячи її більш гнучкою та легше розширюваною.

Основні структурні паттерни включають у себе:

- адаптер (Adapter):

Дозволяє об'єктам з несумісними інтерфейсами працювати разом. Адаптер перетворює інтерфейс одного класу в інший, який очікує клієнт.

- міст (Bridge):

Розділяє абстракцію від її реалізації так, щоб обидві можливо було змінювати незалежно одна від одної.

- компонувальник (Composite):

Дозволяє клієнтам обробляти окремі об'єкти та їхні складові однаковим чином.

- декоратор (Decorator):

Додає новий функціонал об'єктам, не змінюючи їхнього інтерфейсу.

- фасад (Facade):

Надає спрощений інтерфейс для взаємодії з складним підсистемою.

- проксі (Proxy):

Контролює доступ до об'єкта, дозволяючи виконувати додаткові дії при його виклику.

Структурні патерни допомагають створювати більш гнучкі та підтримувані програмні системи, де структура об'єктів та їх взаємодія важлива для успішної розробки та підтримки.

### 3.1.3. Поведінкові патерни (Behavioral Patterns)

Поведінкові патерни (Behavioral Patterns) — це категорія патернів проектування, які визначають способи взаємодії між об'єктами, а також визначають відповідальність між об'єктами. Основна мета цих паттернів — забезпечити ефективний та ефективний спосіб комунікації та координації між об'єктами для досягнення певних цілей.

Основні мети поведінкових паттернів включають:

- розподіл обов'язків:

Забезпечують чітку відповідальність та розподіл обов'язків між об'єктами, щоб система була легко розширюваною та підтримуваною.

- покращення взаємодії:

Забезпечують ефективну взаємодію об'єктів та дозволяють змінювати цю взаємодію без необхідності зміни самого об'єкта.

- зменшення залежностей:

Зменшують залежності між класами, роблячи систему менш вразливою до змін та легше розширюваною.

Основні поведінкові патерни включають у себе:

- стратегія (Strategy):

Визначає сімейство алгоритмів, робить їх взаємозамінними та дозволяє об'єкту вибрати алгоритм на льоту.

- спостерігач (Observer):

Визначає залежність одного об'єкта від змін у іншому об'єкті, так щоб будь-яка зміна стану одного об'єкта автоматично повідомляла інших.

- сповіщення (Command):

Інкапсулює запитання у об'єкті та дозволяє параметризувати клієнта об'єктом, забезпечуючи та зберігаючи запитання.

- ланцюг відповідальності (Chain of Responsibility):

Дозволяє передавати запити послідовно через ланцюг об'єктів, які можуть обробити або відхилити запит.

- інтерпретатор (Interpreter):

Визначає граматику мови та надає інтерпретатор для виконання виразів цієї мови.

- ітератор (Iterator):

Дозволяє послідовно переглядати елементи складної структури об'єктів без розкриття її внутрішньої реалізації.

- стан (State):

Змінює поведінку об'єкта при зміні його внутрішнього стану, надаючи вигляд, що змінюється клас.

- стратегія (Visitor):

Дозволяє оголошувати нові операції для набору об'єктів без зміни їхніх класів.

Ці патерни допомагають реалізувати гнучкі та підтримувані системи, забезпечуючи ефективну взаємодію об'єктів та простоту їхньої модифікації.

### 3.2. Проектування

Тепер, коли ми розібрались в різних патернах проектування, ми можемо приступити до проектування нашої ботоферми. Спершу розберемось в термінології:

Гравець А - гравець, який пропонує ділення.

Гравець В - гравець, який приймає або відхиляє ділення.

Гра (Game) - Позначення, яке означає одну гру в “Ультиматум” між гравцем А і В.

Раунд (Round) - Позначення, яке вказує на ігри (Game) для всіх гравців (ботів) зіграних по одному разу. Тобто якщо ми матимемо 10000 ботів і 1 раунд, тоді відбудеться 5000 ігор (Game)

Тест - Повноцінний цикл зі всіх раундів

Тепер необхідно виділити основні сутності, або ж компоненти нашого коду. Виділимо наступні компоненти:

Все, що пов’язано з Ботом:

1. Interface A
2. Interface B
3. Abstract class Bot

Все, що пов’язано з грою (Game):

1. Game

2. GameMetaData
3. Offer
4. Results

Все, що пов’язано з раундами (Round):

1. Interface RoundHandler
2. Abstract class TwoBotTypesRoundHandler
3. Abstract class ThreeBotTypesRoundHandler

Всі класи та інтерфейси, які приведені вище, відносяться до ядра (core) ботоферми. Там відокремлена вся логіка для створення ботів, реалізації їхньої логіки, циклу раундів, розрахунку виграшів та всього процесу виконання тестів над ботами.

Також окремо існує ui модуль, який використовується для візуалізації тесту. Основними компонентами в ньому є:

1. ProgressBar
2. Diagram
3. DiagramElement
4. UiManager

Тепер розглянемо як це все виглядає.

### 3.2.1. A, B, Bot

Основною ідеєю цих інтерфейсів та класів є абстракція основної логіки гравців. Кожен гравець має поведінку, коли він грає за A, та поведінку, коли він грає за B. Гравець A (рис. 3.1.) - може лише запропонувати (offer) пропозицію. У той час як гравець B (рис. 3.1.) може вирішити (decide) чи приймати йому пропозицію чи відхилити.

Також кожен гравець (бот) може грати як за A так і за B. Тому клас Bot (рис. 3.2.) імплементує і A і B. Водночас, ми хочемо мати різні поведінки для ботів,

тому не повинні реалізовувати методи offer та decide безпосередньо у класі Bot. Для цього ми будемо створювати окремі класи, які будуть розширювати (extends) клас Bot, та реалізовуватимуть усі необхідні методи. У такому випадку необхідно помітити клас Bot як абстрактний (abstract).

У класі Bot, ми бачимо іще декілька додаткових методів та поле ID. Поле ID необхідне щоб відрізняти ботів один від одного. Воно генерується рандомно за допомогою класу RandomStringUtils від appache commons. Щодо методів, у нас їх два:

Перший - afterGame(GameMetaData). Він потрібен для того, аби бот проаналізував гру по її закінченню. У більшості це потрібно саме для аналітиків. Інші ж не аналізують, тому за умовчуванням цей метод нічого не робить.

Другий - wasIOffering(GameMetaData). Цей метод поверне true, якщо бот грав у якості гравця A.

A.java

```
package com.core.bots;

import com.core.game.Offer;

public interface A {

    Offer offer();
}
```

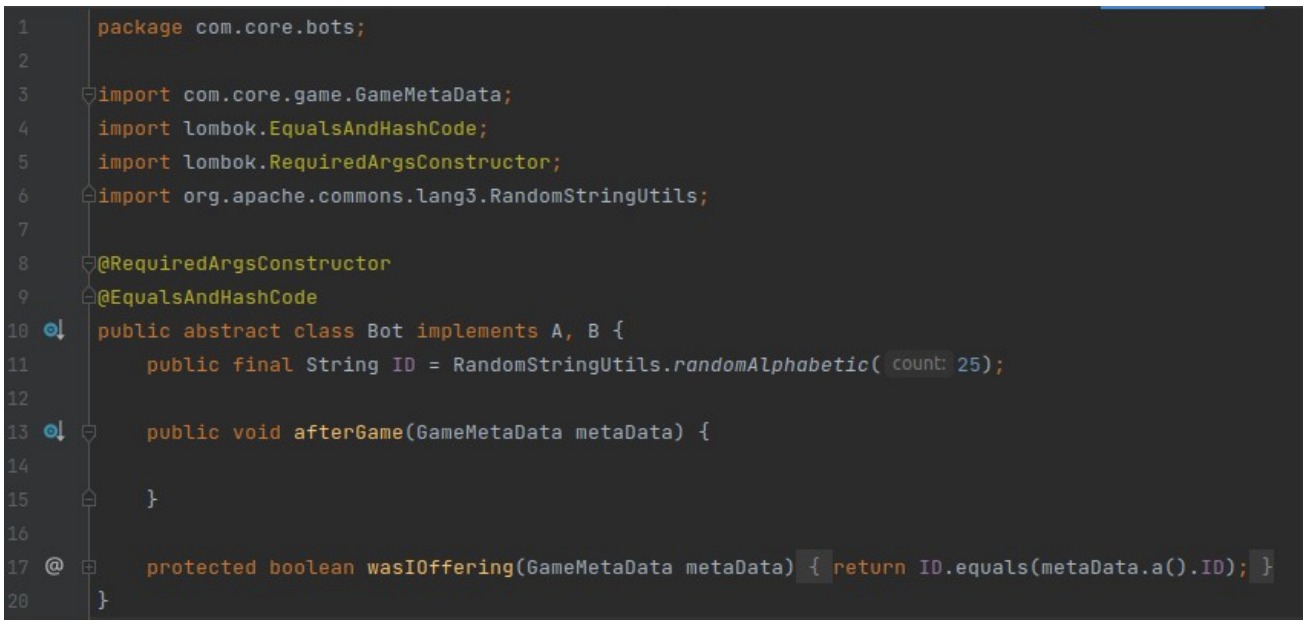
B.java

```
package com.core.bots;

import com.core.game.Offer;

public interface B {
    boolean decide(Offer offer);
}
```

Рис. 3.1. Інтерфейси A та B



```java
package com.core.bots;

import com.core.game.GameMetaData;
import lombok.EqualsAndHashCode;
import lombok.RequiredArgsConstructor;
import org.apache.commons.lang3.RandomStringUtils;

@RequiredArgsConstructor
@EqualsAndHashCode
public abstract class Bot implements A, B {
    public final String ID = RandomStringUtils.randomAlphabetic( count: 25);

    public void afterGame(GameMetaData metaData) {

    }

    protected boolean wasIOffering(GameMetaData metaData) { return ID.equals(metaData.a().ID); }
}
```

Рис. 3.2. Абстрактний клас Bot

### 3.2.2. Game, GameMetaData, Offer, Results

Далі представлено класи, які безпосередньо відносяться ігор, їх обробки та підрахунку результатів.

Клас Game (рис. 3.3.) - це клас, який відповідає за саму гру. Тут відбувається пропозиція від гравця A, та рішення від гравця B. Після чого викликаються методи afterGame, які забезпечують виконання “роздумів” ботів після гри, та вносяться результати.

Рекорди GameMetaData та Offer (рис. 3.4.) - це рекорди, які слугують моделями для передачі даних між класами. Offer - зберігає пропозицію, в якій вказаний відсоток для гравця A, та відсоток для гравця B. GameMetaData - це рекорд, який зберігає всю необхідну інформацію про гру. Хто грав, Який був офер, чи бу він прийнятий та якою була ставка.

Ну і нарешті клас Results (рис. 3.5.). Цей клас забезпечує логіку для обчислення та зберігання результатів всього тесту. В нього можна додати запис після гри за допомогою методу addResults(GameMetaData). Також він може

згенерувати Report за допомогою метода report(). Report - це внутрішній клас класу Results, Який вміє видавати результати, або друкувати їх в консоль.

Друк в консоль необхідний тоді, коли ми хочемо подивитись якомога детальні дані про тест. Наприклад дані по кожному окремому боту. Вміщувати це в ui - доволі важка операція, та і не завжди необхідна, тому було прийняте рішення залишити вивід в консоль як метод спостереження повної інформації.

```java
package com.core.game;

import com.core.bots.Bot;

public record Game(Bot a, Bot b) {

    public void startGame(Results results) {
        int stake = 100;
        Offer offer = a.offer();
        boolean isAccepted = b.decide(offer);

        GameMetaData metaData = new GameMetaData(a, b, offer, isAccepted, stake);
        a.afterGame(metaData);
        b.afterGame(metaData);

        results.addResults(metaData);
    }
}
```

Рис. 3.3. Клас Game

GameMetaData.java

```java
package com.core.game;

import com.core.bots.Bot;

public record GameMetaData(Bot a, Bot b, Offer offer, boolean isAccepted, int stake) {
}
```

Offer.java

```java
package com.core.game;


public record Offer(int aPercentage, int bPercentage) {
}
```

Рис.3.4. GameMetaData та Offer рекорди

```
public class Results {
    private final Map<Bot, Double> winsPerBot = new ConcurrentHashMap<>();
    private final Map<Class<? extends Bot>, Double> winsPerBotTypes = new HashMap<>();
    private final Map<Class<? extends Bot>, Pair<Integer, Double>> avgWinsPerBotType = new HashMap<>();
    private Double loss = 0d;
    private Double stakes = 0d;

    public void addResults(GameMetaData metaData) {...}

    public Report report() {...}

    private void putWinsPerBot(GameMetaData metaData) {...}

    private void putWinsPerBotType() {...}

    private void putAvgWinsPerBotType(GameMetaData metaData) {...}

    private Pair<Integer, Double> updateAvgWinsPair(Pair<Integer, Double> gamesCountAndValue, double newValue) {...}

    private double getPercentage(int stake, int percent) { return (percent * stake) / 100d; }

    private double getPercentFromTotal(double value) { return (value * 100) / stakes; }

    public class Report {
        public Map<Bot, Double> winsPerBot() {
            return winsPerBot;
        }

        public Map<Class<? extends Bot>, Double> winsPerBotTypes() { return winsPerBotTypes; }

        public Map<Class<? extends Bot>, Pair<Integer, Double>> avgWinsPerBotType() { return avgWinsPerBotType; }

        public void printResults() {...}

        private void printBlock(String name, Runnable printMethod) {...}

        private void printResultsByBot() {...}

        private void printResultsByBotType() {...}

        private void printAvgResultsByBotType() {...}

        private void printTotalWinsCountByBotType() {...}

        private void printGeneralResults() {...}
    }
}
```

Рис. 3.5. Results

### 3.2.3. Round - RoundHandlers

На цьому місці варто зупинитись детальніше, адже тут будуть використовуватись GOF патерни. Загалом, основним класом для запуску всього тесту - є RoundHandler. Він відповідає за:

- створення ботів
- визначення гравців A та B
- створення пар для гри
- підготовку раунду
- запуск раунду
- очищення після раунду
- цикл раундів
- конфігурацію тесту

На рис. 3.6. показано конфігурацію цього класу (його поля, або ж глобальні змінні класу). Там ми можемо побачити змінну для кількості раундів, для кількості ботів. Список ботів та пари ботів, які грають один між одним (Пари очищаються після кожного раунду і створюються нові). Також створюється об’єкт результатів, який передається для кожної гри в кожному раунді.

```
14
15      protected final int rounds = 1000;
16      protected final int botsCount = 10000;
17
18      protected final List<Bot> bots = new ArrayList<>(botsCount);
19      protected final List<Pair<Bot, Bot>> botPairs = new ArrayList<>( initialCapacity: botsCount/2);
20
21      protected final Results results = new Results();
22
```

Рис. 3.6. Поля RoundHandler’a

Далі можемо подивитись на основний метод цього класу, який запускає весь flow (рис. 3.7.). З сигнатури методу зрозуміло, що цей метод повертає Report для відображення даних в ui та консолі. Також цей метод приймає ProgressBar, про який трохи згодом в секції про ui.

Варто також звернути увагу на саму реалізацію методу RoundHandler.run(). Цей метод спочатку виконує початкову ініціалізацію за допомогою методу setup(), а потім запускає життєвий цикл раундів: Підготовка раунду, старт раунду, очищення раунду. Також по закінченню кожної ітерації ми оновлюємо значення ProgressBar'у. Наприкінці повернемо Report з о'бєкту results.

