МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних технологій

Кваліфікаційна робота бакалавра

на тему: «Створення ігрового штучного інтелекту для гри «Монополія»»

Виконав студент групи К-19і
Спеціальності 122 Комп'ютерні науки,
Піщік Владислав Сергійович

Керівник доктор філософії, асистент
Бучинська Ірина Вікторівна

Рецензент д.ф.-м.н., професор
Ковальчук Володимир Володимирович

Одеса 2021

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА ТЕРМІНІВ

ІШІ – штучний інтелект

ШНМ – штучні нейроні мережі

AWT – Abstract Window Toolkit

FFNN – Feed Forward Neural Networks

JPQL – Java Persistence Query Language

JVM – Java Virtual Machine

HTML – HyperText Markup Language

IDE – Integrated Drive Electronics

Рефакторинг – перетворення програмного коду

Тензор (tensor) – стандартний спосіб представлення даних в системах глибокого навчання

C – мова програмування

C++ – мова програмування

Java – мова програмування

IntelliJ IDEA – середовище розробки для Java

Kotlin – мова програмування

Scala – мова програмування

SQL – мова програмування структурних запитів

SWING – бібліотека для створення графічного інтерфейсу

TensorFlow – опенсорсна бібліотека

ВСТУП

В останні роки розробляються системи штучного інтелекту (ШІ), які обіграють людей в різні комп'ютерні ігри. Для впровадження таких систем в ігри, треба зрозуміти, як вони працюють. До систем ШІ варто відносити ті методи і алгоритми, які симулюють одну або кілька когнітивних здібностей, властивих живим організмам. При цьому ці алгоритми зовсім не обов'язково повинні бути аналогічними тим, які використовують живі організми.

Наприклад, система, що розпізнає контури предметів на фотографії, буде системою ШІ, оскільки завдання розпізнавання є когнітивної. Але ось більшість ботів в сучасних іграх до ШІ віднести повною мірою не можна, оскільки вони працюють за суворим алгоритмом, де на конкретну дію гравця бот реагує конкретним чином. Ніяких когнітивних завдань не вирішується.

Крім цьогос існують системи, які можуть симулювати гравця і стати опонентом в іграх. Це алгоритми ШІ, які здатні симулювати ту чи іншу поведінку людини. В їх архітектуній основі – штучні нейроні мережі (ШНМ).

ШНМ це не лінійний алгоритм, а величезна кількість пов'язаних між собою штучних нейронів. Кожен нейрон отримує сигнали від сотень нейронів і сотням ж інших нейронів передає.

Підсумком цього стає те, що такі мережі, за великим рахунком – чорний ящик, поведінка якого передбачити неможливо, якщо не написати програму в десятки разів більше і складніше для аналізу створеної мережі. Більш того, така мережа в процесі створення модифікує сама себе. Вони не є статичні, так як навчаються методом підкріплення [1].

Метою дипломного проекту є розробка бота на основі нейронної мережі для ведення гри «Монополія».

Для реалізації мети необхідно виконати наступні завдання:

− реілазувати алгоритм гри за спрощеними умовами;

− для навчання штучного інтелекту використовувати гені алгоритми

для нейроної мережі

- проаналізувати повевідну примірників проходження партії для виявлення найбільш успішнішого бота.

Загальні характеристики кваліфікаційної роботи:

- повний обсяг сторінок пояснювальної записки – 50;
- кількість рисунків – 20;
- кількість посилань – 14.

1 АНАЛІТИЧНИЙ РОЗДІЛ

1.1 Опис предметної області

Монополія –це гра, в якій гравці прагнуть збільшити свій стан шляхом покупки, оренди або продажу нерухомості, і переможцем стає найбагатший. Гра починається з поля «ВПЕРЕД», якщо гравець опиняєтеся на поле, яке ще не зайняте іншими, він може купити його у Банку. У разі вирішення не купувати його, воно продається з аукціону тому, хто запропонує за нього найвищу ціну. Гравці, які володіють Нерухомістю, беруть орендну плату, зі своїх супротивників, які виявляться на полях.

Якщо гравцю потрібно залучити додаткові кошти, Банк може дати позику під заставу ділянки для будівництва. Необхідно завжди строго слідувати інструкціям, наведеним на картках «Громадської Скарбниці» та «Шансів». За результатами коштів визначається переможець у кінці вказаного ліміту часу.

Штучну нейронну мережу можна навчити грати в якусь гру, але для більш поглибленого навичка їй необхідно треба надати якомога більше ігрових ситуацій. Зі зростанням кількості розборів таких ситуацій мережа буде мати складнішу поведінку. Вона буде «пам'ятати» досвід минулих ігор. З кожною новою вивченої ситуацією, на старі вона буде реагувати вже по-іншому. І якщо ми говоримо про мережу, яка зможе грати на рівні людини, ми повинні говорити про мережі, навченої на мільйонах ігрових ситуацій.

1.2 Характеристика об’єкту розробки

Торгівля нерухомістю може здатися не дуже привабливою передумовою для настільної гри. Але «Монополія» довела, що це так, і мільйони настільних гравців у всьому світі згодні з цим. Початковий набір з'явився в 1935 році, але його походження можна простежити і далі: активістка на ім'я

Ліззі Мегі розробила попередницю відомої гри під назвою «The Landlord's Game», намагаючись проілюструвати небезпеки монополізму.

В даний час гра більше цікава, ніж освітні, і існують її версії, натхненні незліченними містами і популярними вигаданими вселеними. Ця класична версія гри надає цифрові засоби накопичення багатства – а це означає, що гравцю не доведеться мати справу з паперовими грошима або гральними кістками.

Монополія належить «Board» і часто асоціюється з «Tycoon Games». Ця гра отримала 6991 голосів, 4956 позитивних і 2035 негативних, і має середній бал 3,6. Це гра, в яку грають в горизонтальному режимі, і в неї можна грати на робочому столі на сайті www.gamepix.com. На мобільних пристроях він також доступний в Google Play і Apple App Store.

Планується створити гру з обмеженими умовами для тестування ботів та навчання їх на основі гених алгоритмів. За результатами тестів буде виявлено найуспішнішого гравця та проаналізовані вимоги та час, який йому знадобиться для перемоги.

1.3 Аналіз аналогів

Боти – це автоматичні програми, які допомагають економити сили і час на пошук інформації. Принцип їх роботи такий: вони задають вам кілька запитань про те, що ви шукаєте, аналізують відповідь і видають список найкращих збігів.

Боти – це помічники, які роблять велику частину роботи за користувачів. Ще одна функція ботів – гри.

Діапазон їх складності надзвичайно широкий – від простих ребусів до багаторівневих багатокористувацьких ігор.

Слід розглянути аналоги ботів для ігор більш детально для подальшої постановки завдання до дипломної роботи. «@M0n0Bot» – ігровий бот, у якого багато ігор на різні смаки і теми (рис. 1). Гравець може обрати одну з

ігор: карти, кістки, дурень, монополія та інші розваги. Простенький бот з інтуїтивно зрозумілим інтерфейсом і меню:

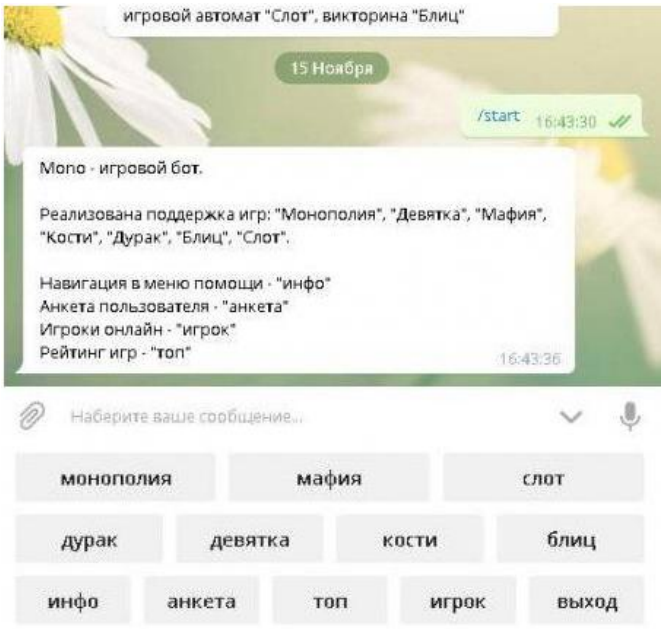


Рисунок 1 – Скрін роботи чат-боту Mono

«RENTO MONOPOLY» (рис. 2) – це розрахована на багато користувачів настільна онлайн-гра-монополіст.

Рисунок 2 – Скрін роботи гри Ретро-Монополія

Можна приймати учать одночас 2-6 гравцям. Крім цього є можливість грати в неї онлайн проти друзів, офлайн проти роботів, на одному телефоні з друзями або по Bluetooth. Також можна запросити своїх друзів з Facebook пограти.

У грі гравці торгують землями, будують будинки, вигравають аукціони. Мета – отримати монополію і розорити інших гравців. У грі є безліч різних дошок, на яких розміщуються дії гри, призначені для користувача пішаки, призначені для користувача кістки, інші настройки для різних правил будинку і багато іншого.