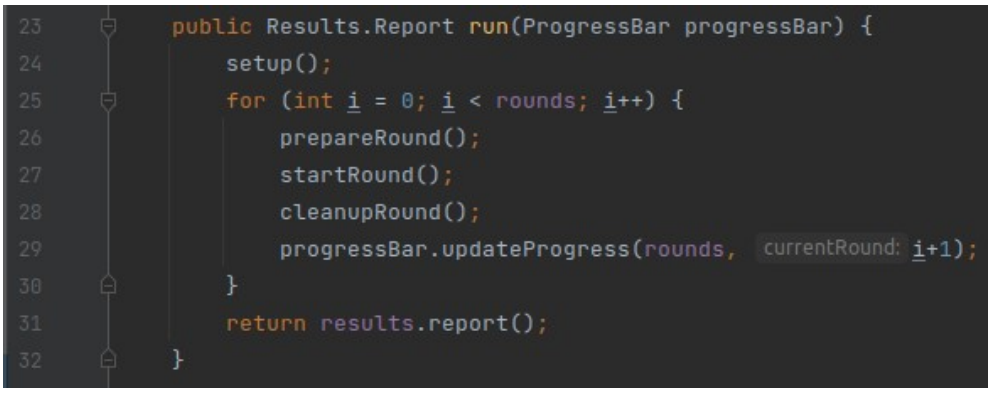



Рис. 3.7. RoundHandler.run()

Далі ідуть методи, які впроваджують життєвий цикл раунду (рис. 3.8.): prepareRound(), startRound(), cleanupRound().

В методі prepareRound() ми спершу перемішуємо колекцію bots, а потім створюємо пари для кожного бота і записуємо це в колекцію botPairs. В методі startRound ми для кожної пари ботів - створюємо та запускаємо гру (Game). А в методі cleanupRound() - ми просто очищаємо колекцію botPairs.

```
36      private void prepareRound() {
37          Collections.shuffle(bots);
38          for (int i = 0; i < bots.size() - 1; i+=2) {
39              Bot a = bots.get(i);
40              Bot b = bots.get(i + 1);
41              botPairs.add(Pair.of(a, b));
42          }
43      }
44
45      private void startRound() {
46          botPairs.stream()
47                  .map(pair -> new Game(pair.getLeft(), pair.getRight()))
48                  .forEach(game -> game.startGame(results));
49      }
50
51      private void cleanupRound() { botPairs.clear(); }
```

Рис. 3.8. Життєвий цикл раунду

Залишився лише один метод, про який не було розказано - це метод setup. Але справа в тому, що він не реалізований в цьому класі. Він реалізується в класах наслідниках. Це зроблено для того, щоб ми могли створювати різних ботів для різних тестів. Для прикладу ми хочемо проаналізувати тест, коли маємо 5000 егоїстів та 5000 альтруїстів. А в наступному тесті уже хочемо проаналізувати тільки аналітиків. Тобто для різних тестів ми маємо окремий клас.

Хоча, здається такий варіант може дублювати багато коду. Для прикладу: Ми хочемо запустити тест з 10000 ботів, де 50% - це егоїсти, а інші 50% - це альтруїсти. А також ми хочемо зробити те саме тільки 50% - це будуть егоїсти, а інші 50% - це аналітики. Якщо подумати, то код для створення відрізняється мінімально. І якраз тут нам на допомогу приходять патерни. А саме, Абстрактна фабрика.

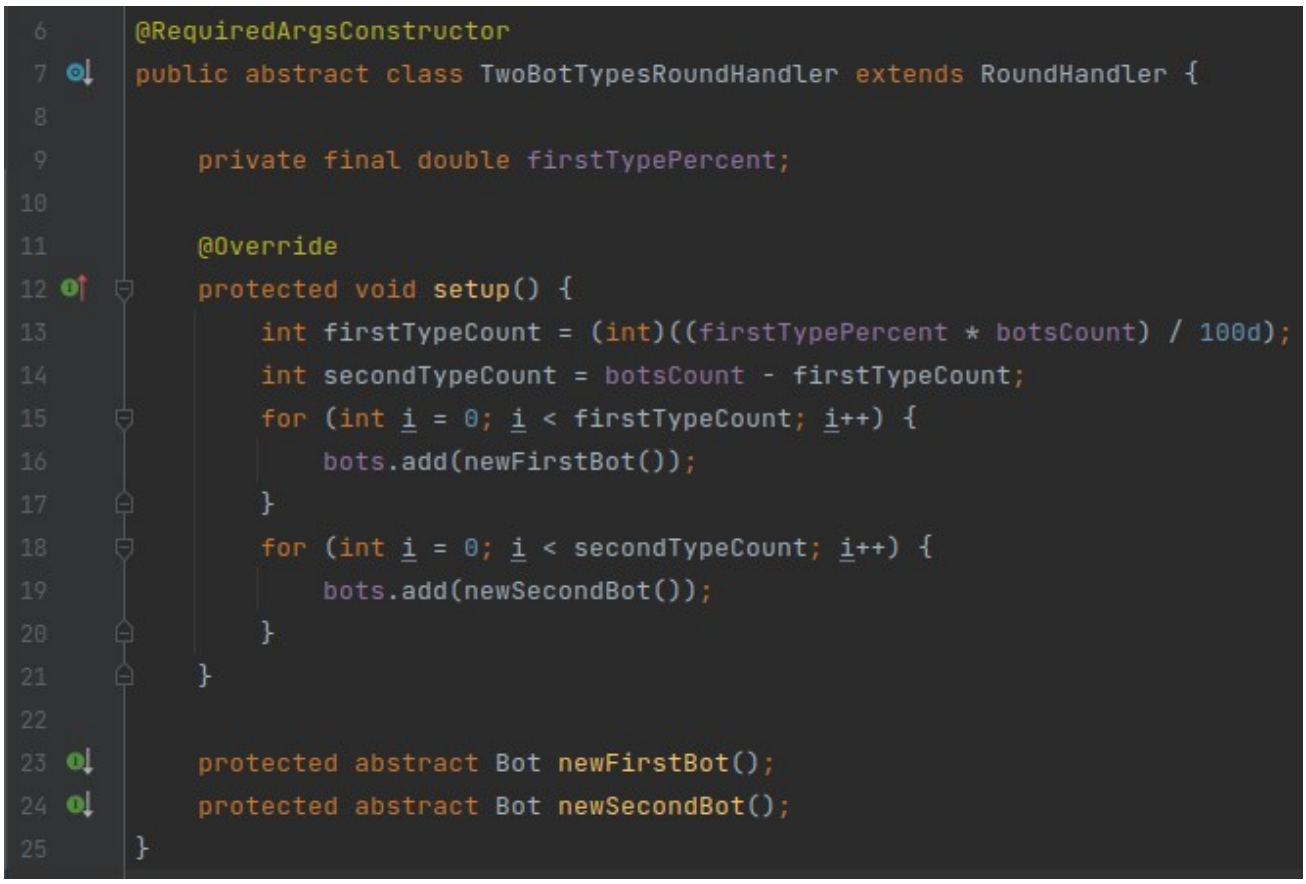
Абстрактна фабрика — це породжувальний паттерн проектування, який надає інтерфейс для створення сімейств пов'язаних або взаємозалежних об'єктів без зазначення їхніх конкретних класів. Цей паттерн вводить абстракцію, яка

дозволяє створювати об'єкти, що взаємодіють між собою, без необхідності знати їхню конкретну реалізацію.

Тепер поглянемо на код (рис. 3.9.). Тут ми бачимо TwoBotTypesRoundHandler клас, який теж являється абстрактним та реалізовує RoundHandler. Особливість цього класу полягає в тому, що той, хто його реалізує - може створювати тести з двома типами ботів, визначаючи їхні пропорції у відсотках.

Вся логіка створення ботів (метод setup()) - уже реалізована і її не потрібно повторювати. Цьому класу не важливо яких саме ботів ми створимо, допоки вони наслідуються від класу Bot.

Таким чином, щоб створити тест для егоїстів та альтруїстів, нам необхідно унаслідуватись від класу TwoBotTypesRoundHandler і реалізувати методи newFirstBot() та newSecondBot(). Як це виглядає можна подивитись на рис. 3.10.



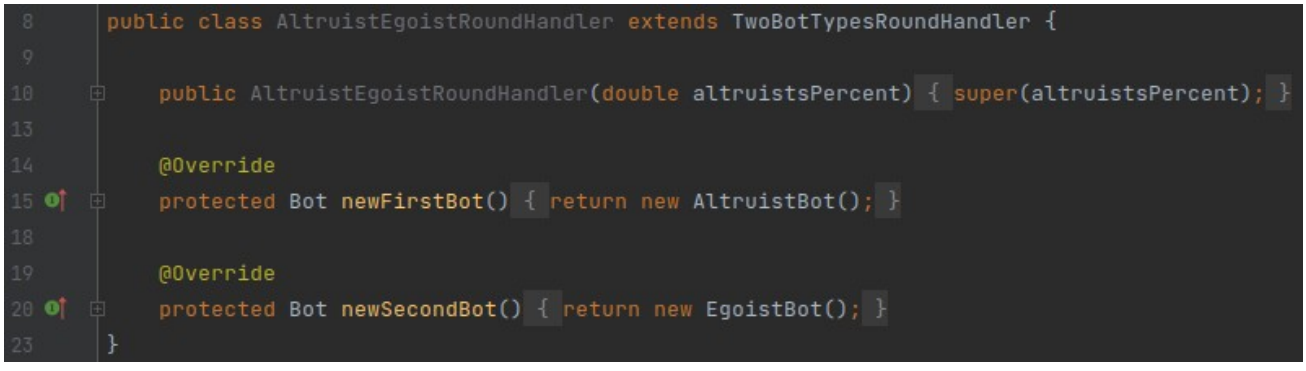
```
@RequiredArgsConstructor
public abstract class TwoBotTypesRoundHandler extends RoundHandler {

    private final double firstTypePercent;

    @Override
    protected void setup() {
        int firstTypeCount = (int)((firstTypePercent * botsCount) / 100d);
        int secondTypeCount = botsCount - firstTypeCount;
        for (int i = 0; i < firstTypeCount; i++) {
            bots.add(newFirstBot());
        }
        for (int i = 0; i < secondTypeCount; i++) {
            bots.add(newSecondBot());
        }
    }

    protected abstract Bot newFirstBot();
    protected abstract Bot newSecondBot();
}
```

Рис. 3.9. TwoBotTypesRoundHandler



```
public class AltruistEgoistRoundHandler extends TwoBotTypesRoundHandler {

    public AltruistEgoistRoundHandler(double altruistsPercent) { super(altruistsPercent); }

    @Override
    protected Bot newFirstBot() { return new AltruistBot(); }

    @Override
    protected Bot newSecondBot() { return new EgoistBot(); }
}
```

Рис. 3.10. AltruistEgoistRoundHandler

Аналогічна історія для RoundHandler’а з трьома різними ботами (Рис. 3. 11.). Тут ідея залишається тою самою, проте дещо ускладнюється логіка. Замість

одного відсотку з’являється три. Також додається перевірка на суму загального відсотку.

```java
public abstract class ThreeBotTypesRoundHandler extends RoundHandler {

    private final double firstTypePercent;
    private final double secondTypePercent;
    private final double thirdTypePercent;

    public ThreeBotTypesRoundHandler(double firstTypePercent, double secondTypePercent, double thirdTypePercent) {
        double percentSum = firstTypePercent + secondTypePercent + thirdTypePercent;
        if (percentSum != 100) {
            throw new IllegalArgumentException("All percents should be 100 in sum, but was: " + percentSum);
        }
        this.firstTypePercent = firstTypePercent;
        this.secondTypePercent = secondTypePercent;
        this.thirdTypePercent = thirdTypePercent;
    }

    @Override
    protected void setup() {
        int firstTypeCount = (int) ((firstTypePercent * botsCount) / 100d);
        int secondTypeCount = (int) ((secondTypePercent * botsCount) / 100d);
        int thirdTypeCount = (int) ((thirdTypePercent * botsCount) / 100d);
        for (int i = 0; i < firstTypeCount; i++) {
            bots.add(newFirstBot());
        }
        for (int i = 0; i < secondTypeCount; i++) {
            bots.add(newSecondBot());
        }
        for (int i = 0; i < thirdTypeCount; i++) {
            bots.add(newThirdBot());
        }
    }

    protected abstract Bot newFirstBot();

    protected abstract Bot newSecondBot();

    protected abstract Bot newThirdBot();

}
```

Рис. 3.11. ThreeBotTypesRoundHandler

### 3.2.4. UI - UIManager, Diagram, DiagramElement, ProgressBar

Основні компоненти ядра ми уже подивились. Тепер час розглянути UI компоненти. Думаю варто почати з ProgressBar (Рис. 3.12.). Тут все доволі просто. ProgressBar - це клас, який наслідується від JProgressBar. Тобто по-суті це лише розширення уже існуючого функціоналу задля зручності. Тут ми маємо один публічний метод, який приймає загальну кількість раундів для тесту та номер поточного раунду. Метод називається updateProgress і все, що він робить, це оновлює прогрес бар. Цей метод ми уже бачили в RoundHandler'і. Там він викликається після кожного раунду. При досяганні максимального значення викличеться метод remove(), який видалить ProgressBar зі сцени.

```
6   public class ProgressBar extends JProgressBar {
7
8       public ProgressBar() {
9           super();
10
11          setStringPainted(true);
12          setForeground(Color.WHITE);
13          setBackground(Color.BLACK);
14          setBounds( x: 650,  y: 430,  width: 300,  height: 40);
15          setMinimum(0);
16          setMaximum(100);
17          setValue(0);
18          setString("Loading...");
19      }
20
21      public void updateProgress(int roundsCount, int currentRound) {
22          setValue((int) (((double) currentRound / (double) roundsCount) * 100));
23          repaint();
24          if(getValue() == 100) {
25              remove();
26          }
27      }
28
29      private void remove() {
30          setVisible(false);
31          getParent().remove( comp: this);
32      }
33  }
```

Рис. 3.12. ProgressBar

Далі розглянемо рекорд DiagramElement (Рис. 3.13.). Це звичайний java record, який містить в собі: String name, double value, Color color. Це є відображенням одного елементу діаграми.

```java
package com.ui.model;

import java.awt.*;

public record DiagramElement(String name, double value, Color color) {
}
```

Рис. 3.13. DiagramElement

Тепер подивимось де ж використовується цей рекорд. А саме будемо розглядати клас Diagram (рис. 3.14.). Тут є про що розказати, адже цей клас динамічно створює діаграми. Почнемо з того, що клас Diagram має декілька констант на рівні класу. Основні - це ELEMENT_WIDTH та MAX_HEIGHT. Ці цілочисельні константи вказують на те яких розмірів буде набувати кожен елемент діаграми. Ширина - завжди буде сталою і буде дорівнювати 200пікселів. Висота - буде різною (на те це і діаграма), адже кожен елемент має різні значення і повинен відображатись по-різному. Проте максимальною висотою вибрано - 400пікселів.

Тепер поглянемо на конструктор. Він вимагає список DiagramElemnt, які необхідно відображати. Також варто звернути увагу, що в якості LayoutManger'а використовується GridBagLayout. GridBagLayout є одним із менеджерів розташування (layout manager) у бібліотеці Java Swing. Він використовується для організації компонентів (віджетів) у вигляді сітки, де кожен компонент може займати різну кількість рядків та стовпців. GridBagLayout дозволяє створювати багатовимірні сітки та детально налаштовувати розташування кожного

компонента в цій сітці. Ключевим також є той момент, що ми вказуємо anchor=SOUTH для кожного елементу діаграми, щоб вони відображались знизу вверх.

Потім визначається максимальне значення з усіх переданих елементів. Цей елемент займе весь простір догори, тобто 400пікселів. Усі інші елементи будуть відштовхуватись від цієї позначки. Метод визначення висоти називається defineHeight() та приведений на рис. 3.14.

```
public class Diagram extends JPanel {

    private static final int ELEMENT_WIDTH = 120;
    private static final int MAX_HEIGHT = 400;

    private static final DecimalFormat DECIMAL_FORMAT = new DecimalFormat( pattern: "#,###.##");

    public Diagram(List<DiagramElement> elements) {
        this.setLayout(new GridBagLayout());
        GridBagConstraints gridBagConstraints = new GridBagConstraints();
        gridBagConstraints.anchor = GridBagConstraints.SOUTH;

        double max = elements.stream()
                .mapToDouble(DiagramElement::value)
                .max().orElse( other: 0d);
        elements.stream()
                .map(element -> createElement(element, max))
                .forEach(panel -> this.add(panel, gridBagConstraints));
    }

    private JPanel createElement(DiagramElement element, double max) {
        JPanel jPanel = new JPanel();
        jPanel.setLayout(new BorderLayout( hgap: 0,  vgap: 10));
        jPanel.setBackground(element.color());
        jPanel.setPreferredSize(new Dimension(ELEMENT_WIDTH, defineHeight(element.value(), max)));

        jPanel.add(getNameLabel(element.name()), BorderLayout.SOUTH);
        jPanel.add(getValueLabel(element.value()), BorderLayout.CENTER);

        return jPanel;
    }

    private int defineHeight(double value, double max) {
        double percent = (100 * value) / max;
        return (int) ((percent * MAX_HEIGHT) / 100d);
    }

    private JLabel getNameLabel(String text) { return getLabel(text, Color.BLACK); }

    private JLabel getValueLabel(double value) { return getLabel(formatValue(value), Color.WHITE); }

    private JLabel getLabel(String text, Color color) {...}

    private String formatValue(double value) {...}
}
```

Рис. 3.14. Diagram

Також є невеличкий клас - StatsPanel. Цей клас відповідає за панель для діаграми  та її назви (рис. 3.15).

```
public class StatsPanel extends JPanel {
    private static final int NAME_HEIGHT = 30;

    private final Diagram diagram;

    public StatsPanel(String name, Diagram diagram) {
        this.diagram = diagram;

        this.setBackground(new Color( r: 243,  g: 243,  b: 243));
        this.setLayout(new BorderLayout( hgap: 0,  vgap: 0));
        this.add(getNameLabel(name), BorderLayout.NORTH);
        this.add(diagram, BorderLayout.CENTER);
    }

    private JLabel getNameLabel(String name) {
        JLabel nameLabel = new JLabel(name);
        nameLabel.setHorizontalAlignment(JLabel.CENTER);
        nameLabel.setPreferredSize(new Dimension(diagram.getWidth(), NAME_HEIGHT));
        return nameLabel;
    }
}
```

Рис. 3.15. StatsPanel

Ну і накінець головний ui клас - UiManger (рис. 3.16.). Цей клас відповідає за відображення сцен. Він вміє показувати сцену з ProgressBar’ом та сцену з діаграмами. За це відповідають методи startUiApplication (показує сцену з прогрес баром) та showResults (показує діаграми).

Варто трошки заглибитись у те, як працює саме showResults. Всі компоненти знаходяться на mainPanel, яка являється простим розширенням JPanel. Під час виконання showResults, ми задаємо layoutManager=FflowLayout. Центруємо всі елементи по горизонталі та задаємо відступи по 30пікселів по горизонталі та по вертикалі. Після чого просто поміщаємо StatsPanel’і за

допомогою методу JPanel.add(). І обов’язково не забуваємо перевалідувати і перемалювати mainPanel, адже ми змінили їй layout та розмістили нові компоненти.

```java
public class UiManager {

    @Getter
    private final ProgressBar progressBar;
    private final MainPanel mainPanel;

    private final Map<Class<? extends Bot>, Color> botTypeColors = Map.of(
            EgoistBot.class, Color.RED,
            DummyBot.class, Color.YELLOW,
            AltruistBot.class, Color.GREEN,
            AnalyticBot.class, new Color( r: 255,  g: 167,  b: 0),
            AnalyticV2Bot.class, new Color( r: 255,  g: 128,  b: 0)
    );

    public UiManager() {
        progressBar = new ProgressBar();
        mainPanel = new MainPanel(progressBar);
    }

    public void startUiApplication() { new MainWindow(mainPanel); }

    public void showResults(Results.Report report) {
        mainPanel.setLayout(new FlowLayout(FlowLayout.CENTER,  hgap: 30,  vgap: 30));
        mainPanel.setBackground(Color.WHITE);

        mainPanel.add(new StatsPanel( name: "WINS:", new Diagram(getWinsAggByTypeDiagramElements(report))));
        mainPanel.add(new StatsPanel( name: "AVERAGE WINS:", new Diagram(getAvgWinsByTypeDiagramElements(report))));
        mainPanel.add(new StatsPanel( name: "TOTAL WIN COUNT:", new Diagram(getTotalWinsByTypeDiagramElements(report))));

        mainPanel.revalidate();
        mainPanel.repaint();
    }

    private List<DiagramElement> getWinsAggByTypeDiagramElements(Results.Report report) {...}

    private List<DiagramElement> getAvgWinsByTypeDiagramElements(Results.Report report) {...}

    private List<DiagramElement> getTotalWinsByTypeDiagramElements(Results.Report report) {...}
}
```

Рис. 3.16. UiManager

### 3.3. Імплементації

Основну логіку ми уже розглянули. Залишається розглянути деякі імплементації. На рис. 3.17. можна побачити всі імплементації, які є в проекті. Проте, розглянемо деякі з них детально. Особливу увагу приділяючи саме реалізації ботів.

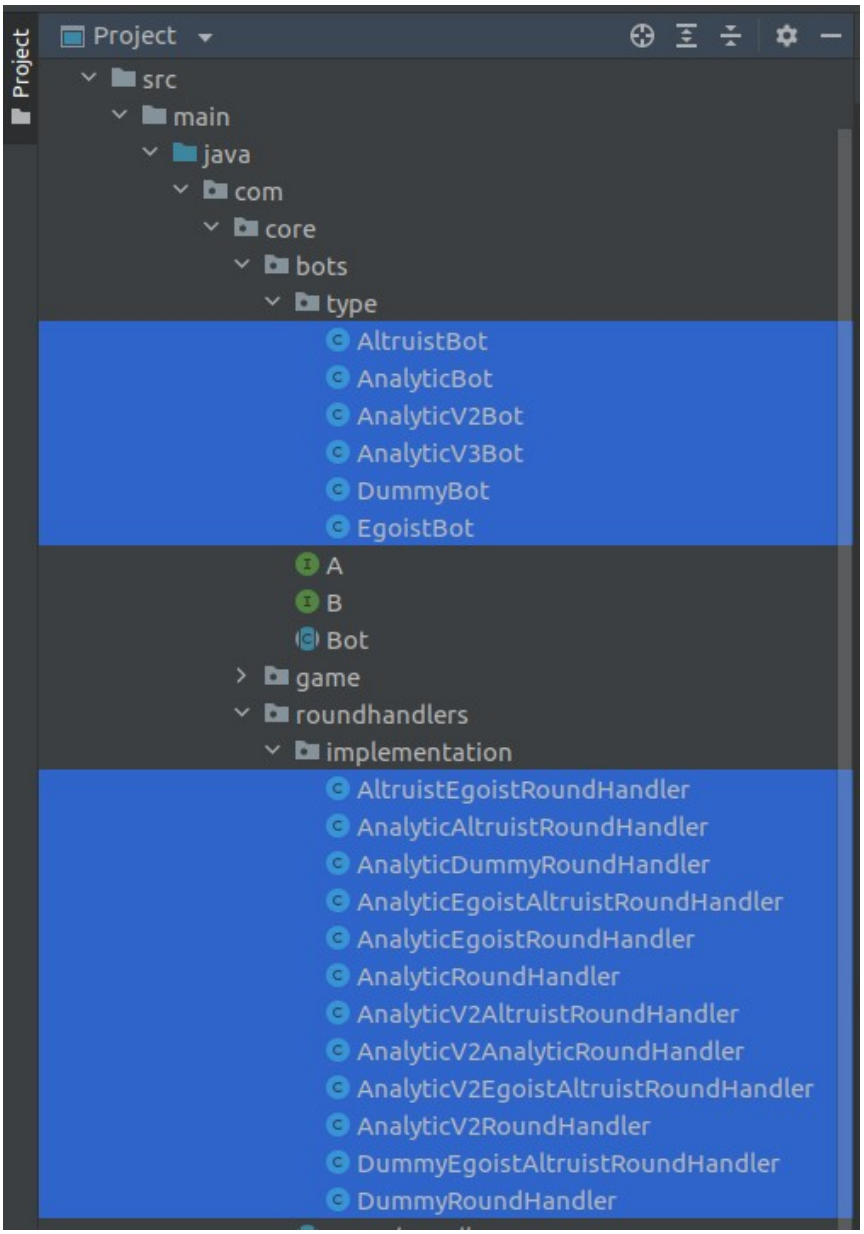



Рис. 3.17. Імплементації

### 3.3.1. Імплементація Ботів

Загалом ми маємо 6 видів ботів - це DummyBot, Egoist, Altruist та 3 типи Analytic'ів. Найпростіший вид - це DummyBot (рис. 3.18). На ньому можна розглянути як взагалі виглядає імплементація боту. Цей бот, як і будь-який інший, розширює клас Bot. Як ми пам'ятаємо, клас Bot - це реалізація інтерфейсів A та B. Тож нам необхідно імплементувати їхні методи. А саме: offer(), decide(). DummyBot робить доволі дивні пропозиції, а погоджується як середньостатистичний гравець в "Ультиматум". Проте це радше навчальний або показовий бот, ніж справжня реалізація.

Давайте розглянемо метод offer(). Цей метод використовує утильний клас Randomizer, щоб отримати рандомне значення від 50 до 90. Таким чином він формує свій відсоток. А решта (100 - aPrecentage) - буде відсотком ділення для противника.

Тепер метод decide(). Тут нам приходить пропозиція (Offer). З неї ми дістаємо відсоток B, тобто наш відсоток. І потім уже визначаємо рандомну НИЖНЮ границю прийняття. Ця границя коливається від 20% до 30%. Після чого порівнюємо наш відсоток з нижньою границею, якщо приймаємо - повернемо true, інакше - false.

```
7   public class DummyBot extends Bot {
8
9       @Override
10      public Offer offer() {
11          int aPercentage = Randomizer.randomBounds( lowerBounds: 50,  upperBounds: 90);
12          return new Offer(aPercentage,  bPercentage: 100 - aPercentage);
13      }
14
15      @Override
16      public boolean decide(Offer offer) {
17          return offer.bPercentage() >= Randomizer.randomBounds( lowerBounds: 20,  upperBounds: 30);
18      }
19  }
```

Рис. 3.18. DummyBot

Іншим, уже цікавішим ботом можна представити EgoistBot'а (рис. 3.19.). Особливість цього бота в тому, що він поводиться як егоїст. Тобто коли грає в якості гравця А - вибирає собі якомога більший відсоток. А коли грає за гравця В - приймає лише близькі до 50% ділення. Всі інші він вважатиме не чесними. Більш детально можна роздивитись на рис. 3.19.

```
7   public class EgoistBot extends Bot {
8
9       @Override
10      public Offer offer() {
11          int aPercentage = Randomizer.randomBounds( lowerBounds: 60,  upperBounds: 90);
12          return new Offer(aPercentage,  bPercentage: 100 - aPercentage);
13      }
14
15      @Override
16      public boolean decide(Offer offer) {
17          int acceptance = Randomizer.randomBounds( lowerBounds: 30,  upperBounds: 50);
18          return offer.bPercentage() >= acceptance;
19      }
20  }
```

Рис. 3.19. EgoistBot

На противагу егоїстам, завжди знайдуться і альтруїсти. Їх представлено у класі AltruistBot (рис. 3.20.). Вони поводяться як справжні альтруїсти. Тобто ці боти пропонують більший відсоток супротивнику, інколи рівні відсотки. А приймають як середньостатистичні гравці. Тобто ніколи не нижче 20%.

```java
public class AltruistBot extends Bot {

    @Override
    public Offer offer() {
        int aPercentage = Randomizer.randomBounds( lowerBounds: 20,  upperBounds: 50);
        return new Offer(aPercentage,  bPercentage: 100 - aPercentage);
    }

    @Override
    public boolean decide(Offer offer) {
        int acceptance = Randomizer.randomBounds( lowerBounds: 20,  upperBounds: 30);
        return offer.bPercentage() >= acceptance;
    }
}
```

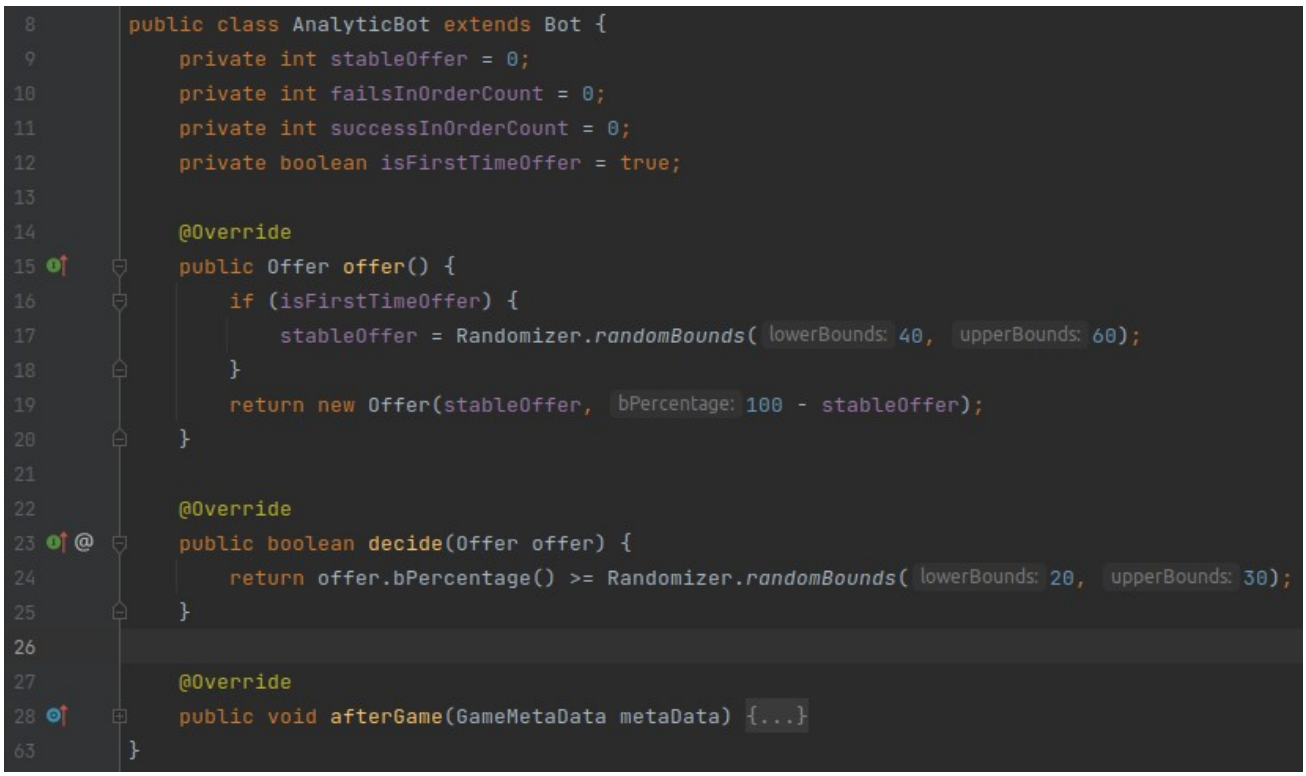
Рис. 3.20. AltruistBot

А ось де стає дійсно цікаво, так це аналітики. Вони представлені класом AnalyticBot (рис. 3.21.). Вся справа в тому, що вони аналізують свої ігри у випадку коли вони грали за A. Це зроблено для того, аби вибрати найбільш оптимальну стратегію. Загалом алгоритм наступний:

- бот пропонує ставку і робить її сталою (тобто пропонує її і надалі)
- фналізує чи була вона прийнята
- якщо ставка прийнята - бот запам’ятовує це
- якщо ставку відхилили - бот також запам’ятовує це
- коли бот розуміє, що його ставку занадто багато разів підряд відхиляють - він понижає сталу ставку.
- коли бот розуміє, що його ставку деяку кількість разів підряд приймають - він підвищує сталу ставку.