Крім цього, при відсутності інших гравців можна включити функцію гри з ботом, але в його алгоритмі тільки прості інструкції до дій, тобто ілюзії що гра йде з людиною відсутня.

Наступний аналог є одним з топових онлайн платформ гри «Монополія» – «marmaladegamestudio» (рис. 3).

Рисунок 3 – Приклад ігрового поля для монополії

Цієї популярної настільної грою від Hasbro і позачасовий сімейної класикою користуються більше мільярда людей по всьому світу, і тепер вона до-

ступна на мобільних пристроях і планшетах. Marmalade Game Studio оживляє ігрову дошку за допомогою красивого анімованого 3D-міста, ретельно розробленого для легкого і інтерактивної взаємодії з мобільними пристроями[2]..

Гру можна налаштувати за індивідуальних побажанням – «швидким режимом» і «домашніми правилами». Крім цього, підтримується режим «Один гравець» – коли людина має можливість грати з штучним інтелектом. Як додатковий бонус – ігрове поле можна обрати із п’яти типів.

«Monopoly One» – ще один онлайн ресурс для улюбленої гри [3]. По-перше, для участі в грі необхідна реєстрація користувача, але відсутня плата та розсилки. Гарний дизайн поля (рис. 4) та наявність декількох мов.

Рисунок 4 – Скріншот гри «Monopoly One»

Гравець має можливість обрати собі аватарку та зберігати історію попередніх змагань. Крім цього, дуже зручне посилання-запит, який можна пересилати своїм друзям у різні соціальні мережі.

За кожного нового гравця на даній платформі, користувач отримує додаткові бонуси. Після завершення раунду в кабінет гравця висвічується ста-

тистика та зберігається історія (з ким він грав і який рахунок при завершені гри).

Останнім аналогом було розглянуто гру «Monopoly Leealex», скріншот якої представлено на рис. 5.

Рисунок 5 – Скріншот гри «Monopoly Leealex»

Зручний інтерфейс та вдале графічне оформлення зробили цей варіант гри одним з топових. Гра розрахована до 6 гравців, якщо гравець використовує онлайн-версію, можна грати як з людиною, так і з штучним інтелектом, який вдало веде свою стратегію.

За результатами аналітичного огляду предметної області і аналогів було прийнято рішення розробити гру – як площадку для навчання ботів з обмеженими умовами.

2 ПРОГРАМНІ ЗАСОБИ РОЗРОБКИ ТА АЛГОРИТМИ

2.1 Середовище розробки для Java IntelliJ IDEA

IntelliJ IDEA середовище розробки, що було обрано для розробки дипломного проекту. IntelliJ IDEA це найрозумніша і зручне середовище розробки для Java, що включає підтримку всіх останніх технологій і фреймворків. Кожен компонент IntelliJ IDEA створений для того, щоб максимально підвищити продуктивність розробки. Розумний редактор коду в поєднанні з ергономічним дизайном роблять розробку не тільки ефективною, а й приємною. IntelliJ IDEA (рис. 6) надає інструменти для продуктивної роботи і ідеально підходить для створення комерційних, мобільних і веб-додатків [3].

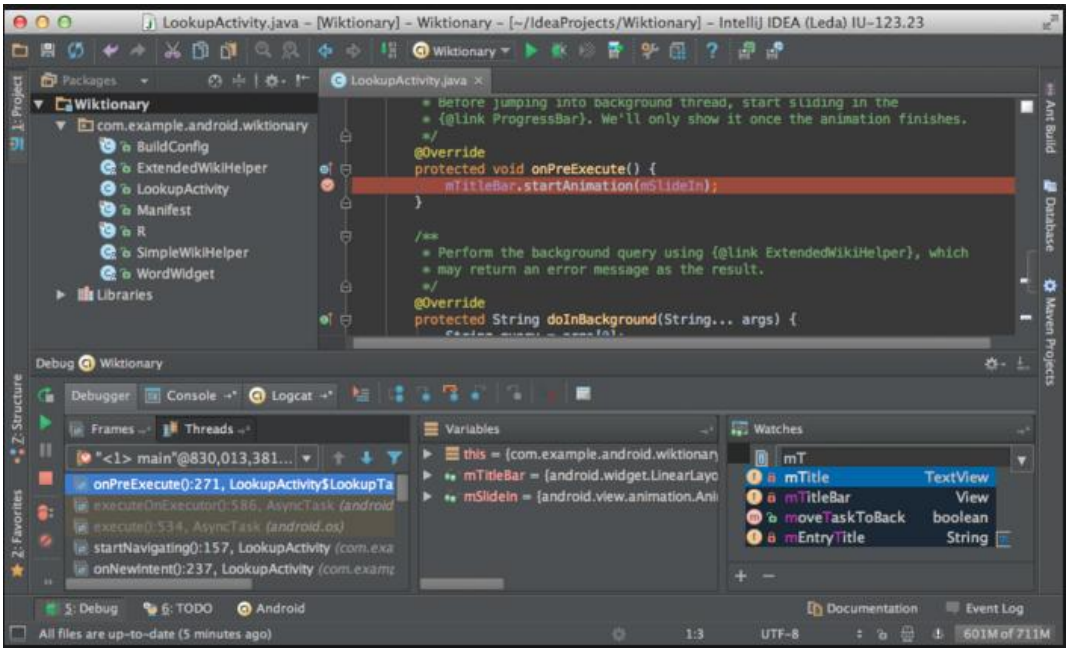


Рисунок 6 – Вікно середи розробки IntelliJ IDEA

Після індексування вихідного коду IntelliJ IDEA надає масу можливостей для швидкої і ефективної розробки: розумне автодоповнення, аналіз коду в реальному часі і надійні рефакторингом.

Всі необхідні інструменти вже під рукою, а значить не потрібно шукати і встановлювати плагіни для інтеграції з системами контролю версій і підтримки популярних мов і фреймворків. У той час як базове автодоповнення пропонує імена класів, методів, полів і ключових слів в області видимості, розумне автодоповнення пропонує тільки ті елементи коду, які актуальні в поточному контексті.

Незважаючи на те, що IntelliJ IDEA – в першу чергу IDE для Java, вона розуміє і надає інтелектуальну допомогу при написанні коду на SQL, JPQL, HTML, JavaScript і багатьох інших мовах і дозволяє редагувати код, написаний нема на Java, всередині строкових літералів Java-коду.

IntelliJ IDEA аналізує одноманітні завдання в процесі розробки і автоматизує їх, щоб ви могли зосередитися на загальній картині. Це не тільки допомагає писати код в редакторі, але і спрощує роботу в цілому. Розробник зможе легко і швидко заповнити поле, знайти елемент у списку, відкрити потрібне вікно, поміняти настройки та інше [4].

2.2 Опис мови програмування Java

Мова програмування Java був розроблений компанією Sun Microsystems і є об'єктно-орієнтованим. Вихідний код програми Java перетвориться компілятором javac в спеціальний байт-код для виконання під управлінням віртуальної Java машиною.

Віртуальна Java машина JVM (Java Virtual Machine) – це програма, яка обробляє байт-код і передає інструкції обладнанню як інтерпретатор. Одним з основних переваг даного способу виконання програм є повна незалежність від операційної системи і устаткування, що дозволяє виконувати Java-додатки на будь-якому пристрої, для якого існує відповідна віртуальна машина.

Також до важливих особливостей технології Java слід віднести гнучку систему безпеки, в рамках якої виконання програми повністю контролюється

віртуальною машиною. Будь-які дії, які порушують встановлені програмою повноваження (наприклад, спроба несанкціонованого доступу до даних або з'єднання з іншим комп'ютером), викликають негайне переривання роботи програми.

До недоліків концепції використання віртуальної машини слід віднести зниження продуктивності, з яким боряться різними способами:

- застосування технології трансляції байт-коду в машинний код безпосередньо під час роботи програми – JIT-технологія;
- широкого використання переносних орієнтованого коду (native коду) в стандартних бібліотеках, наприклад SWT;
- апаратні засоби, що забезпечують прискорену обробку байт-коду, наприклад, технологія Jazelle, підтримувана деякими процесорами фірми ARM [5]. Переваги Java як мови програмування: об'єктно-орієнтована: в Java все є об'єктом. Доповнення може бути легко розширено, тому що він заснований на об'єктної моделі. Крім цього, це платформонезавісимость: на відміну від багатьох інших мов, включаючи C і C++, Java, коли був створений, він не компілювався в платформі конкретної машини, а в незалежній від платформи байт-коді. Цей байт код поширюється через інтернет і інтерпретується в Java Virtual Machine (JVM), на якій він в даний час працює.

Java простий: процеси вивчення та введення в мову програмування Java залишаються простими. Методи перевірки автентичності засновані на шифруванні з відкритим ключем. Компілятор генерує архітектурно-нейтральні об'єкти формату файлу, що робить скомпільований код виконуваним на багатьох процесорах, з наявністю системі Java Runtime.