Тобто цей бот наврядчи зможе заробити велику суму при маленькій кількості раундів. Проте в теорії на безкінечно великій кількості раундів цей бот може заробити суму наближеною до найбільш ймоіврної. Звичайно, все залежить від супротивника. Адже коли аналітик грає з егоїстами, а потім аналітик грає з

альтруїстами, то це різні рівні прийняття. Проте аналітик якраз нівелює цю різницю, адже аналізує кожного, а не просто ставить ставки опираючись на свій характер.



```
public class AnalyticBot extends Bot {
    private int stableOffer = 0;
    private int failsInOrderCount = 0;
    private int successInOrderCount = 0;
    private boolean isFirstTimeOffer = true;

    @Override
    public Offer offer() {
        if (isFirstTimeOffer) {
            stableOffer = Randomizer.randomBounds( lowerBounds: 40,  upperBounds: 60);
        }
        return new Offer(stableOffer,  bPercentage: 100 - stableOffer);
    }

    @Override
    public boolean decide(Offer offer) {
        return offer.bPercentage() >= Randomizer.randomBounds( lowerBounds: 20,  upperBounds: 30);
    }

    @Override
    public void afterGame(GameMetaData metaData) {...}
}
```

Рис. 3.21. AnalyticBot

Насправді AnalyticBot досить великий і не поміщається весь на рисунок, тому розберемо його на двох окремих. На рис. 3.21. Ми бачимо загальну поведінку цього боту. Коли він пропонує, то спершу перевіряє чи це перша його пропозиція. Якщо так - то пропонує рандомно від 40 до 60%. При цьому записує в stableOffer цю пропозицію. Тепер вона вважається стабільною. Потім бот просто пропонує офер з цією stableOffer пропозицією.

Вся “кухня” відбувається саме в методі afterGame(), про який ми уже згадували раніше, коли розглядали клас Bot. Давайте подивимось на рис. 3.22.

На цьому рисунку, ми бачимо реалізацію afterGame методу. Спершу ми дізнаємось чи ми взагалі робили пропозицію. Тобто чи грали ми за гравця А. Якщо ж ні, то ми виходимо з методу. Інакше підемо далі. Якщо нашу ставку все-ж прийняли - ми збільшуємо рахунок успішних угод та скидаємо рахунок зафейлених угод. Якщо ставку не прийнто, тоді робимо все навпаки. Скидаємо рахунок успішних угод та збільшуємо рахунок не прийнтих угод.

Далі за умови, що рахунок фейлів >= 2, ми скинемо цей рахунок в нуль та понизимо ставку рандомно від 1 до 10%. Якщо ж у нас було 3 поспіль успішні згоди, тоді ми підвищимо ставку теж на від одного до десяти відсотків.

Саме так працює AnalyticBot. Проте проаналізувавши різні тести, можна зрозуміти, що це не найоптимальніша стратегія, тому почав експериментувати і з іншими. Можна спробувати змінювати різні значення для підвищення та для пониження стабільної ставки. Таким чином можна знайти дещо кращу пропорцію, яка показує кращі результати в різних тестах. Цей бот названо - AnalyticV2Bot. Аналітик другої версії.

Він нічим іншим не відрізняється від AnalyticBot’а окрім як інших порогів пониження та підвищення ставок. Тому не бачу сенсу розглядати його цілком і повністю. Хоча можна поглянути на метод afterGame(GameMetaData). Його можна знайти на рис 3.23.

```java
@Override
public void afterGame(GameMetaData metaData) {
    boolean wasIOffering = wasIOffering(metaData);
    if (!wasIOffering) {
        return;
    }

    isFirstTimeOffer = false;
    if (metaData.isAccepted()) {
        successInOrderCount++;
        failsInOrderCount = 0;
    } else {
        failsInOrderCount++;
        successInOrderCount = 0;
    }

    boolean shouldIncrease = false;
    boolean shouldDecrease = false;
    if (failsInOrderCount >= 2) {
        shouldDecrease = true;
        failsInOrderCount = 0;
    }
    if (successInOrderCount >= 3) {
        shouldIncrease = true;
        successInOrderCount = 0;
    }

    if (shouldIncrease) {
        int increase = Randomizer.randomBounds( lowerBounds: 1,  upperBounds: 10);
        stableOffer = Math.min(100, stableOffer + increase);
    }
    if (shouldDecrease) {
        int decrease = Randomizer.randomBounds( lowerBounds: 1,  upperBounds: 10);
        stableOffer = Math.max(0, stableOffer - decrease);
    }
}
```

Рис. 3.22. AnalyticBot.afterGame()

```java
@Override
public void afterGame(GameMetaData metaData) {
    boolean wasIOffering = wasIOffering(metaData);
    if (!wasIOffering) {
        return;
    }

    isFirstTimeOffer = false;
    if (metaData.isAccepted()) {
        successInOrderCount++;
        failsInOrderCount = 0;
    } else {
        failsInOrderCount++;
        successInOrderCount = 0;
    }

    boolean shouldIncrease = false;
    boolean shouldDecrease = false;
    if (failsInOrderCount >= 1) {
        shouldDecrease = true;
        failsInOrderCount = 0;
    }
    if (successInOrderCount >= 5) {
        shouldIncrease = true;
        successInOrderCount = 0;
    }

    if (shouldIncrease) {
        int increase = Randomizer.randomBounds( lowerBounds: 1,  upperBounds: 10);
        stableOffer = Math.min(100, stableOffer + increase);
    }
    if (shouldDecrease) {
        int decrease = Randomizer.randomBounds( lowerBounds: 1,  upperBounds: 10);
        stableOffer = Math.max(0, stableOffer - decrease);
    }
}
```

Рис. 3.23. AnalyticV2Bot.afterGame()

### 3.3.2. Імплементації RoundHandler'ів

Тут не варто зупинятись на кожній окремій імплементації, адже вони між собою дуже схожі. Тож розберемо лише основне.

Як уже було сказано, у нас є основний клас RoundHandler, який використовується для створення та прогону життєвого циклу раундів. Всі тести, які включають в себе створення лише одного бот тайпу, можуть наслідувати безпосередньо клас RoundHandler та реалізовувати метод setup() таким чином, щоб створювати в циклі необхідних ботів.

Для більш складних тестів, де є 2 або 3 бот тайпи, необхідно реалізовувати або TwoBotTypesRoundHandler, або ThreeBotTypesRoundHandler. Деякі приклади таких хендлерів можна знайти на рис. 3.24. та рис. 3.25.

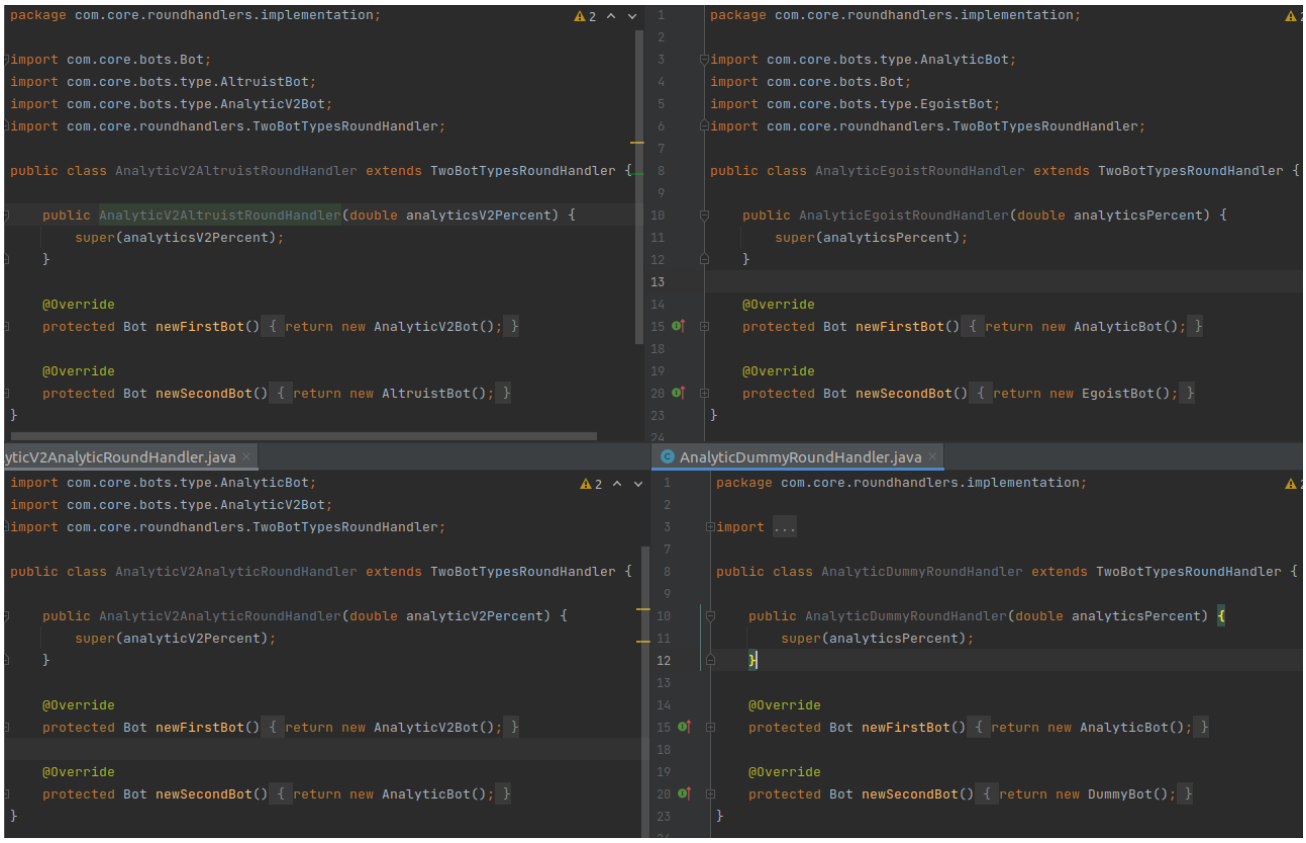



Рис. 3.24. TwoBotTypesRoundHandler імплементації

AnalyticEgoistAltruistRoundHandler.java

```java
import com.core.roundhandlers.ThreeBotTypesRoundHandler;

public class AnalyticEgoistAltruistRoundHandler extends ThreeBotTypesRoundHandler {

    public AnalyticEgoistAltruistRoundHandler(double analyticsPercent, double egoistsPercent, double altruistsPercent) {
        super(analyticsPercent, egoistsPercent, altruistsPercent);
    }

    @Override
    protected Bot newFirstBot() { return new AnalyticBot(); }

    @Override
    protected Bot newSecondBot() { return new EgoistBot(); }

    @Override
    protected Bot newThirdBot() { return new AltruistBot(); }
}
```

DummyEgoistAltruistRoundHandler.java

```java
import com.core.roundhandlers.ThreeBotTypesRoundHandler;

public class DummyEgoistAltruistRoundHandler extends ThreeBotTypesRoundHandler {

    public DummyEgoistAltruistRoundHandler(double dummiesPercent, double egoistsPercent, double altruistsPercent) {
        super(dummiesPercent, egoistsPercent, altruistsPercent);
    }

    @Override
    protected Bot newFirstBot() { return new DummyBot(); }

    @Override
    protected Bot newSecondBot() { return new EgoistBot(); }

    @Override
    protected Bot newThirdBot() { return new AltruistBot(); }
}
```

Рис. 3.25. ThreeBotTypesRoundHandler імплементації

Це звичайно не всі імплементації RoundHandler’ів. Наведені вище імплементації це скоріше просто приклад для розуміння того, як це виглядає. Немає сенсу показувати більше так як з назви завжди зрозуміло що це за RoundHandler. А реазізація завжди практично однакова.

## ВИСНОВКИ ДО РОЗДІЛУ 3

В цьому розділі було показано та пояснено як спроектована ботоферма, які вона моє основні класи та підходи. Було ретельно досліджено та представлено процес створення ботоферми, спеціально призначеної для використання в грі "Ультиматум". Основним завданням цього етапу дослідження було розроблення оптимального архітектурного рішення, яке враховує усі технічні, функціональні та ігрові вимоги.

У рамках проектування ботоферми було визначено ключові класи та структурні компоненти, які гармонійно взаємодіють для досягнення максимальної ефективності та продуктивності. Розроблена ботоферма є не лише ефективним інструментом для гри "Ультиматум", але й відкриває перспективи для подальших досліджень, що ми і дослідимо у наступному розділі.

# РОЗДІЛ 4
# ТЕСТУВАННЯ БОТОФЕРМИ

Настав час протестувати той код, що у нас вийшов і вийзначити яка стратегія буде найкращою. Як будуть себе поводити різні стратегії в залежності від інших ботів та їхньої кількості.

## 4.1. DummyBot тест

Першим протестуємо самого звичайного бота - тестового бота DummyBot. Конфігурація проста:

- Грають 10,000 ботів
- Кількість раундів 1,000 при ставці 100
- Всі боти типу DummyBot

Поглянемо на результати (рис. 4.1.)

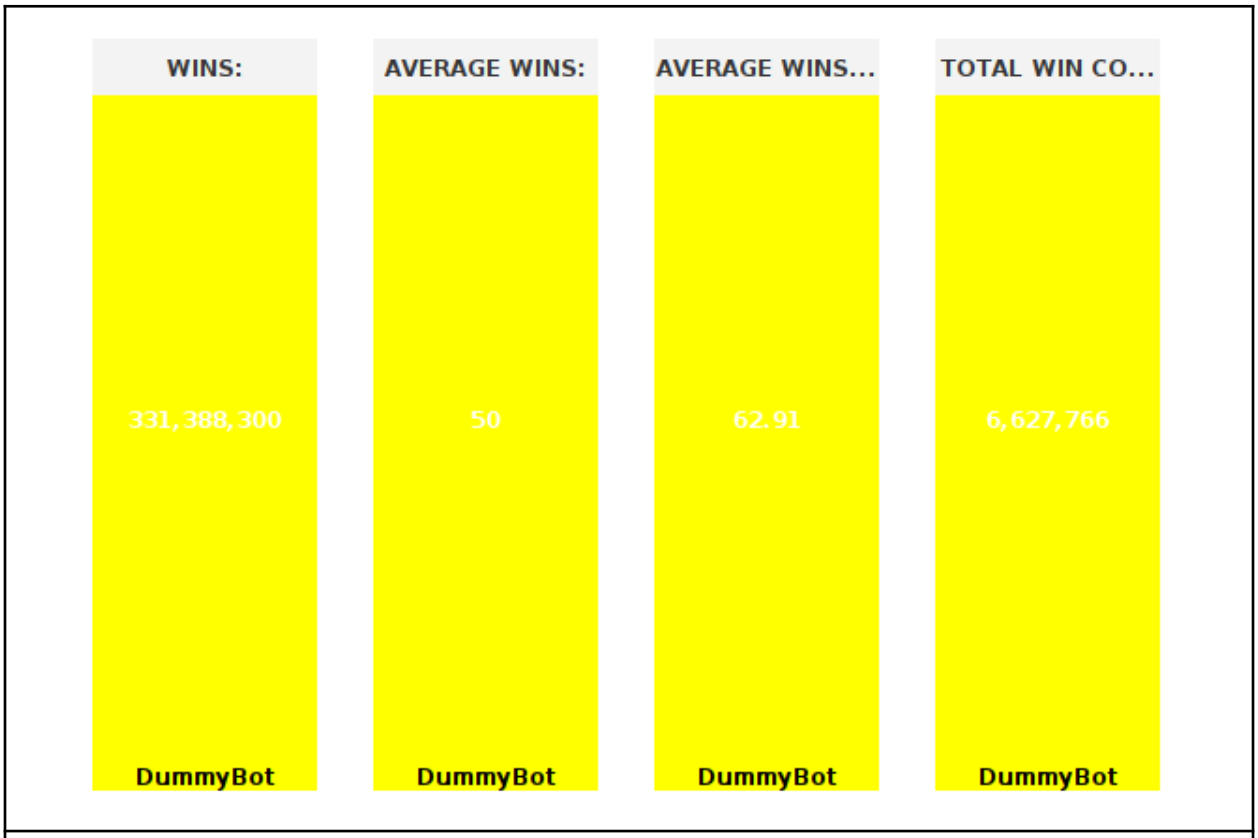



Рис. 4.1. DummyBot тест

Тут ми бачимо 4 діаграми. Перша діаграма показує загальну кількість вигранних грошей для всіх DummyBot. Це 331,388,300 - що доволі непоганий результат > 50%.

Також друга діаграма показує, що в середньому DummyBot'и вигравали зі ставкою 50. Хоча це є логічно, адже в цій діаграмі враховуються як виграші A, так і виграші B. Так як всі боти - DummyBot - середній виграш - 50. Ну от наприклад грають 2 DummyBot. Розподіл: 80/20%. Бот B приймає ставку, тоді ми запишемо, що бот1 виграв 80, а бот2 - виграв 20. Середнє значення - 50.

Третя діаграма показує лише середні виграші категорії ботів коли вони грали за A. Це значення дещо вище за середні виграші пораховані як за виграші за A так і B.

Четверта діаграма показує кількість загальних перемог. Тобто DummyBot загально виграли більше шести з половиною мільйонів разів. Давайте поглянемо на рис. 4.2. І розберемо детальніше даний тест.

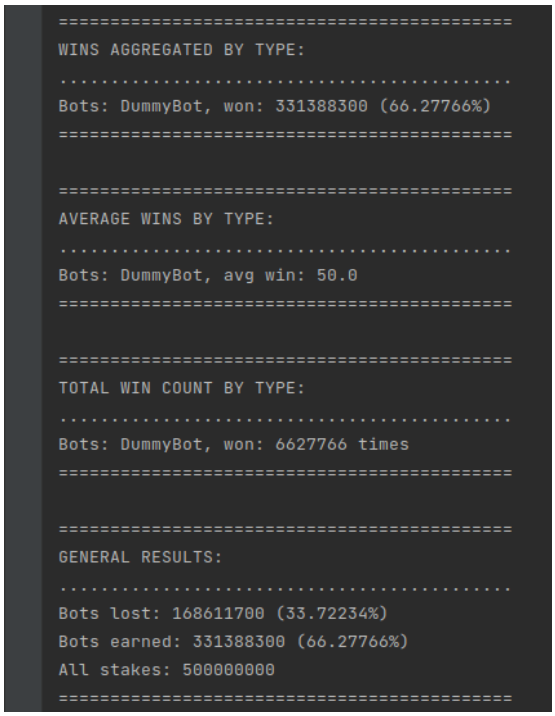



Рис. 4.2. DummyBot тест (консоль)

Тут загалом ті самі данні окрім того, що можна подивитись на відсотки та загальну статистику (General Results). Звідси можна зробити висновки, що DummyBot'и граючи самі з собою змогли заробити 66.2% від загального капіталу. В той час як втратили вони більше півтора мільйона, що приблизно = 33.7%.

Цей тест є більше показовим ніж реальним, адже DummyBot'и це тестові боти які слугують лише для тестування функціоналу і ставлять досить рандомні ставки.

Хочеться додати, що наприкінці всіх тестів буде приведено таблицю зі всіма тестами, щоб можна було проаналізувати всі стратегії в одному, зручному місці.

### 4.2. EgoistBot тест

Другим протестуємо EgoistBot'а, який уособлює собою поведінку середньостатистичного егоїста. Як ми можемо пам'ятати з попереднього розділу, цей бот приймає лише вигідні йому пропозиції в більшості схильні до поділу навпіл. А пропонує він лише вигідні собі пропозиції, які часто досягають значень 90%.

Очікується, що ці боти зароблять зовсім невелику суму грошей, адже це егоїсти, які грають з егоїстами і їм буде важко дійти згоди. Хоча всі результати згодом. Поглянемо на результати. рис. 4.3.

Ось конфігурація цього тесту:

- Грають 10,000 ботів
- Кількість раундів 1,000 при ставці 100
- Всі боти типу EgoistBot

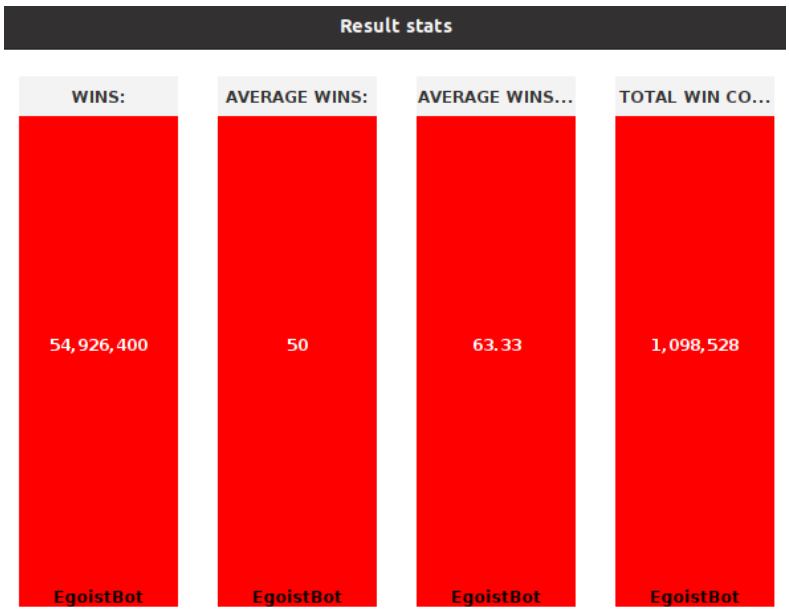



Рис. 4.3. EgoistBot тест

Знову бачимо 4 діаграми. Вони статичні і не будуть змінювати свою суть в залежності від тестів. Змінюватимуться лише значення.

Перша діаграма показує загальну кількість виграних грошей для всіх EgoistBot. Це 54,926,400- що є дуже низьким результатом і знаходиться в межах 10%

Також друга діаграма показує, що в середньому EgoistBot'и вигравали зі ставкою 50. Ситуація така ж сама як і в попередньому тесті з DummyBot'ами. Вона буде такою щоразу, коли гратимуть лише один тип ботів, тому не будемо зупинятись на цьому в наступних тестах.

Третя діаграма показує лише середні виграші категорії ботів коли вони грали за A. Тут ми бачимо егоїзм ботів, адже це значення більше за половину і дорівнює 63.33.

Четверта діаграма показує кількість загальних перемог. Тобто EgoistBot'и загально виграли трошки більше мільйона разів. Для детальнішої статистики варто подивитись що ж нам написало в консоль. Давайте поглянемо на рис. 4.4. І розберемо детальніше цей тест.

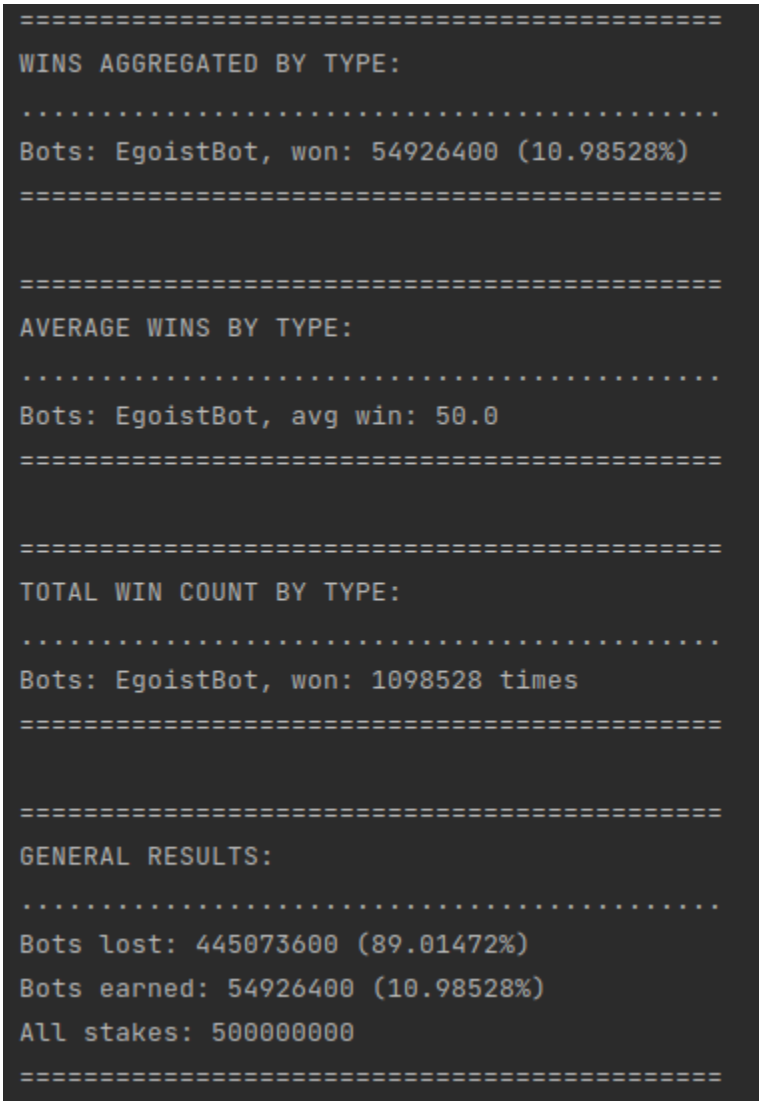



Рис. 4.4. EgoistBot тест (консоль)

Тут ми бачимо, що егоїсти втратили доволі велику суму порівняно з тестовим ботом. Та і загалом їм не вдалось заробити багато грошей. Навіть середні виграші за А були на рівні з DummyBot’ами.

Насправді ж егоїсти у грі з егоїстами ніяк не можуть зійтись в пропозиціях. Це наглядно видно по вище проведеному тестуванню. Проте вони беруть своє у інших тестах. Наприклад тестах з альтруїстами. Такий тест ми обов’язково проведемо. Проте спочатку розглянемо альтруїстів окремо.

### 4.3. AltruistBot тест

Третім протестуємо AltruistBot'а, який уособлює собою поведінку звичайного альтруїста. Як ми пам'ятаємо з попереднього розділу, цей бот має звичайний поріз прийняття, як і більшість ботів. А пропонує він в більшості не вигідні собі пропозиції, тобто менше 50% дял себе та більше для опонента.

Очікується, що ці боти зароблять найбільше грошей, адже всі пропозиції повинні прийматись.

Ось конфігурація цього тесту:

- Грають 10,000 ботів
- Кількість раундів 1,000 при ставці 100
- Всі боти типу AltruistBot

Поглянемо на результати. рис. 4.5.

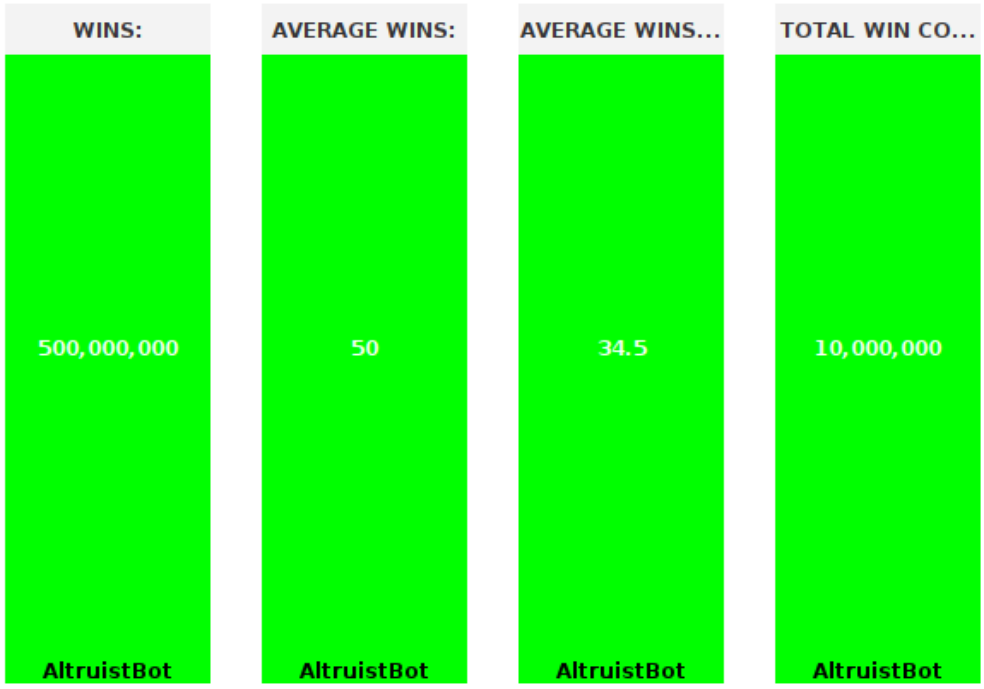



Рис. 4.5. AltruistBot тест

Перша діаграма показує загальну кількість вигранних грошей для всіх AltruistBot’ів. Це 500,000,000 - що є повною сумою всіх ставок. Тобто ці боти виграли абсолютно всі гроші, які були задіяні в тестуванні.

Також друга діаграма знову показує, що в середньому AltruistBot’и вигравали зі ставкою 50.

Третя діаграма показує лише середні виграші категорії ботів коли вони грали за A. І тут проявляється справжній альтруїзм ботів. 34.5% - середній розподіл при грі за A.