Java портативний: архітектурно-нейтральний і не має залежності від реалізації аспектів специфікацій – все це робить Java портативним. Компілятор в Java написаний на ANSI C з чистою переносимість, який є підмножиною POSIX. Java виконує зусилля, щоб усунути помилки в різних ситуаціях, спираючись в основному на час компіляції, перевірку помилок і пере-

вірку під час виконання. Можна писати програми, які можуть виконувати безліч завдань одночасно. Введення в мову Java цієї конструктивної особливості дозволяє розробникам створювати налагоджені інтерактивні додатки.

Java байт-код переводиться на льоту в машинні інструкції та ніде не зберігається. Роблячи процес більш швидким і аналітичним, оскільки зв'язування відбувається як додаткове з невеликою вагою процесу. Введення Just-In-Time компілятора, дозволило отримати високу продуктивність.

Програмування на Java вважається більш динамічним, ніж на C або C++, так як він призначений для адаптації до мінливих умов. Програми можуть виконувати велике кількість під час обробки інформації, яка може бути використана для перевірки і дозволу доступу до об'єктів на час виконання [6].

2.3 Опис бібліотеки SWING

Графічний інтерфейс в Java пройшов вельми тернистий шлях розвитку і становлення. Першою спробою Sun створити графічний інтерфейс для Java була бібліотека AWT (Abstract Window Toolkit) – інструментарій для роботи з різними віконними середовищами. Sun зробив прошарок на Java, яка викликає методи з бібліотек, написаних на C.

Бібліотечні методи AWT створюють і використовують графічні компоненти операційного середовища. Так як програма на Java схожа на інші програми в рамках однієї ОС, то при запуску її на іншій платформі можуть виникнути розбіжності в розмірах компонентів і шрифтів, які будуть псувати зовнішній вигляд програми.

Щоб забезпечити мультиплатформеність AWT інтерфейси викликів компонентів були уніфіковані, внаслідок чого їх функціональність вийшла трохи урізаною. Та й набір компонентів вийшов досить невеликий. Так наприклад, в AWT немає таблиць, а в кнопках не підтримує відображення іконок. Проте пакет java.awt входить в Java з самого першого випуску і його можна використовувати для створення графічних інтерфейсів [7].

Слідом за AWT Sun розробила графічну бібліотеку компонентів Swing, повністю написану на Java. Для відтворення використовується 2D, що принесло з собою відразу кілька переваг. Набір стандартних компонентів значно перевершує AWT за різноманітністю і функціональності. Swing дозволяє легко створювати нові компоненти, наслідуючи від існуючих, і підтримує різні стилі і скіни.

Потрібний ступінь гнучкості і керованості забезпечували тільки легковагі компоненти. На діаграмі успадкування представлена зв'язок між AWT і Swing (рис. 7).

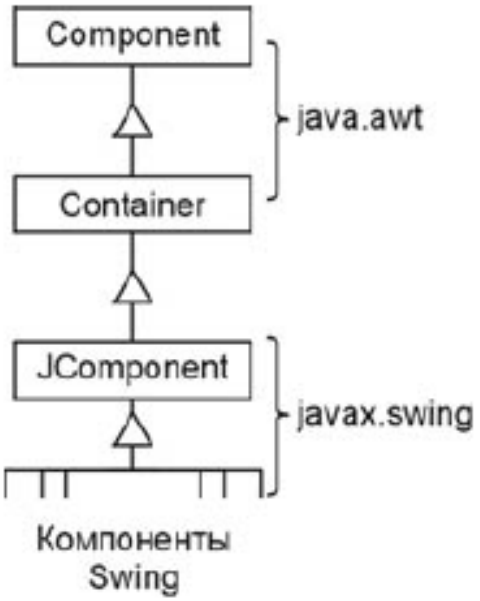


Рисунок 7 – Зв'язок між AWT і Swing

Найважливішою відмінністю Swing від AWT є те, що компоненти Swing взагалі не пов'язані з операційною системою і тому набагато більш стабільні і швидкі. Такі компоненти в Java називаються легковагими (lightweight), і розуміння основних принципів їх роботи багато в чому пояснить роботу Swing [7].

2.4 Основны характеристики нейронних мереж типу Feed Forward

Перші штучні нейронні мережі були створені в результаті спроб створити комп'ютерну модель, яка б відтворювала діяльність мозку в спрощеній формі.

Звичайно, можливості людського мозку незмірно більше, ніж можливості найпотужнішою штучної нейронної мережі. Однак штучні нейромережі мають ряд властивостей властивих біологічним нейромереж, в тому числі і людському мозку.

Нейронні мережі прямого поширення (feed forward neural networks, FF або FFNN) дуже прямолінійні, вони передають інформацію від входу до виходу. Нейронні мережі часто описуються у вигляді листкового торта, де кожен шар складається з вхідних, прихованих або вихідних клітин.

Клітини одного шару не пов'язані між собою, а сусідні шари зазвичай повністю пов'язані. Найпростіша нейронна мережа має дві вхідних клітини і одну вихідну, і може використовуватися в якості моделі логічних вентилів. FFNN зазвичай навчається за методом зворотного поширення помилки, в якому мережа отримує безлічі вхідних і вихідних даних (рис. 8).

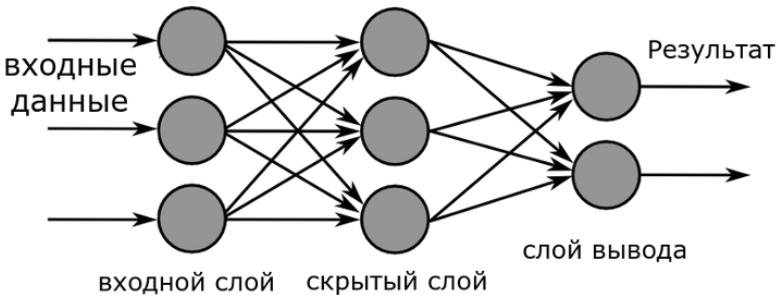


Рисунок 8 – Структура FFNN

Цей процес називається навчанням з учителем, і він відрізняється від навчання без учителя тим, що в другому випадку безліч вихідних даних ме-

режу становить самостійно. Вищезазначена помилка є різницею між введенням і висновком. Якщо у мережі є достатня кількість прихованих нейронів, вона теоретично здатна змоделювати взаємодію між вхідним і вихідними даними. Практично такі мережі використовуються рідко, але їх часто комбінують з іншими типами для отримання нових [8].

Головне властивість нейромереж – здатність до навчання. Для вирішення будь-якої задачі на комп'ютері традиційним методом необхідно знати правила, за якими з вхідних даних можна отримає вихідні. За допомогою нейромережі можна знайти рішення, маючи кілька прикладів.

Нейромережі використовують підхід до вирішення завдань ближчий до людського, ніж традиційні обчислення. Справді, наприклад, коли людина переходить вулицю, він оцінює швидкість руху автомобілі виходячи з попереднього досвіду не використовуючи математичних обчислень.

2.5 Опис бібліотеки TensorFlow

Машинне навчання набирає популярність і використовується у всьому світі. Він уже радикально змінив спосіб створення певних програм і, ймовірно, буде продовжувати залишатися величезною (і постійно збільшується) частиною нашого повсякденного життя. Такі компанії, як Google, взяли на себе завдання наблизити концепції машинного навчання до розробників і дозволити їм поступово, з серйозною допомогою, робити свої перші кроки. Так народилися такі фреймворки, як TensorFlow [9].

TensorFlow – це опенсорсна бібліотека, створена Google, яка використовується при розробці систем, що використовують технології машинного навчання. Тензор (tensor) – це стандартний спосіб представлення даних в системах глибокого навчання.

Тензори – багатовимірні масиви, розширення двовимірних матриць для представлення даних, що мають більш високі розмірності. Ця бібліотека включає в себе реалізацію безлічі потужних алгоритмів, розрахованих на рі-

шення поширених завдань машинного навчання, серед яких можна відзначити розпізнавання образів і прийняття рішень. Розробники бібліотеки TensorFlow прагнули до того, щоб вона була б зробили її гнучкою, ефективною, що розширюється, яку переносять.

В результаті їй можна користуватися в самих різних обчислювальних середовищах – від тих, які формуються мобільними пристроями, до середовищ, представлених величезними кластерами. Бібліотека дозволяє швидко готувати до реальної роботи навчені моделі, що усуває необхідність у створенні особливих реалізацій моделей для продакшн-цілей.

Бібліотека TensorFlow, з одного боку, привертає до себе увагу опенсорс-спільноти та відкрита для інновацій, а з іншого – користується підтримкою великої корпорації. Це дозволяє говорити про те, що у неї є всі шанси на стабільний розвиток.

З того моменту, як ця бібліотека стала опенсорсний, вона є однією з найцікавіших бібліотек машинного навчання. Її все частіше використовують при проведенні досліджень, при розробці реальних програм, при навчанні. TensorFlow постійно поліпшується, її постійно постачають чимось новим, оптимізують [10].

TensorFlow Java може працювати на будь-який JVM для створення, навчання і розгортання моделей машинного навчання. Він підтримує виконання як CPU, так і GPU в графічному або активному режимі і представляє багатий API для використання TensorFlow в середовищі JVM. Java та іншими мовами JVM, такі як Scala і Kotlin, часто використовуються на великих і малих підприємствах по всьому світу, що робить TensorFlow Java стратегічним вибором для широкого впровадження машинного навчання [11].