Четверта діаграма показує кількість загальних перемог. Тобто AltruistBot’и загально виграли всі можливі рази. Для детальнішої статистики варто подивитись що ж нам написало в консоль. рис. 4.6.

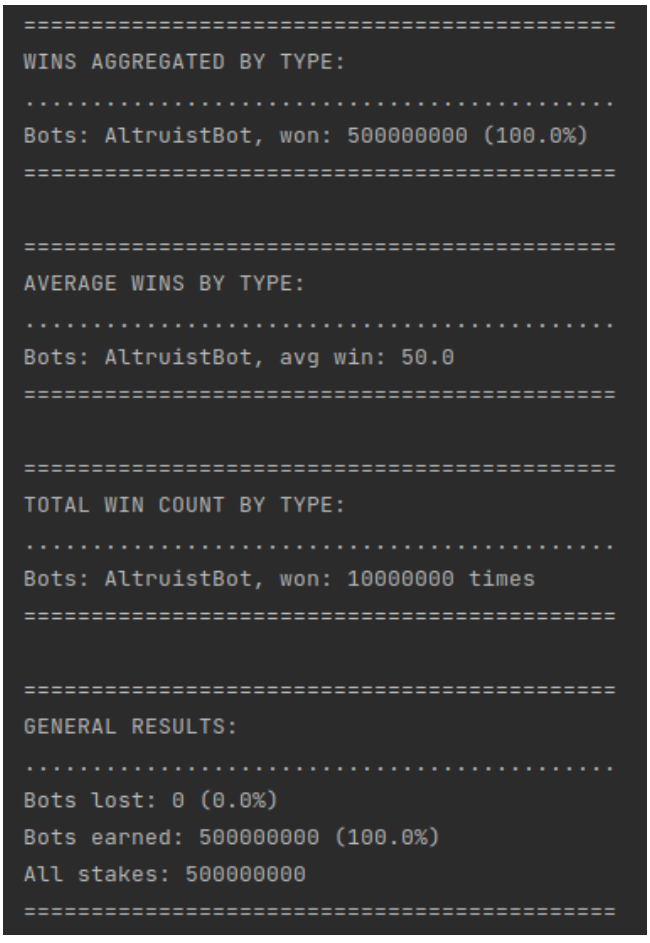



Рис. 4.6. AltruistBot тест (консоль)

Всі числа максимальні, адже не було жодного програшу. Якщо подивитись на Bots lost - то можна побачити значення 0 (0.0%).

Є ще дещо, що справді варте уваги в цьому тесті. В консоль виводяться також дані про виграні гроші для кожного бота окремо. Ці данні відсортовані за зростанням. І якщо поглянути на першого та останнього ботів з цього списку, то ми побачимо наступне:

Bot: AltruistBot, earned: 47712.0

Bot: AltruistBot, earned: 51874.0

Як бачите відрив дуже невеликий і можна зробити висновки, що стратегі альтруїзму - тобто допомоги іншим в убиток собі - працює і кожен заробляє приблизно однакові суми грошей.

Але як щодо егоїстів? Давайте поглянемо їхній стан найбіднішого та найбагатшого ботів:

Bot: EgoistBot, earned: 3438.0

Bot: EgoistBot, earned: 7538.0

Різниця відчутна. Якщо для альтруїстів різниця у відсотку склала близько 8%, то для егоїстів близько 119%.

### 4.4. AnalyticBot тест

Тепер давайте проведемо тест для AnalyticBot’a, який дійсно аналізує свої перемоги та невдачі. Очікується, що середнє значення пропозицій для цього бота повинно бути близько 70%, адже поріг відхилення у цього бота стандартний - від 20 до 30%. Тобто мінімальне середнє значення пропозицій повинно бути 70%, а максимальне 80%.

Очікується, що ці боти зароблять багато грошей, адже вони гратимуть багато раундів аналізуючи свої помилки на вдачі і підбираючи найоптимальніші стратегії.

Ось конфігурація цього тесту:

- Грають 10,000 ботів
- Кількість раундів 1,000 при ставці 100
- Всі боти типу AnalyticBot

Поглянемо на результати (рис. 4.7).

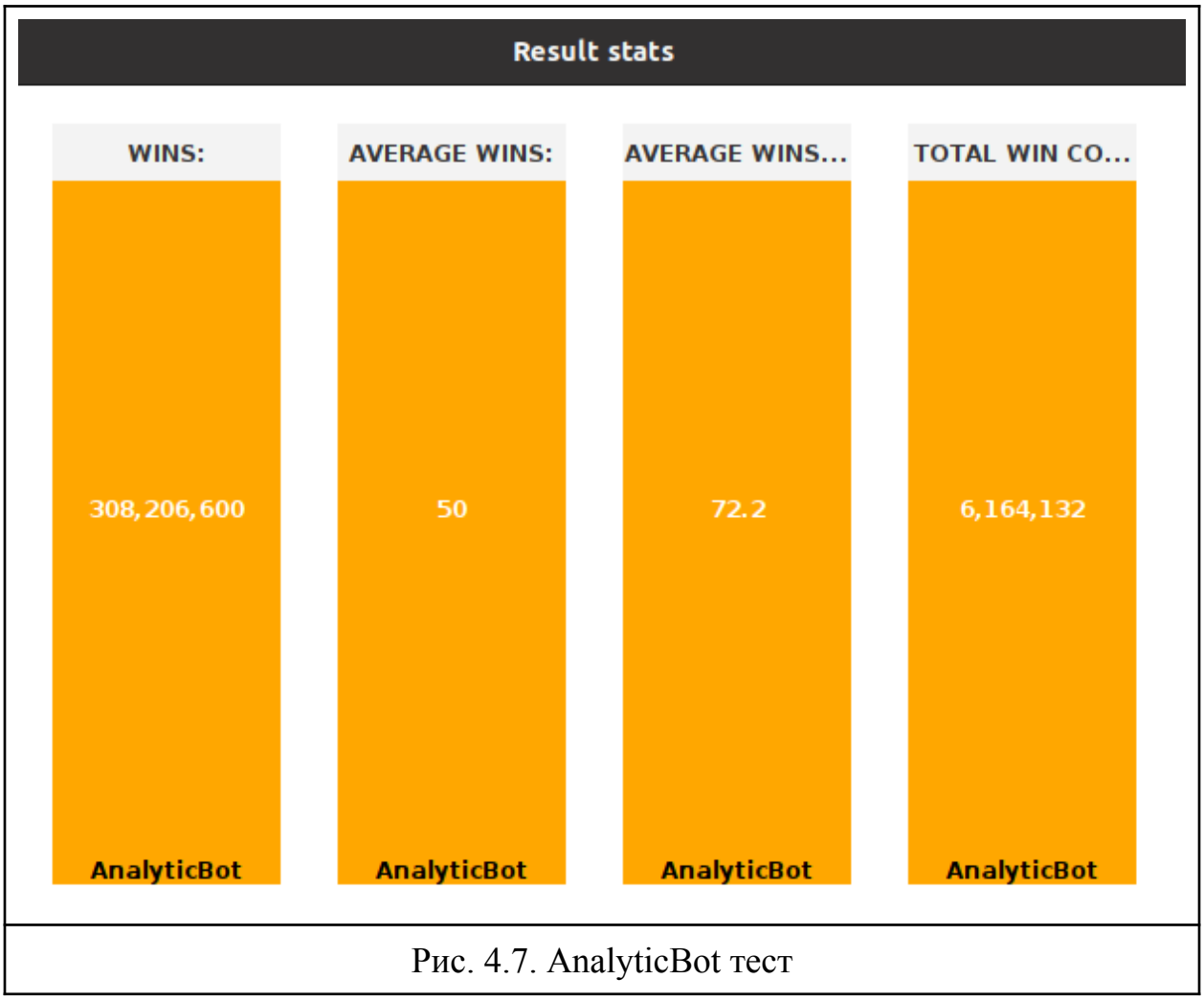



Рис. 4.7. AnalyticBot тест

Перша діаграма показує загальну кількість виграних грошей для всіх AnalyticBot’ів. 308,206,600 - тобто ці боти виграли більшу половину від загальної суми.

Також друга діаграма знову показує, що в середньому AnalyticBot’и вигравали зі ставкою 50.

Третя діаграма показує лише середні виграші категорії ботів коли вони грали за А. І тут ми бачимо те, що прогноз по середнім виграшам при грі за А справдився. 72.2%.

Четверта діаграма показує кількість загальних перемог. Тобто AnalyticBot’и загально виграли трохи більше 6ти мільйонів разів. І це доволі хороший результат. Результат - більше половини.

Для детальнішої статистики варто подивитись що ж нам написало в консоль.

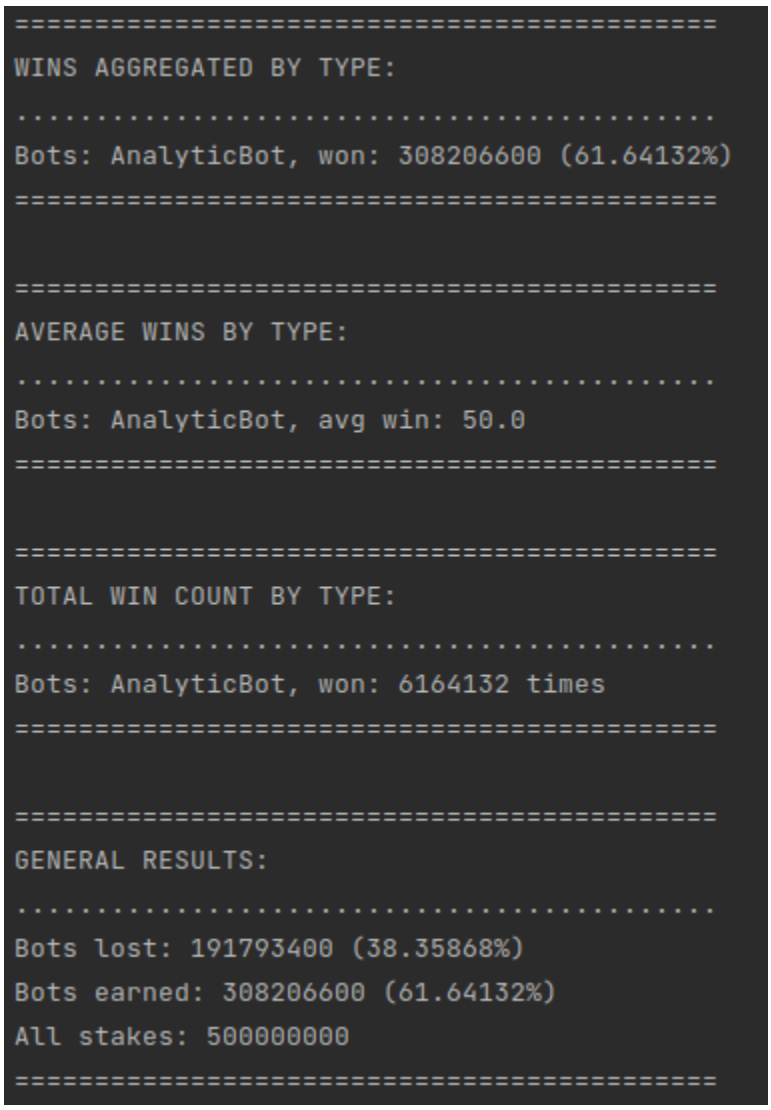



Рис. 4.8. AnalyticBot тест (консоль)

61% виграних грошей, що на теперішній момент є третім за кількістю виграшів. Лідирують альтруїсти, за ними ідуть dummy боти з 66% загального виграшу. І невже dummy боти зі своєю рандомною стратегією краще заробляють гроші ніж аналітики? Не зовсім. Справа в тому, що рандомна стратегія дійсно працює непогано, але лише для тесту, в якому всі гравці - ставлять рандомно. Нас же більше цікавлять ситуації, коли різні боти грають один з одним.

Давайте також проаналізуємо різницю найуспішніших та найменшуспішних ботів для dummy та аналітиків:

Bot: AnalyticBot, earned: 28220.0

Bot: AnalyticBot, earned: 33496.0

Bot: DummyBot, earned: 30218.0

Bot: DummyBot, earned: 36283.0

Різниця у аналітиків несуттєво менша ніж у тестових ботів

Загалом, впевнений, що якщо провести тест DummyBot проти AnalyticBot - аналітики виграють. Такий тест звичайно буде згодом.

### 4.5. AnalyticV2Bot

Що ж, як ми пам’ятаємо з попереднього розділу, аналітик другої версії в теорії повинен мати кращі результати аніж аналітик першої версії. Давайте проведемо тест, де аналітики другої версії будуть грати самі з собою і дослідимо чи дійсно вони будуть краще знаходити вигідні пропозиції.

Ось конфігурація цього тесту:

- Грають 10,000 ботів
- Кількість раундів 1,000 при ставці 100
- Всі боти типу AnalyticV2Bot

Поглянемо на результати. рис. 4.9.

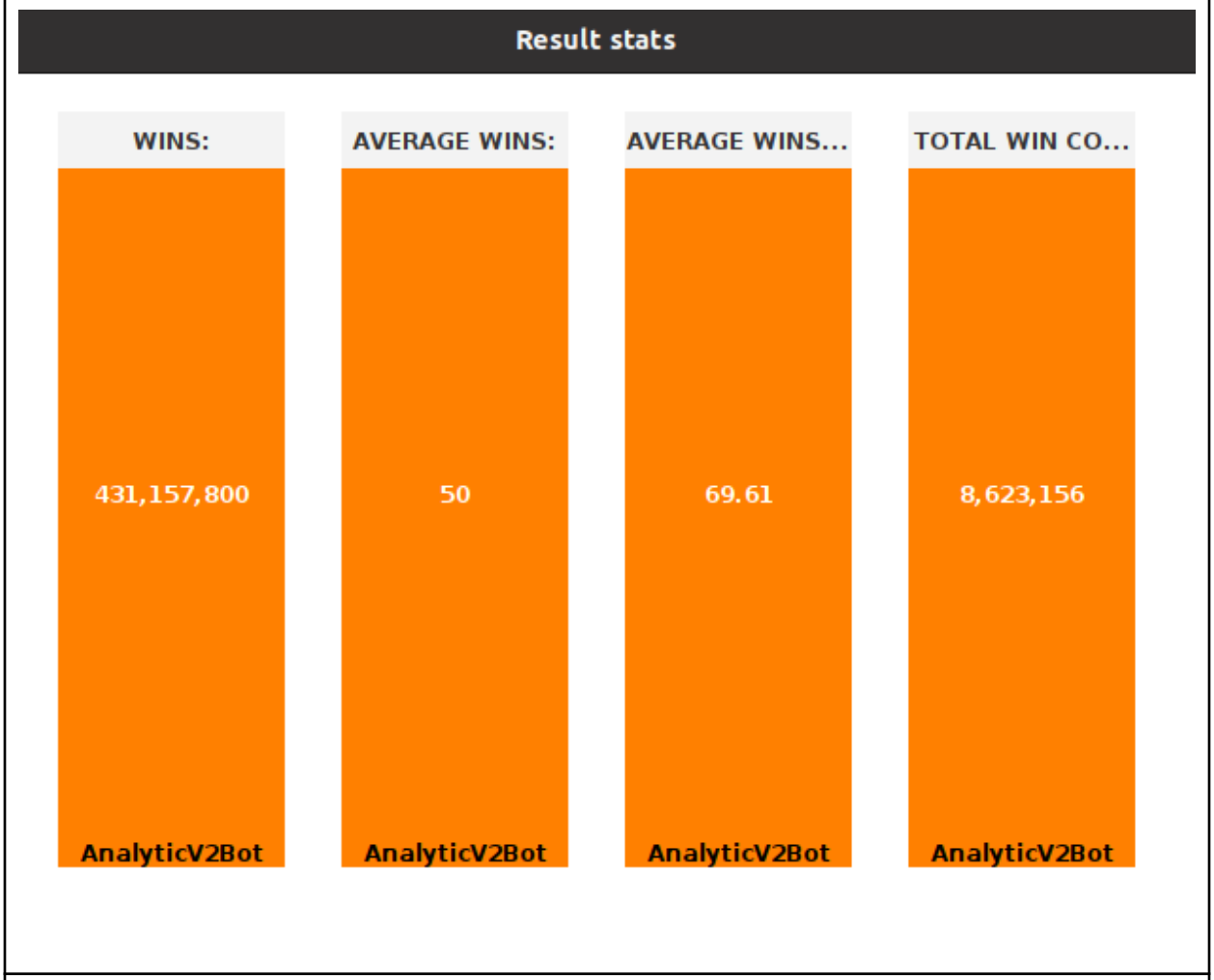



Рис. 4.9. AnalyticV2Bot тест

Перша діаграма показує загальну кількість виграних грошей для всіх AnalyticV2Bot'ів. 431,157,800 - це практично вся сума, яку було розіграно в тесті.

Також друга діаграма знову показує, що в середньому AnalyticV2Bot'и вигравали зі ставкою 50.

Третя діаграма показує лише середні виграші категорії ботів коли вони грали за A. Середній поділ дорівнює 69.61, що є менше ніж у AnalyticBot.

Четверта діаграма показує кількість загальних перемог. Тобто AnalyticV2Bot'и загально виграли трохи більше 8ми з половиною мільйонів разів. І це прекрасний результат.

Давайте дивитись детільніші цифри, щоб зрозуміти більш широку картину цього тесту (рис. 4.10).

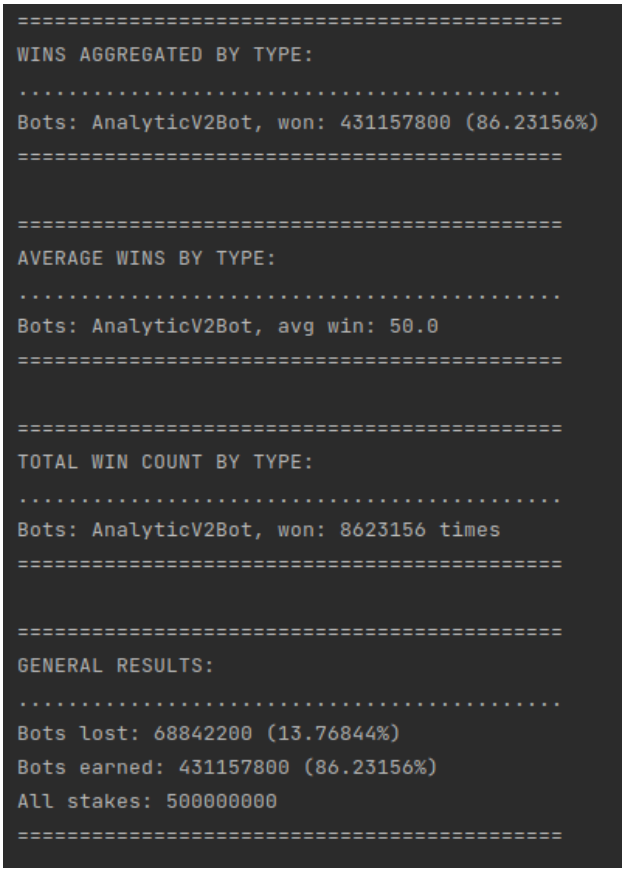



Рис. 4.10. AnalyticV2Bot тест (консоль)

Проаналізувавши рис. 4.10., 4.8. видно, що аналітики другої версії дійсно діють мудріше та вибирають свою стратегію більш оптимальною. Хоча середня А пропозиція для аналітиківV1 склала 72.2, а для аналітиківV2 – 69.61.

Якщо подумати, то дійсно, число, яке найближче до 70 ставити найвигідніше, адже проти тебе грають аналітики, які мають найменший поріг

прийняття за Б - 20%, а найбільший 30%. Тобто ставка в 70 для А пройде завжди! У той час як 72.2 - з ймовірністю 22% не пройде. Різниця в ставках не велика, а програш дійсно дає про себе знати.

Тепер щодо загальних чисел, порівнюючи аналітиків. Аналітики другої версії виграли: 86.2% всього капіталу. Аналітики першої - 61%. Переглядаючи також найуспішніших та найменш успішних ботів, бачимо таку картину:

Bot: AnalyticV2Bot, earned: 40519.0

Bot: AnalyticV2Bot, earned: 45546.0

А для аналітиків першої категорії, нагадую, значення були наступними:

Bot: AnalyticBot, earned: 28220.0

Bot: AnalyticBot, earned: 33496.0

### 4.6. Altruist vs Egoist тест

Ми розглянули усі доступні реалізації ботів. Тепер давайте конбінувати їх! Для початку скомбінуємо альтруїстів та егоїстів і подивимось що з цього вийде, та хто ж заробить більше грошей. Ми проведемо декілька тестів. Ось їхня загальна конфігурація:

- Грають 10,000 ботів
- Кількість раундів 1,000 при ставці 100
- Деякі боти EgoistBot, деякі AltruistBot.

На цій конфігурації ми проведемо декілька різних тестів, де буде змінюватись пропорція егоїстів до альтруїстів.

### 4.6.1 Altruist-50% Egoist-50%

Розглянемо результати при тесті, де 50% - це альтруїсти, а інші 50% - це егоїсти. рис. 4.11.

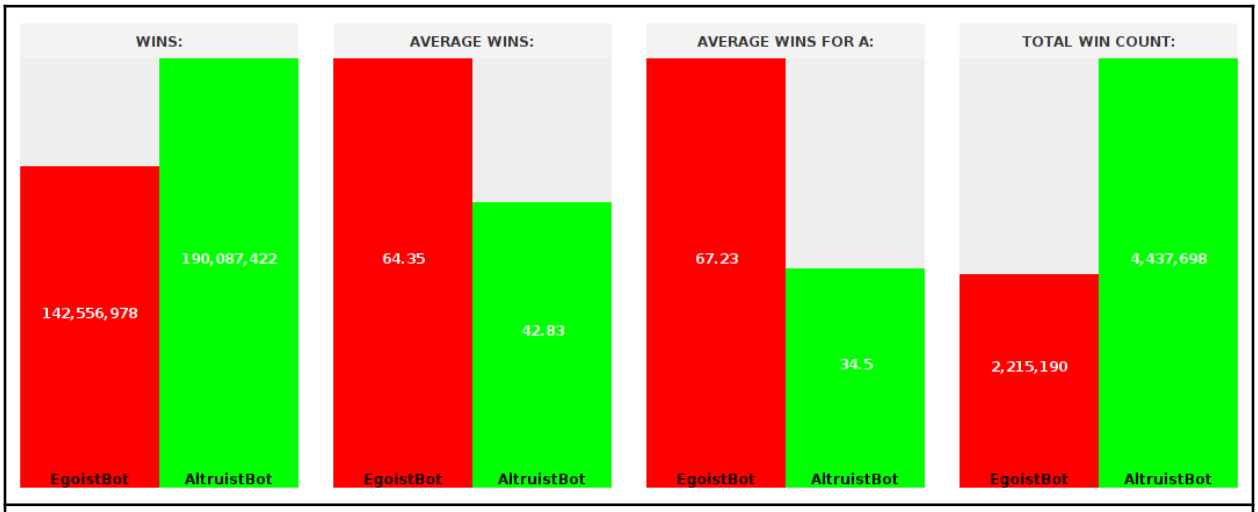



Рис. 4.11. Altruist-50% Egoist-50%

Тут очевидно, що альтруїсти отримали перемогу. Їхні виграші значно більші, навіть попри те, що середня переможна ставка у егоїстів значно більша. Все це нівелюється кількістю перемог, яка значно більша у альтруїстів.

Далі приведемо найуспішніших та найменш успішних ботів з кожної категорії:

Bot: EgoistBot, earned: 25093.0

Bot: EgoistBot, earned: 33014.0

Bot: AltruistBot, earned: 35430.0

Bot: AltruistBot, earned: 40138.0

### 4.6.2 .Altruist-10% Egoist-90%

Але що ж буде, якщо пропорції зміняться. Тепер альтруїсти в тотальній меншості, а егоїстів в 10 разів більше. Звичайно, що загальна картина буде в користь егоїстів, проте не все так просто. Результат можна побачити на рис. 4.12.

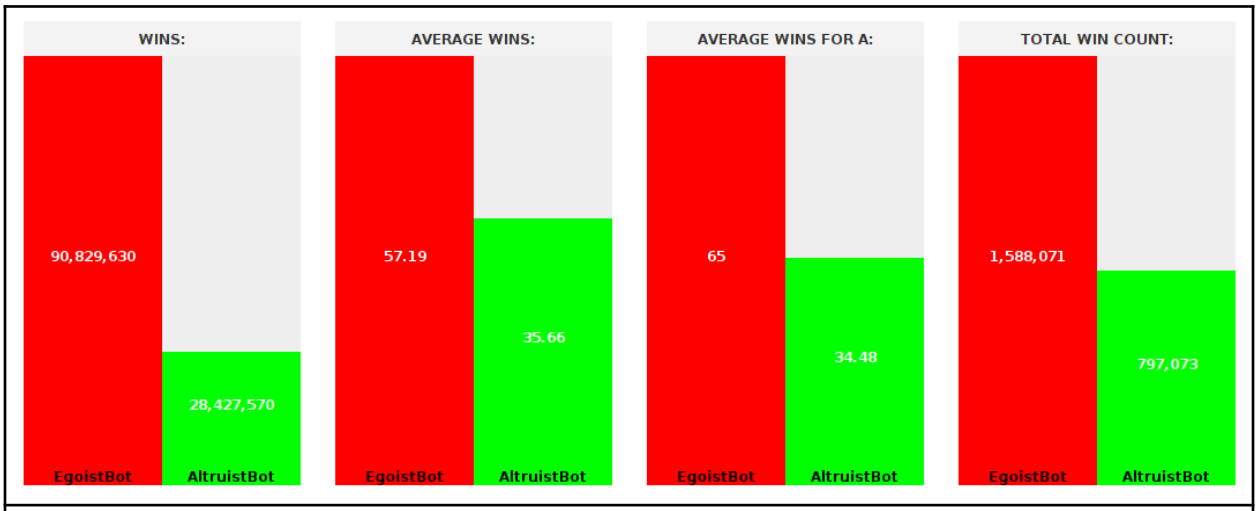



Рис. 4.12. Altruist-10% Egoist-90%

Дійсно, егоїсти лідирують на всіх графіках. Проте варто все-ж поглянути на найуспішніших та найменш успішних ботів з кожної категорії:

Bot: EgoistBot, earned: 7349.0

Bot: EgoistBot, earned: 13136.0

Bot: AltruistBot, earned: 26846.0

Bot: AltruistBot, earned: 30272.0

Успішність егоїстів значно скоротилась, якщо дивитись на окремі випадки. Хоча успішність альтруїстів теж впала - вони все-ще заробляють набагато більше.

### 4.6.3 Altruist-0.01% Egoist-99.99%

Фінальний тест - де альтруїст буде лише один серед 9999 егоїстів. Чи зможе він здобути найкращі показники? Давайте дивитись на результати. Див. рис. 4.13.

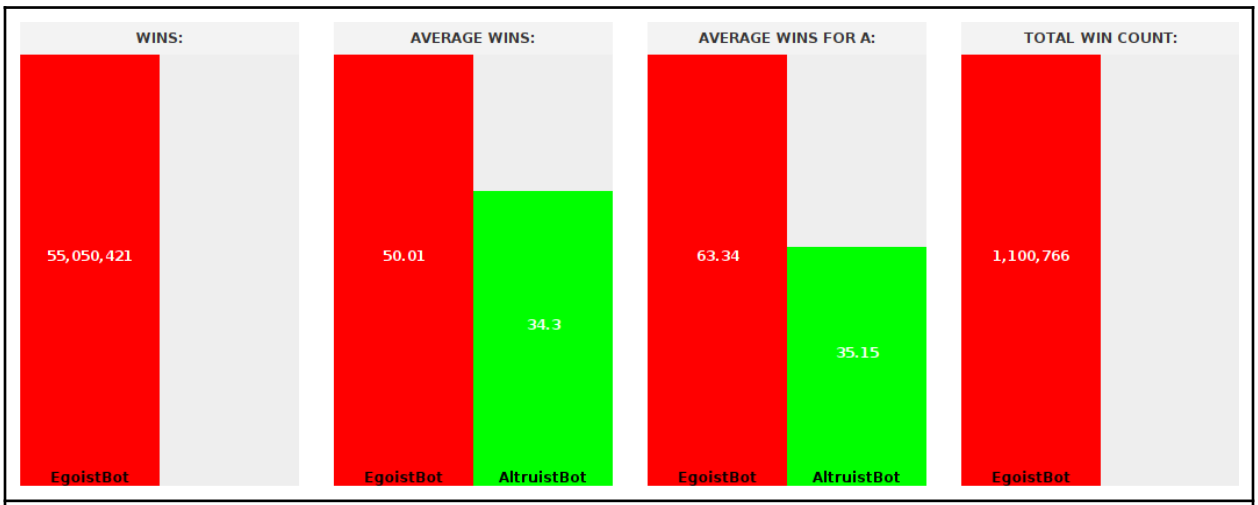



Рис. 4.13. Altruist-0.01% Egoist-99.99%

Звичайно, що загальна картина для альтруїстів дуже погана на фоні егоїстів, адже альтруїстів в тисячі разів менше. Числа настільки маленькі, що на фоні егоїстів - це практично заокруглені нулі, тому їх навіть не видно на графіках.

Але тепер до самого цікавого. Поглянемо на найуспішніших та найменш успішних ботів з кожної категорії:

Bot: EgoistBot, earned: 3594.0
Bot: EgoistBot, earned: 7629.0
Bot: AltruistBot, earned: 26479.0

Висновки такі, що альтруїст - завжди безумовний лідер серед будь-якої кількості егоїстів. Навіть якщо він один проти 9999 егоїстів, що є доволі цікавим спостереженням.

### 4.7. Analytic vs DummyBot тест

Це тест, який було обіцяно провести, коли ми розглядали перших аналітиків - невже dummy зможе заробити більше за аналітика? Давайте проведемо тести! Ось їхня загальна конфігурація:

- грають 10,000 ботів
- кількість раундів 1,000 при ставці 100
- деякі боти AnalyticBot, деякі DummyBot.

На цій конфігурації ми проведемо декілька різних тестів, де буде змінюватись пропорція аналітиків до dummy.

#### 4.7.1. Analytic-50% Dummy-50%

Розглянемо результати при тесті, де 50% - це аналітики, а інші 50% - це dummy. Див. рис. 4.14.