2.6 Характеритика генетичних алгоритмів

Машинне навчання стало такою популярною і активно розвивається наукою. глибокому навчанні, яке засноване на використанні багатошарових

нейронних мереж в якості базового інструменту. для навчання і вдосконалення нейронних мереж використовується генетичний алгоритм.

Генетичний алгоритм – це техніка оптимізації на основі пошуку, яка копіює природний відбір і генетику. У ній використовується таке ж поєднання відбору, кросинговеру і мутації для зміни вихідної випадкової популяції [12].

Одним з найбільш важливих переваг генетичних алгоритмів є відсутність необхідності інформації про поведінку функції і незначний вплив можливих розривів на процеси оптимізації. Також як і у випадку нейронних мереж, відбувається відхід від необхідності аналізу причинно-наслідкових зв'язків, шляхом побудови «підсумкового» образу – цільової функції. У цьому сенсі, з точки зору вирішення аналізу тексту, пошуку генетичні завдання вирішують такі ж завдання або дуже схожі, що і методи латентно-семантичного аналізу. В питаннях семантичного пошуку і індексації текстів генетичні алгоритми мають набагато більші перспективи, в порівнянні методами латентно-семантичного аналізу.

З точки зору розпізнавання образів, з дуже сильною натяжкою цільову функцію можна порівняти з шаром вхідних нейронів і з очікуванням максимуму як аналога максимізації сигналу нейрона вихідного шару. Генетичні алгоритми використовуються для підвищення ефективності навчання нейронних мереж, але все ж не можуть розглядатися як конкуренція нейронними мережами [13].

Оскільки генетичний алгоритм самонавчальний, то спектр застосування вкрай широкий:

- завдання на графи;
- завдання компонування;
- складання розкладів;
- створення «штучного інтелекту».

Генетичний алгоритм – це в першу чергу еволюційний алгоритм, оснований на комбінуванны. Ідея алгоритму взята у природи, тобто шляхом перебору і найголовніше відбору виходить правильна «комбінація».

Алгоритм ділиться на три етапи:

1) схрещування;
2) селекція (відбір);
3) формування нового покоління.

Якщо результат нас не влаштовує, ці кроки повторюються до тих пір, поки результат нас не почне задовольняти або станеться одне з нижче перерахованих умов:

- кількість поколінь (циклів) досягне заздалегідь обраного максимуму;
- вичерпано час на мутацію [14].

Навчання методом зворотного поширення помилки зводиться до підбору значень ваг прямонаправлене нейронної мережі, грунтуючись на принципах найшвидшого спуску, один з основних недоліків цього класичного алгоритму полягає в можливому попаданні в локальні мінімуми функції вартості. Застосування генетичного алгоритму дозволяє уникнути цієї небезпеки.

Генетичні алгоритми для навчання нейромережі застосовуються як альтернатива методу зворотного поширення помилки. Навчання методом зворотного поширення помилки зводиться до підбору значень ваг прямонаправленное нейронної мережі, грунтуючись на принципах найшвидшого спуску. Один з основних недоліків цього класичного алгоритму полягає в можливому попаданні в локальні мінімуми функції вартості.

3 РЕАЛІЗАЦІЇ ПРОЕКТУ ГРИ «МОНОПОЛІЯ»

3.1 Опис основних підходів навчання, типів ботів, правил гри

Є безліч різних підходів в машинному навчанні, в тому числі генетичні алгоритми і нейронні мережі. У даній грі була обрана модель нейронної мережі FeedForward.

Зазвичай нейронні мережі обучаються на підставі тестів – вибірка даних, що складається з вхідних даний в нейронну мережу, і очікують виходять з неї. Але гра зазвичай не детермінованість заздалегідь, наприклад, граючи в «Маріо» треба стрибати в різних місцях, ворожі боти ходять, яких треба оминати або знищувати, а так само є грибочки і монетки, які треба збирати. Тому в таких випадках застосовують генетичні алгоритми в зв'язці з нейронними мережами. Створюється цілий набір нейронних мереж, і кожна з них грає, потім вибіраються найболее кращі і скрещіваються. І далі все згідно навчання з генетики.

«Монополія» відрізняється від ігор, подібного характеру. Основною відмінністю в тому, що це змагальна гра, як шахи, і тут можуть змагатися люди один з одним. Це додає складності.

Зазвичай, при навчанні таких ігор генетика використовується лише на останніх стадіях для кореляції/поліпшення, але ніяк не навчання з нуля. Отже збираються тести на базі безлічі зіграних партій між гравцями. А потім, в якості текстів, нейронці подаються ходи виграв гравця.

Такий підхід нам не підійшов лише з тієї причини, що просто немає часу проходити ці ігри. Для цього зазвичай залучається велика кількість людей, які паралельно грають, нарощуючи базу. Плюс, це повинні були досить скілові гравці в монополію. Тому ми вибрали навчати генетично і ось які етапи ми пройшли, для того, щоб реалізувати бота для Монополії. Почнемо з того, що ми урізали правила Монополії, тому що реалізації повноцінного бота,

який враховує всі тонкощі гри, було б дуже складно. Плюс, в реально «Монополії» існує ряд речей, які привносять ентропію – тобто рандомний, наприклад, бонусні картки.

Тому гра була спрощена, і ось які правила: існують два типи карток – бонусні, які просто дають гроші за те, що гравець став на цю картку, і підприємства, які можна купувати і продавати.

У кожної картки-підприємства є ціна, а від неї залежать рента, продаж і перепокупка. Так само є завжди додатковий гравець – банк, у якого немає ліміту по грошах.

Гравець, потрапляючи на картку має всього 2 дії, щодо того, кому належить картка. Якщо картка належить банку, гравець може або заплатити ренту в банк (зазвичай це 20% від ціни підприємства), або купує.

Якщо вона належить цьому ж гравцеві, то він або збирає ренту з підприємства, або продає підприємство банку (зазвичай за пів ціни від вартості підприємства), і нарешті, якщо підприємство належить іншому гравцеві, гравець або платить ренту іншому гравцеві, або викуповує у нього підприємство (зазвичай воно в 1.5 рази більше коштує, ніж початкова ціна). Гравці по колишньому кидають кубики і роблять крок згідно з ними.

Генетично в роботі створювали 100 примірників, і кожен грав партію з іншими. Тепер треба було зрозуміти, за якими критеріями обирати успішність бота.

Спочатку бот вибирався той, який найбільше зібрав грошей з усіх ігор. Це не дало належного результату. Потім ми ввели ще один параметр – кількість перемог. І ми хотіли знаходити бота, який і збирає більше всіх, і виграв більше всіх. В кінцевому підсумку це не дало результатів.

Проаналізувавши, було прийнято рішення рішення – вважає більш успішного того, хто зробив найбільше ходів. Так, хоч і умовою програшу вважає баланс, який нижче-одно 0, той грає довше як видно краще розуміє як грати. Цей варіант дав куди краще результати, але вони надостаточность для нас.

Ці всі варіанти не дали результату, і ось чому:

− багато що залежить від гральних кісток, які вносять рандом в гру;

− немає чіткої послідовності і логіки, як наприклад в шахах;

− якщо перший бот переміг, це може означати не тільки, що він хороший, а то, що його противник достоточно поганий.

Ну звичайно ж не виключений варіант, де 1-й бот, виігрвиает 2-го, 2-й виграє 3-го, а 3-й перемагає 1-го. Таким чином вони балансують і все три недостатньо гарні. Так само, генетичної моделі складно, адже їй потрібно не тільки логіку визначити, тобто як грати щоб вигравати, а й як грати зовсім, тобто вивчати правила.

Ось саме тому, спочатку нейронкі навчають тестами. Наступним рішенням було створити бота, який прораховував кілька кроків наперед. Тобто він заздалегідь міг знати як випадуть кістки, і перебирав абсолютно всі варіанти своїх можливих ходів і гравця. А потім вибирав найбільш кращий результат.

В роботі було створено два таких бота, один з них передбачав на 20 кроків вперед, другий на 15. Потенційно другий був слабшим, він не міг бачити далі ніж перший, тому зазвичай програвав, але вони вже могли грати. Боти розуміли правила і прагнули виграти.

На основі їхніх ігор були зібрані чатина тестів. І ще частина була зібрана граючи з цими ботами безпосередньо. як виявилося, це було достатчно, щоб змусити бота імітувати поведінки гравця.

Ітелект бота, а саме його нейронна мережа складаються з вхідного, вихідного і ще двох прихованих шарів. Кількість нейронів на вхідному шарі розраховується за формулою 1:

$$InnerNeuronsCount = fieldItemsCount * 3 + 2 \quad (1)$$

Де fieldItemsCount – кількість елементів на поле, тобто підприємства та бонусні осередки.

Кількість елементів множиться на три, бо кожен набір елементів відповідає за свої характристики.

Перший набір відображає ціни на підприємства. Другий набір стан осередку/підприємства (1 – якщо свій осередок, -1 – якщо противника, 0 – якщо нейтральна, тобто належить банку).

І останній набір відповідає за поточну позицію, так само має три значіенія: 1 – своя поточна позиція, -1 противника, 0 – ніким не зайнята осередок.

В кінці форули вар 2-ка, тобто ще 2 нейрона, одні нейрон показує, чи варті гравці на одній клітинці, а другий за поточний баланс гравця. Цих параметрів повинно бути достатньо для того, щоб здійснювати обмірковані крок – бот бачить абсолютно весь стан поля – де який будинок і де хто знаходиться.

Приблизно було підібрано кількість нейронів на прихованих шарах. Їх 1000 і 100 відповідно. І нарешті останній шар має всього один нейрон, який говорить яку з двох дій зробити. За обидвом правилам всі ваги заполняються значеннями в межах [-0.5; 0.5].

Функція активації обрана гіперболічний танген, він подібний сигмоид (однієї з найбільш розповсюджений функцій активації), але варіуется в межах [-1; 1] замість [0; 1].

Приклад налагоджувальної інформації представлено на рис. 9. Тут позначення R-> і B-> показують хто робить хід. P0 і P1 це гравці, у них відображена інформація про поточний баланс, а так само позиція на полі. [Pr: 50 Side: N] і подібні блоки показують поточний стан поля – ціна і поле відповідно. Це більш докладна інформація про те, як проводиться гра між гравцями.

Але з огляду на, що у нас 100 гравців, і кожен повинен зіграти одну гру з іншими, то це занадто багато інформації. Є більш узагальнена інформація про одну симуляцію в цілому.

«Iterations» показує кількість ітерацій які були зроблені в іграх. Відпо-

відно, «max» – це максимальна кількість ітерацій за все ігри, а aver це середнє значення між усіма іграми.

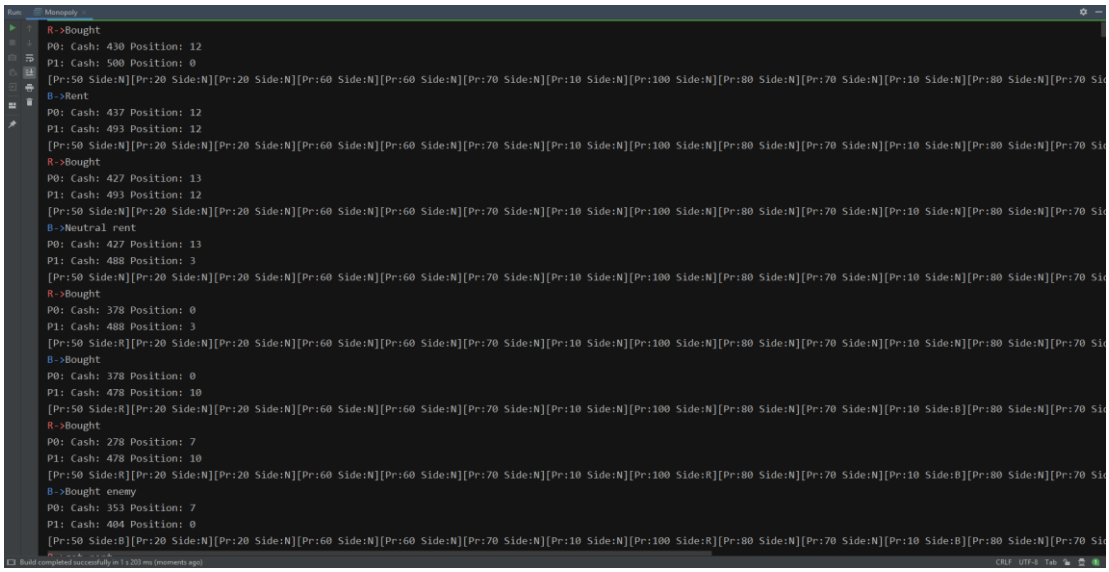


Рисунок 9 – Приклад налагоджувальної інформації

Далі параметр «Simulation» відображає кращого в симуляції гравця (рис. 10).

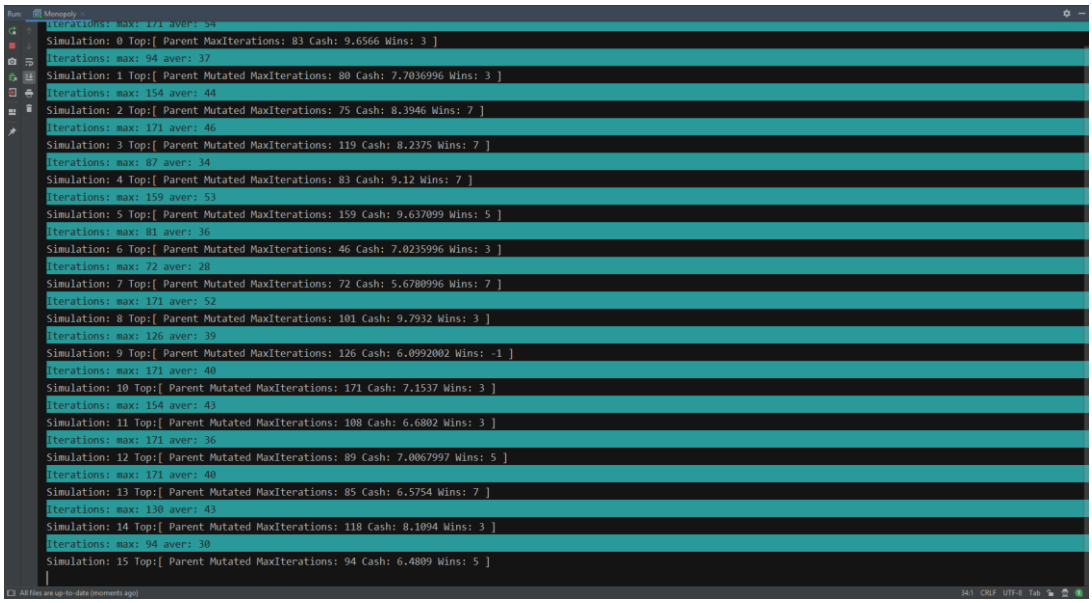


Рисунок 10 – Приклад інформації щодо кращого гравця

Перші пару слів говорять про те, хто є переможцем в генетичному сенсі. В даному випадку ми бачимо «Parent Mutated», це означає, що цей гравець був у попередній симуляції він прісутвовал, а в поточній мутував і зайняв топове місце.

Однак «Parent» в топі симуляції це не до добра, так як прогресу особливого немає. «MaxIterations» відображає максимальний набір ітерацій для поточного гравця, а так само соответсвенно його баланс за все ігри і число перемог. На цих скронах відображена статистика, коли ціллю бота був якомога більший баланс, при чому за всі ігри.

На рис. 1 відображено приклад статистики ігор, коли цілю було саме виграти, байдуже якої баланс при цьому був. Можна помітити, що перемоги в симуляції стали менше скакати, і утримувалися десь у 7.

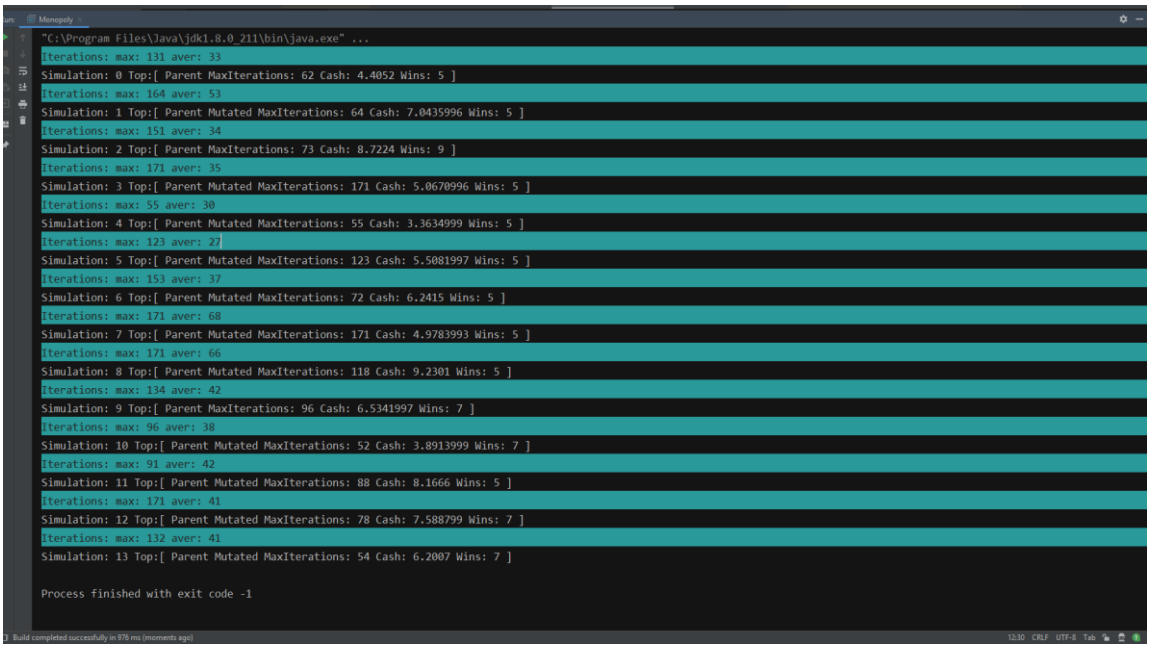


Рисунок 11 – Приклад виводу статистики

І тут приходить думка, що якщо все боти однаково хороші, тоді співвідношення перемог і проиграшей буде не настільки велика. А тому важливою метою має бути те. наскільки вони довго грають. Якщо розглядати ме-

ту перемогу, то кожен прагне перебити всіх, і при цьому ідеальний випадок, це коли все боти погані в поколінні, а лише один кращий. Але такого бути практично не може, а тому було прийнято інше рішення. І тепер мета це якомога довше грати. І якщо звернути внімаеніе, якщо всі будуть хороші, то тільки всі від цього виграють – легше буде відсіяти мене довго грають, тобто гірших.