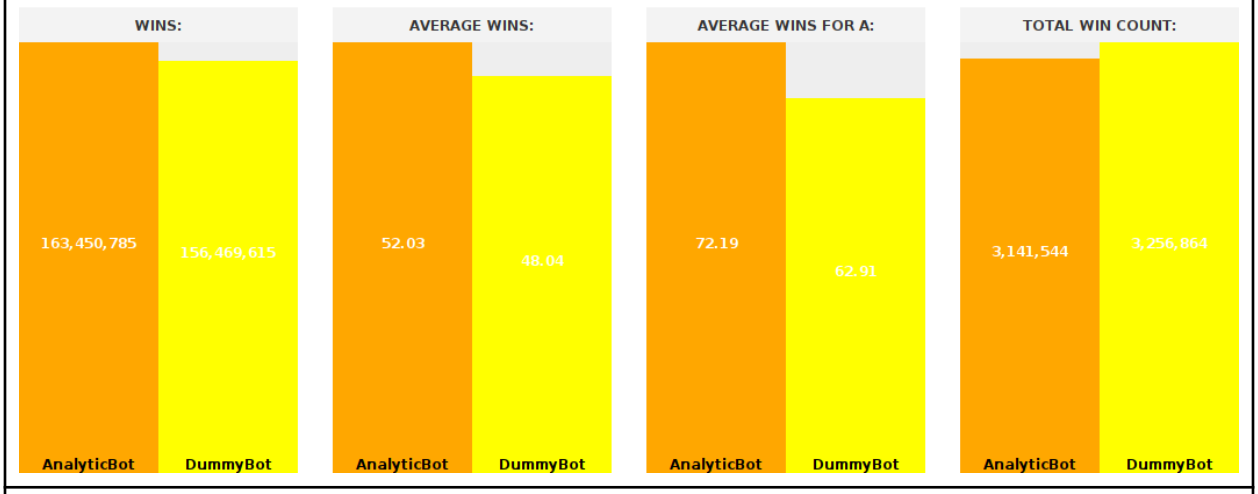



Рис. 4.14. Analytic-50% Dummy-50%

І от що ми бачимо. Хоча в одиночних тестуваннях, де DummyBot грав сам із собою, а AnalyticBot із собою - DummyBot виграв, в цьому тесті, де вони грають один проти одного - перемога за аналітиком!

Також поглянемо на найуспішніших та найменш успішних ботів з кожної категорії:

Bot: DummyBot, earned: 28614.0

Bot: DummyBot, earned: 34872.0

Bot: AnalyticBot, earned: 29839.0

Bot: AnalyticBot, earned: 35340.0

Як ми бачимо, ці відрізки наклались один на одного. Тобто в середньому вони заробляли однаково. Хіба-що аналітик трохи більше.

### 4.7.2. Analytic-10% Dummy-90%

Змінимо пропорції аналітиків до dummy. Тепер буде 10% до 90% відповідно. Проведемо тест (рис. 4.15.)

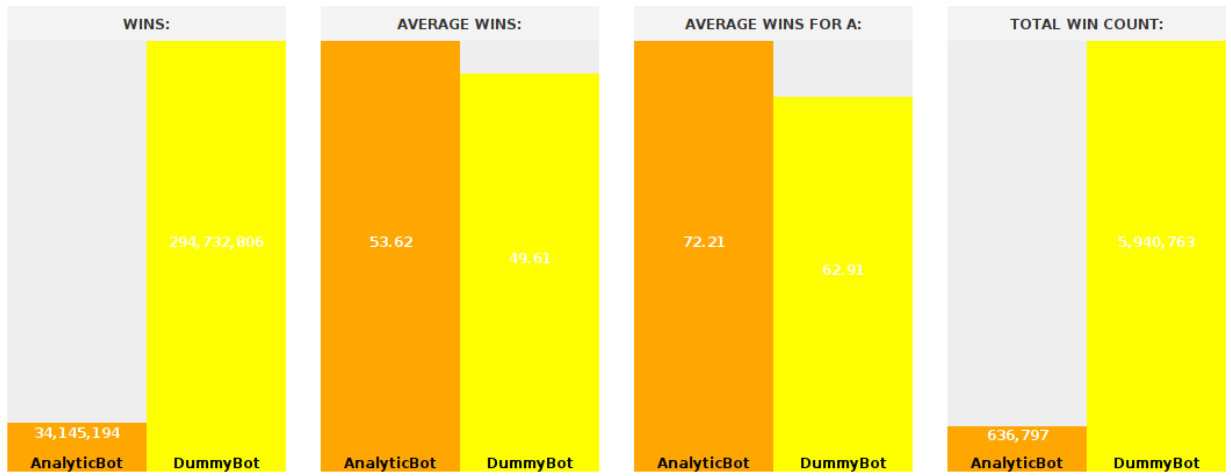



Рис. 4.15. Analytic-10% Dummy-90%

Звичайно, що загальні графіки значно впали. Перемог менше, сума виграшів відповідно теж менша. Поглянемо на найуспішніших та найменш успішних ботів з кожної категорії:

Bot: DummyBot, earned: 29709.0

Bot: DummyBot, earned: 36066.0

Bot: AnalyticBot, earned: 32042.0

Bot: AnalyticBot, earned: 36872.0

Практично нічого не змінилось. Хіба-що різниця між найуспішнішими на найменш успішними ботами зросла.

### 4.7. Analytic vs AnalyticV2 тест

Протестуємо чи краще працюють аналітики другої версії у порівнянні з аналітиками першої. Проведемо також тест проти dummy ботів, щоб порівняти з аналітиками першої категорії. Конфігурація буде стандартною. Також кількість ботів буде 50/50.

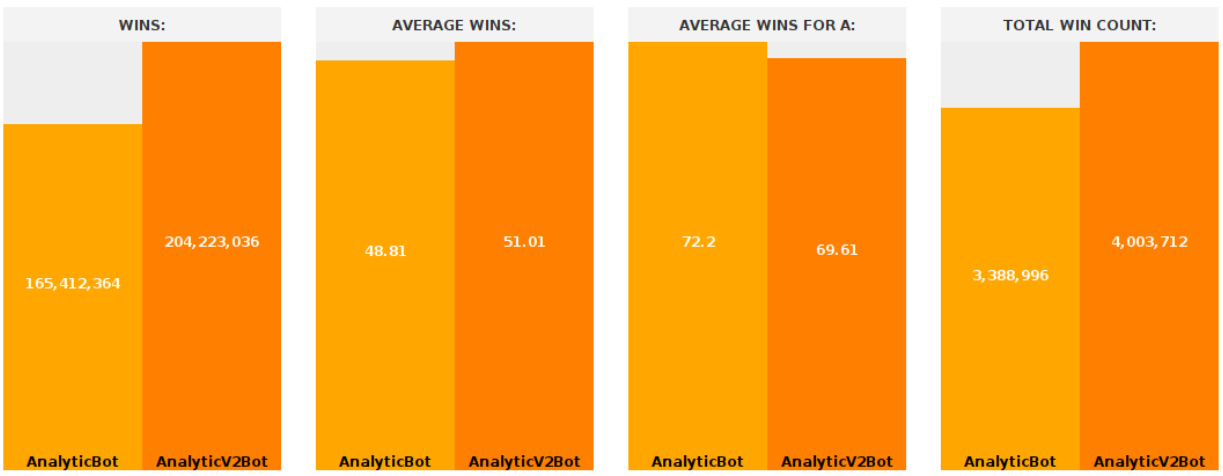
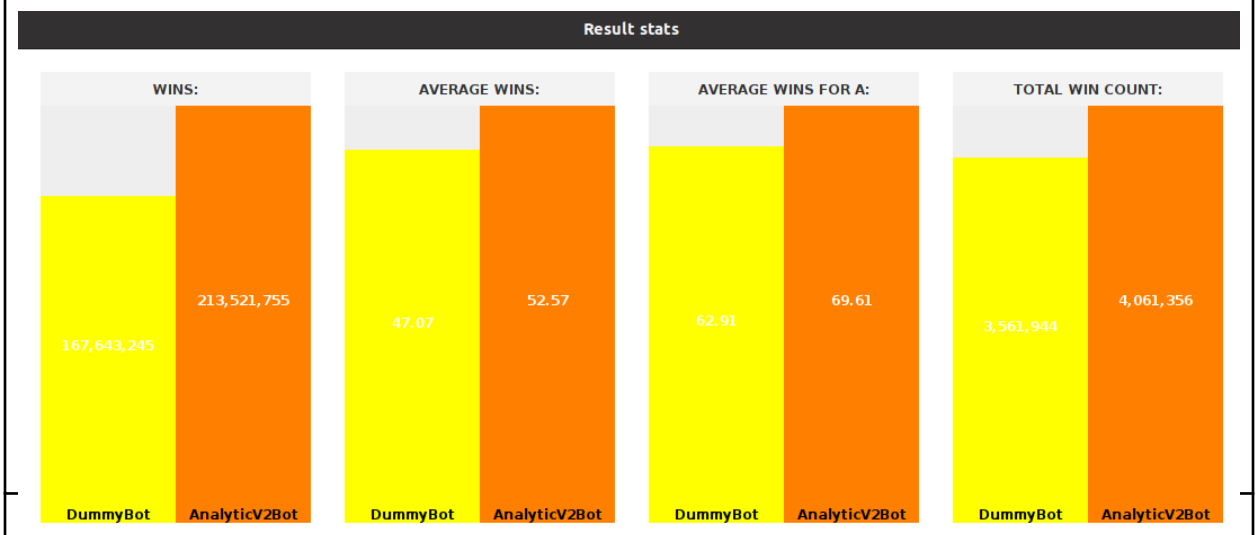



Рис. 4.16. Analytic-50% AnalyticV2-50%

Наглядно видно, що аналітики другої категорії стали переможцями у цьому тесті (рис. 4.16.). Тому давайте одразу порівняємо аналітиків другої версії з dummy ботами (рис. 4.17.).




Лідерство аналітиків другої версії очевидне. Справа в тому, що вони не витрачають час на пониження. Таким чином їм вдається набагато швидше і дещо точніше визначити межу прийняття опонента.

## ВИСНОВКИ ДО РОЗДІЛУ 4

У розділі "Тестування ботоферми" проведено докладний аналіз ефективності розробленої системи в різних категоріях ботів, що виокремлюються за їхнім стилем гри та стратегією поведінки.

Згідно проведеним тестуванням, було виявлено, що боти категорії AnalyticV2Bot демонструють найвищий рівень ефективності у грі "Ультиматум".

Друге місце у тестуванні зайняли боти AnalyticBot, що також підтверджує їхню значущість та придатність для використання у грі.

Значущим аспектом тестування є висновок, що AltruistBot в умовах гри "Ультиматум" завжди демонструє вищі показники доходу порівняно з EgoistBot.

# ВИСНОВКИ

У рамках даної кваліфікаційної роботи було проведено комплексні дії, спрямовані на створення та розвиток ботоферми для моделювання стратегій у психологічно-соціальній грі "Ультиматум".

Початковим етапом роботи був аналіз ультиматуму як слова. Визначення та розуміння самої гри “Ультиматум” та її ріних видів, таких як диктатор, ультимату на трьох та більше гравців, тощо. Процес включав вивчення загальних концепцій, пов'язаних із психологічно-соціальними аспектами гри, а також дослідження різних варіантів гри "Ультиматум".

На другому етапі роботи проводилося детальне ознайомлення з інструментами, які були використані для імплементації ботоферми, і вибір найбільш оптимальних для досягнення поставлених цілей. Зокрема, особлива увага приділялась аспектам роботи з інтерфейсом користувача (UI) в середовищі програмування Java.

Результатом третього етапу стало створення робочої програми, яка включала в себе не лише основний функціонал, а й забезпечувала зручний інтерфейс для користувачів. Було описано всі концепції, використані для проектування та розробки. Розповідалось також про внутрішню реалізацю різних класів, описувались різні підходи та основні концепції, використані у ході розробки. Також була розібрана тема GOF патернів.

На завершальному етапі було протестовано програму різними шляхами. Було виявлено явних лідерів при різних стратегіях гри та різним набором типів ботів. Найуспішнішим виявився бот - аналітик другої версії, який аналізував свої перемоги та поразки і вибирав найоптимальнішу ставку. Причому робив він це краще за аналітиків першої версії, саме тому вони посіли друге місце. Третє місце

за альтруїстами, четверте за егоїстами. Тестовий бот - DummyBot до розрахунку не брався.

Отриманий результат представляє собою готовий продукт, який може бути використаний в подальших дослідженнях в галузі психології та соціальних наук. Його особливість полягає в тому, що він містить вже реалізовані типові поведінки для використання у грі "Ультиматум" та надає можливість легко реалізувати власні стратегії без глибокого вивчення мов програмування. Це робить його доступним і корисним інструментом для використання як дослідниками, так і простими людьми, які цікавляться даною тематикою.

У подальшому, продукт (ботоферму) можна розвивати, додаючи інших ботів та створюючи інші діаграми для перегляду статистики, покращити ui за допомогою анімацій. Це б допомогло вивчити поведінки різних типів людей та їхні неегоїстичні наміри. Також, за необхідності, можна додати інші режими гри і тестувати існуючих ботів на нових правилах.

Гра “Ультиматум” хоч і проста, проте дуже гарно показує наміри та поведінку людей. Її цікаво досліджувати та дізнаватись більше про людей. Сподіваюсь цей проект у майбутньому стане у нагоді.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1 Навчальний посібник/Підручник: Сієра К., Бейтс Б. "Head First Java." За ред. Бенкета А. Київ: Діалектика, 2021. 720 с.

2 Навчальний посібник/Підручник: Eckel B. "Thinking in Java." Prentice Hall, 2020. 1150 p.

3 Навчальний посібник/Підручник: Фріман Е., Робсон Е., Бейтс Б., Сьюенсон К. "Head First Design Patterns." O'Reilly Media, 2020. 694 p.

4 Навчальний посібник/Підручник: Russell S., Norvig P. "Artificial Intelligence: A Modern Approach" (3rd Edition). Prentice Hall, 2019. 1132 p.

5 Ресурс Інтернету: https://en.wikipedia.org/wiki/Ultimatum_game

6 Ресурс Інтернету: https://inomics.com/terms/ultimatum-game-1538668#:~:text=The%20ultimatum%20game%20is%20a,are%20randomly%20and%20anonymously%20matched.

7 Ресурс Інтернету: https://life.pravda.com.ua/columns/2018/02/3/228782

8 Ресурс Інтернету: https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D1%96%D1%8F_%D1%96%D0%B3%D0%BE%D1%80

9 Ресурс Інтернету: https://en.wikipedia.org/wiki/Swing_(Java)

10 Ресурс Інтернету: https://www.w3schools.com/java/

11 Ресурс Інтернету: https://maven.apache.org/

12 Ресурс Інтернету: https://central.sonatype.com/

13 Ресурс Інтернету: https://refactoring.guru/design-patterns/catalog

14 Ресурс Інтернету: https://refactoring.guru/design-patterns/abstract-factory

15 Ресурс Інтернету: https://www.globallogic.com/ua/insights/blogs/dive-into-the-selenium-patterns/

16 Ресурс Інтернету: https://uk.wikipedia.org/wiki/%D0%A8%D1%82%D1%83%D1%87%D0%BD

%D0%B8%D0%B9_%D1%96%D0%BD%D1%82%D0%B5%D0%BB%D0%B5%D0%BA%D1%82

17 Ресурс Інтернету: https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%B5_%D0%BD%D0%B0%D0%B2%D1%87%D0%B0%D0%BD%D0%BD%D1%8F

18 Ресурс Інтернету: https://ru.wikipedia.org/wiki/Java

19 Ресурс Інтернету: https://www.javatpoint.com/javafx-tutorial

20 Ресурс Інтернету: https://www3.ntu.edu.sg/home/ehchua/programming/java/j4a_gui.html