На рис. 12 явно видно, що результати покращилися, але все ж вони скакали, і хоч це було найкраще з запропонованих результатів, все ж воно не задовольняє умовам. Тому і було прийнято рішення зробити лінійного бота для того, щоб з ним зіграти і на підставі цієї вибірки даних навчити нейронну мережу.

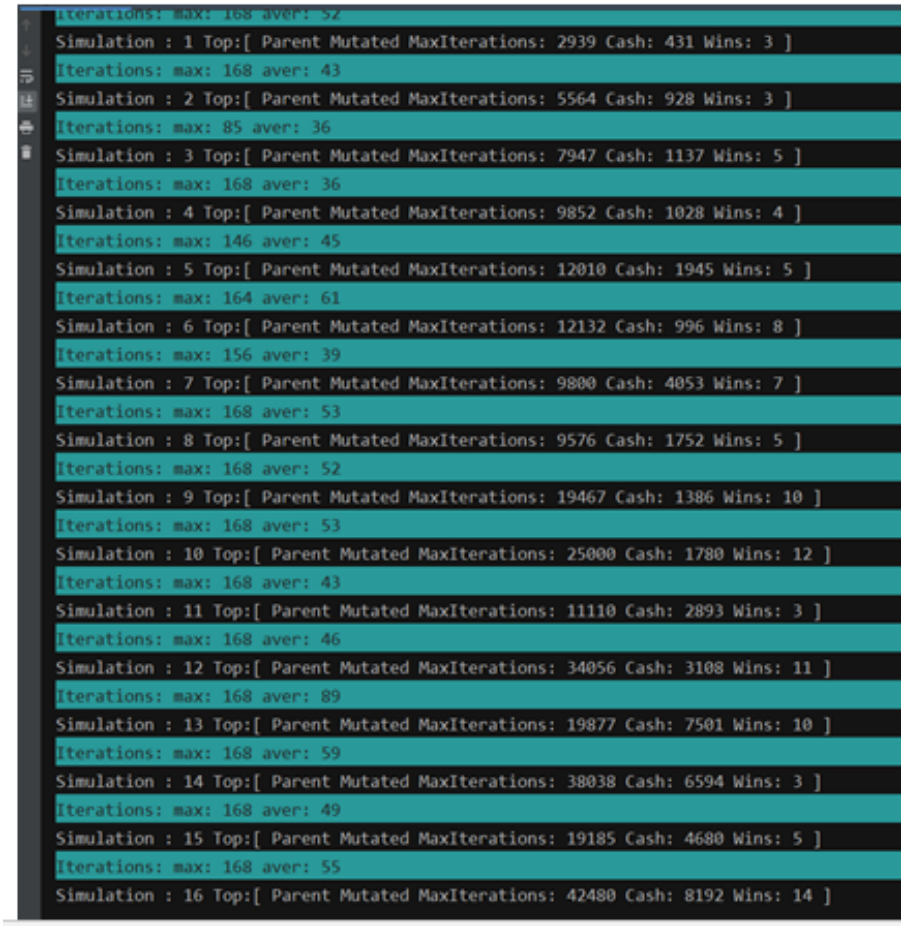


Рисунок 12 – Приклад інформації «успіхів» гравця

3.2 Архітекртура коду. Платорма для навчання

Для бота була реалізована нейронна мережа типу «FeedForward». Так само няряду з нею була розроблена абстракція, для генетичного навчання.

Починаємо з інтерфейсу «IPopulation», він має 4 методу:

- nextGeneration – формує нову генерацію;
- determineTop – визначає кращого в популяції;
- fillTops – заповнює масив кращими гравцями в популяції (від кращого до гіршого);
- evaluateTotalSuccess – повертає загальний успіх популяції.

«IChromosome» абстракція для хромосоми, тобто для гравця в даному проекті:

- метод «Compare» приймає на вхід іншу хромосому для порівняння. Саме завдяки цьому методу відбувається селекція, адже з його допомогою можна порівняти дві хромосоми;
- «getSuccess» повертає успіх хромосоми;
- «resetState» скидає стан хромосом і готує її до наступної симуляції;
- «getChromosomeState» повертає стан хромосоми. Їх є всього 4 – предок, мутований предок, нащадок і мутований нащадок;
- метод це getGens, який вирощує набір генів хромосоми.

«IGen» є абстракцією для гена хромосоми і ось набір властивостей, властиві генам:

- «getItemCount» показує скільки стркутрних одиниць місце ген;
- «getItem» повертає одну структурну одиницю згідно з індексом;
- «setItem» – встановлює значення структурної одиниці.

У загальних рисах ієрархія зрозуміла. Тепер розглянемо більш детально логіку навчання. Почнемо з класу «Monopoly». Цей клас успадковує ASimilarityThresholdPopulation. Це означає, що у нашій популяції веде отслежіваеніе подібності.

Справа в тому, що в генетиці є одна проблема, це коли дуже подібні або зовсім однакові хромосоми займають всю популяцію, тоді немає розвитку, тому що помомкі нічого не відрізняються від предків. Тому при селекції отбрасиваються подібні і дає шанс більш поганим ботам, але не схожим з більш топовими.

Це робить в надії, що гени більш поганих зможуть при скрещевані поліпшити гени більш хороших ботів і дати більш сильне спадок.

Розглянемо конструктор:

```
public Monopoly(){
     super( true,
          new ChromosomeCreator<>(
               new  SpinGenerator.Builder<>().setState((int)
System.currentTimeMillis()).build(),100,
               MonopolyNeuralPlayer.class,
               Monopoly::createNewPlayer,1,
               MonopolyNeuralGen.class,
               Monopoly::createNewGen),
          new GenContainer(new GenInfo(-1, 1)),
          new OneByOneCrossovering<>(),
          new Mutation<>(
               new  SpinGenerator.Builder<>().setState((int)
System.currentTimeMillis()).build(),
               100),0.9f);}
```

Отже, перший параметр true говорить про те, що успіх у нас буле вважатися за спаданням, тобто чим більше значення успіху тим краще. В даному випадку нас цікавить кількість ходів за гру – чим більше тим краще.

Другий параметр – це числова генератор, іншими словами рандом. Третій параметр – це число хромосом в популяції, їх 100. Раніше я згадував, що гравців у нас 100.

Четрвёртий – тип гравця, який буде використовуватися, п'ятий це функція для створення гравця. Шостий параметр – кількість генів. В даному випадку у нас один ген – ціла нейронна мережа. А структурні одиниці цього гена – будуть всі зв'язки нейронної мережі.

Сьомий параметр – тип гена, восьмий – функція для створення гена. Дев'ятий параметр – контейнер гена. Тут зберігається інформація про всі

стректурних одиницях конкретного гена, а саме мінімальні і максимальні межі структурних одиниць. В даному випадку ваги всіх зв'язків вар’їруються в межах від -1 до 1.

Десятий параметр – обробник скрещення. В даному випадку ми використовуємо метод "один-за-одним". Одинадцятий параметр – обробник мутації. Для мутації тут так само зазначений генератор рандомних, а так само кількість змінюваний структурних одиниць гена, в даному випадку лише 100 ваг буде змінюватися. І останній параметр позначає рівень подібності при селекції. Мається на увазі, якщо хромосома подібна до вже раніше вибраних хромосомам на 90% або більше, така хроосома в селекцію не входить.

Далі показаний код, який відображає логіку симуляції: встановлюємо перемикач гравців. він вказує який з гравців в даний момент робить крок:

```
boolean activeFirstPlayer = true;
for (int iteration = 0; ; ++iteration){
```

Оголошуємо вічний цикл, умова зупинки якого прописано нижче. Він так само вважає ітерації.

Отримуємо рандомно сгенерированное значення кроку. Можна помітити, що генератор поверне значення в межах від 0, до 10 включно, а тому треба додати ще 2, щоб результат коливатися від 2 до 12 – це якраз стільки могет викинути гральні кістки.

```
int step = stepGenerator.nextInt(11) + 2;
```

Повідомляємо двох гравців – один активний, інший ворожий. А також визначаємо хто з двох гравців на даній ітерації яку роль приймає.

```
SimulationPlayer active; SimulationPlayer enemy;
     if (activeFirstPlayer){
          active = simPlayer_0;
          enemy = simPlayer_1;
     }
else {
```

```
active = simPlayer_1;
enemy = simPlayer_0;}
```

Виводимо інформацію про крок. Викликаємо у активного гравця метод makeStep для того, щоб переміститися на нову клітку і зробити дію відносно даної клітини.

```
ColoredConsole.print(activeFirstPlayer ? "R->" : "B->",
activeFirstPlayer ? AnsiForegroundColor.RED :
AnsiForegroundColor.BLUE);
active.makeStep(step, buildingsList, enemy);
```

Перемикаємо перемикач, щоб на наступний ітерації гравці помінялися ролями. Далі як раз умова закриття вічного циклу – якщо хоча б один з гравців банкрут або кількість ітерацій одно максимально допустимому, тоді ініціювати процес закриття циклу. Перевірка на ітерації зроблена з метою, щоб уникнути вічної гри. У теорії два цілком хороших гравця можуть грати між собою вічно або досить довго.

```
if (simPlayer_0.isBankrupt() || simPlayer_1.isBankrupt() ||
iteration == Monopoly.MAX_SIMULATION_ITERATIONS){
```

Реєструємо кількість ітерацій в даній грі. По завершенню симуляції все вони будуть аналізуватися для отримання середнього значення і максимального. Далі йде вивод інформації про станах двох гравців та виведення інформації про поле

```
iterations.push(iteration);
//Применение результатов игры для обоих игроков и закрытие
цикла
simPlayer_0.applySimulationResults(iteration);
simPlayer_1.applySimulationResults(iteration);
break;}
ColoredConsole.println ("P0:" + simPlayer_0 + "\ nP1:" +
simPlayer_1);
ColoredConsole.println (buildingsList.toString ()); }}
```

Далі описана логіка здійснення ходу гравцем: отже, метод приймає крок на який гравець повинен зміститься, ігрове поле і ворожий гравець.

```
public void makeStep (int step, BuildingsList buildingsList,
SimulationPlayer enemy) {
```

Локалізуємо змінну боку поточного гравця, а так же масив підприємств і їх кількість:

```
Side currentSide = this.side;
Building [] buildings = buildingsList.buildings;
final int BUILDINGS_COUNT = buildings.length;
```

Визначаємо нову позицію. Позиція не повинна перевищувати або дорівнювати кількість будівель, так як індескація в масиві йде з 0. Якщо все ж перевищує, то зменшувати поціцію до тих пір, поки вона не задовольнить умовам.

```
int position = step + buildingsPosition;
if (position> = BUILDINGS_COUNT) {
addCash (Monopoly.CASH_PER_FIELD_OVERSTEP);
while (position> = BUILDINGS_COUNT)
position - = BUILDINGS_COUNT;}
this.buildingsPosition = position;
```

Локолізуємо нейронну мережу та отримуємо масив вхідних значень і заповнюємо їх нулями:

```
Network network = monopolyPlayer.getGens () [0] .network;
float [] networkValues = getNetworkValues (BUILDINGS_COUNT *
3 + 1);
Arrays.fill (networkValues, 0);
```

Локалізуємо максимальну ціну. Вона нам знадобиться для масштабування цін в межах від 0 до 1:

```
int maxPrice = buildingsList.maxBuildingPrice;
```

Повідомляємо цикл для обходу всіх підприємств та виставляємо ціну пріедпріятія з маштабірованних в вище зазначених межах. Далі визначаємо приналежність підприємства і в залежності від цього вносимо в дані -1, 0 або 1.

```
if (side == Side.NEUTRAL)
sideValue = 0;
else
sideValue = currentSide == side? 1: 1;
networkValues [i + BUILDINGS_COUNT] = sideValue;}
networkValues [(BUILDINGS_COUNT << 1) + position] = 1;
networkValues [networkValues.length - 1] = currentCash /
(float) Monopoly.MAX_GAME_CASH;
```

Далі йдуть три умови, тут ми визначаємо як трактувати відповідь нейронної мережі. Умови формують три секції обробки – в разі, якщо банк, підприємство противника, або підприємство під контролем гравця.

У кожному такому блоці розглядаються два варіанти. В даному випадку, якщо результат менше 0, то це трактуєте як покупка підприємства у банку:

```
if (results [0] <0) {
int price = currentBuilding.price;//Локалізуємо ціну на
підприємство
addCash (-price);//Знімаємо з балансу гравця ціну на
Преприятие
currentBuilding.side =currentSide;//присвоюємо сторону
підприємства суміжну з гравцем та виводимо на екран
інформацію про покупку
ColoredConsole.println ("Bought");}//інакше трактуємо як
плата ренти банку
else {
```

У гравця знімаємо з балансу ренту і виводимо інформацію про плату ренти:

```
addCash (-currentBuilding.neutralRent);
ColoredConsole.println ("Neutral rent");}}
else if (currentBuilding.side! = currentSide) {
```

Якщо результат менше 0 це означає плата ренти противнику, то локалізуємо ренту на підприємство.

3.4 Діаграми класів

Далі розглянемо дві UML-діаграми, що відображають діаграму класів програми навчання ботів для гри «Монополія». Перший більш великий (див. додаток А), що показує всі елементи класів, другий відображає чисто властивості (рис. 13).

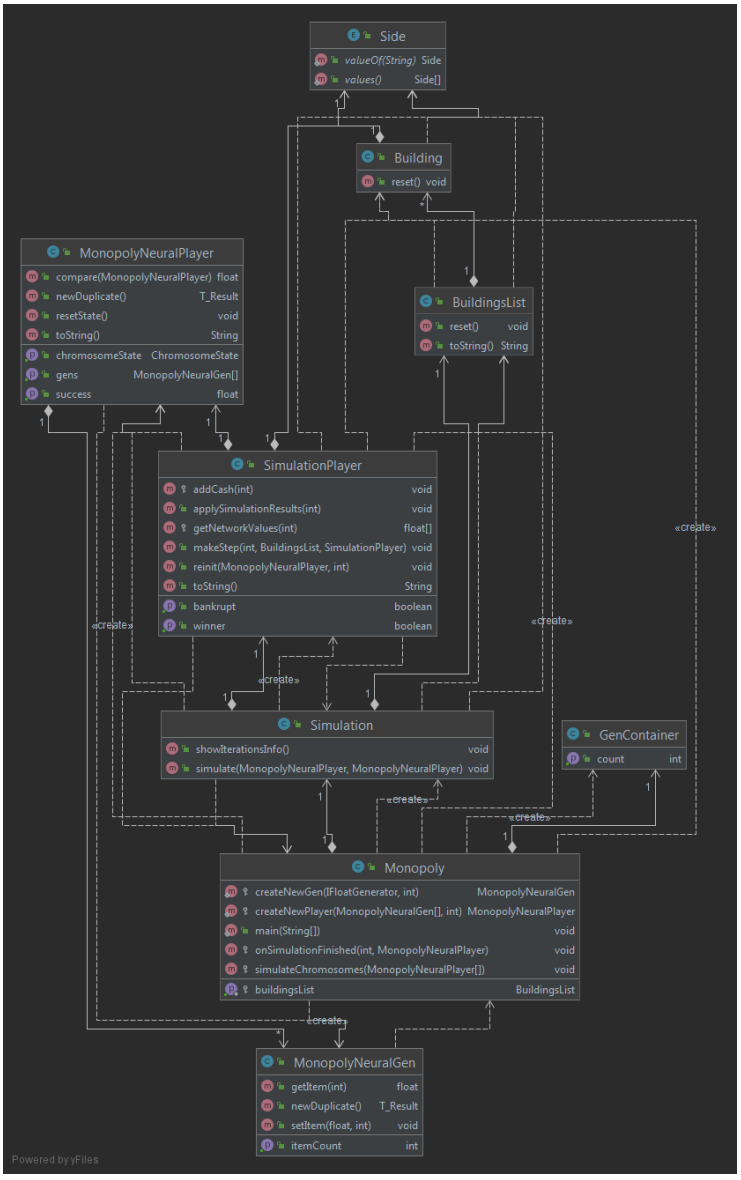


Рисунок 13 – Діаграма класів, відображення властивостей

3.5 Архітектура гри

Розглянемо базовий архітектурні пакети, на основі яких буде видно модулі системи (рис. 14).

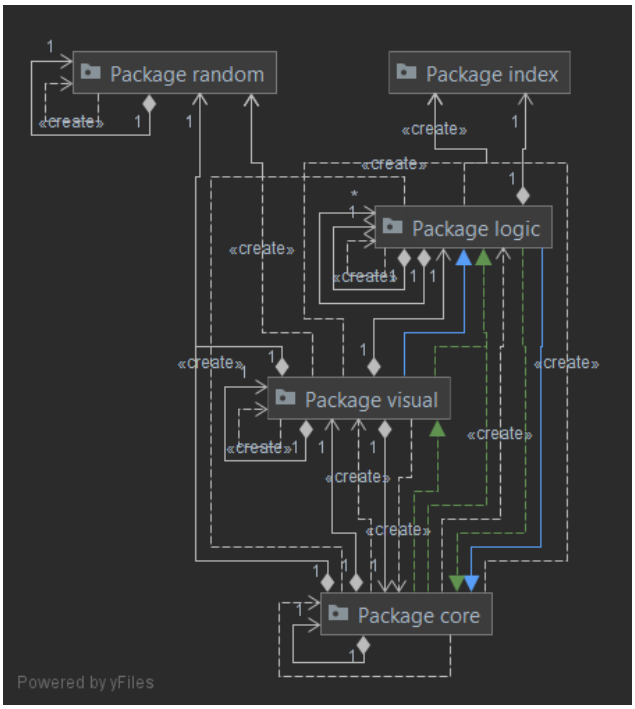


Рисунок 14 – Діаграма пакетів

Отже, пакет «index» складається з класів, які реалізовують абстракцію «AIndex». Цей абстрактний клас призначений для індексованих значень. Коротко про його поведінці: цей метод дає змогу зрозуміти, чи індекс зациклениим.

isCycled () – метод дозволяє виставити зацикленість індексу. Ми використовували цей функціонал в ядрі гри. Кожен крок ми повинні були вибрати наступного гравця в списку, щоб обробити його крок.

Щоб кожен раз не перевіряти, чи є даний гравець останнім у списку, і якщо так, то на наступному кроці почати з першого в списку, ми просто включили зацикленість індексу. Це позбавляє нас від нагромадження коду.

setCycled(boolean cycled) – метод, який повертає стартове значення проміжку, на якому працює індекс. Метод абстрактний, так як логіка визначення проміжку в індексі може бути складною, більш того, вона може бути динамічною. Так що часом тут не обійтися просто одним значенням в полі класу. Тому реалізатор сам визначає цю логіку.

getStart() – той же метод, що і getStart, тільки для отримання кінцевого значення в проміжку.

getEnd() – метод зсуву індексу від початкового значення до кінцевого. Якщо Ідекс далі рухатися не може і досяг кінцевого значення, то метод поверне false, інакше true:

next() – аналогічний методу next, тільки зрушує індекс від кінцевого значення до початкового. Так само, досягнувши початкового значення, поверне false, інакше true.

getCurrent – цей метод призначений для скидання стану індексу на конкретне. Якщо входить значення більше кінцевого значення індекс в результаті буде скинутий на кінцеве, а якщо менше стартового, то на стартове.

Пакет «random» призначений для роботи з рандомних числами. У нас як мінімум кістки працюють на даному функціоналі. Java вже має засоби роботи з випадковими числами, але інтерфейс роботи з ними не настільки великий. Плюс, відсутня робота з станами генераторів.

Пакет «logic» (див. додаток Б) містить всю логіку гри, тут реалізовані ігрове ядро, а також ігрові елементи. Коротенько про кожного з них. «IStatable» це узагальнений інтерфейс, який дає можливість копіювати стан одного об'єкта, іншому. Гравці, ігрові елементи і навіть сама логіка гри реалізовують даний інтерфейс. Наступний клас «APlayer» – абстракція гравця. Розглянемо його:

Для початку можемо звернути увагу, що гравець реалізовує «IStatable», як і було сказано раніше. Як властивості гравець має ім'я, сторону, позицію на ігровому полі, а також композиційну зв'язок на абстракцію логіки, так як гравець не може існувати сам по собі.

Наступні три методи сформовані для отримання результатів в залежності від стану осередку поля, на яку гравець потрапив. Згідно з правилами, у нас є три випадки – нейтральна осередок (належить банку), осередок гравця, осередок противника. Повертають ці функції енамі, кожен з яких має два значення, згідно з правилами гри.

```
    public abstract NeutralCellResult onNeutralCell (ACell
cell);
    public abstract AliasCellResult onAliasCell (ACell cell);
    public abstract EnemyCellResult onEnemyCell (ACell cell);
```

Цей метод дозволяє переводити гравця в режим очікування кроку. Для реального гравця розблокується візуальний інтерфейс, щоб той міг кликати на кістки щоб зробити крок, а для бота це ініціює процес ходу.

getBalance() – метод призначений для очікування кроку гравця. Вище були описані колбекі, які повертають результати щодо того, куди перемістився гравець, але для початку його необхідно перемістити. Для бота кидання костей це автоматична операція його логіки, для реального гравця ядро має очікувати, поки реальний гравець не натисне на візуальний компонент "кинути кістки", і лише після цього робити якусь обробку. Side це перелічення – перераховує весь набір сторін в грі. В даному випадку, є всього лише три сторони – червона, синя і нейтральна.

ACell – абстракція осередки на поле. На даний момент, у нас є два типи осередків – бонусні, які належать тільки банку, і підприємства, який належать банку спочатку, але можуть бути куплені.

Осередок так само реалізує IStatable.

```
    public abstract class ACell implements IStatable <ACell> {
```

З властивостей осередок має композиційну зв'язок з полем, а так само посилання на гравця-власника.

```
    protected final Field field;
    protected APlayer owner;
```

Ось тіло методу, який копіює стан осередку в даний момент. Локалізуємо гравця-власника осередку, чий статок ми будемо копіювати.

```
public void copyState (ACell cell) {
APlayer passedOwner = cell.getOwner();
```

Дана ланцюжок викликів націлена по отримання гравця-власника, аналогічного тому, який зараз знаходитися в оригінальній осередку. Цей осередок може перебувати в абсолютно іншому полі і з зовсім іншими гравцями, але сторони зберігаються, тому ми можемо знайти відповідного гравця по його стороні.

```
APlayer candidate = getField()
.getLogic()
.getPlayers()
.findBySide(passedOwner.getSide());
```

Якщо гравець не був знайдений, то викидаємо помилку про те, що відповідного гравця не знайшли, а інакше до цього вічка присвоюємо нового власника.

```
if (candidate == null)
throw new RuntimeException ( "There's no suitable player");
else
setOwner (candidate);}}
```

Так само розглянемо один з реалізоторов осередки – підприємство. Як і було сказано раніше, підприємство це осередок, яку можна купувати і продавати. Також, гравці можуть збирати ренту або платити її іншим гравцям.

```
public class Industry extends ACell {
```

Серед властивостей підприємства можна виділити ціну на покупку у банку.

Крім цього, рента, ціна на продаж, а так само ціна на перепокупку будівлі у противника.

```
public final int buy;
public final int rent;
public final int sell;
public final int enemyBuy;
```

Цей метод призначений для отримання вартості будівлі для покупки. Як можна помітити, в залежності від того, хто власник змінюється і вартість на будівлю. Якщо власник банк, то повертаємо оригінальну вартість будівлі, інакше повертаємо ціну на перепокупку в іншого гравця.

```
public int getPrice () {
if (getOwner (). getSide () == Side.NEUTRAL)
return buy;
else
return enemyBuy;}}
```

Інтерфейс IDices призначений для абстракції грального кісток, так як по суті грального кісток може бути кілька з різними настройками і різною логікою. Відразу ж бачимо, що кістки також реалізують інтерфейс IStatable.

```
public interface IDices extends IStatable <IDices> {
```

Даний метод призначений для кидання костей. На вхід йому приходить потік для читання рандомних значень. Цей абстрактний клас якраз з пакета random, він має багатий і зручний інтерфейс з отримання випадкових чисел. Даний метод повертає значення на одній з грального кістки. Номер кістки визначаємо за індексом.

```
void roll (ARandomStream stream);
byte getDiceValue (int index);
За допомогою цього методу дізнаємося скільки кісток всього.
int getCount ();
```

Наступним методом можна встановити значення на заданій кістки.

```
void setDiceValue (int index, byte value);
```

Так само є метод, який дозволяє повернути масив значень кісток.

```
byte [] getDicesValues();}
```

У конструкторі вказуємо кількість кісток У грі ми використовуємо дві кістки. Створюємо масив на задану кількість елементів і заповнюємо масив значення 1, так як за замовчуванням джава заповнює всі масиви значенням 0.

```
public Dices (int count) {
this.dicesValues = new byte [count];
Arrays.fill (dicesValues, (byte) 1);}
@Override
public byte getDiceValue (int index) {
return dicesValues [index]; }
```

Повертаємо розмір масиву, це і буде кількість кісток.

```
@Override
public int getCount () {
return dicesValues.length;}
```

Визначаємо нове положення гравця на поле. Тобто ми до сучасному стану гравця додаємо суму всіх значень на всіх кубиках. Якщо дана кількість буде більше, ніж елементів на поле, то ми просто віднімаємо від позиції кількість елементів на поле, щоб відобразити гравця з початку поля. Також додаємо в баланс даному гравцю за прохід поля повністю 200 пунктів.

```
int position = player.getFieldPosition () +
ArrayManager.getSum (dicesResult);
if (position> = field.getCount ()) {
position - = field.getCount ();
player.addCash (200);}
```

Застосовуємо нову позицію гравця та отримуємо осередок, яка зараз перебуває під гравцем, і в залежності від того, кому вона належить визначаємо дії. Якщо власник це і є наш гравець, значить осередок своя. Тому викликаємо метод, який обробляє випадок, коли гравець знаходиться на своєму осередку:

```
setPlayerPosition (player, position);
ACell cell = field.get (position);
APlayer owner = cell.getOwner ();
if (owner == player)
playerOnAliasCell (player, (Industry) cell);
```

Якщо власник має нейтральну сторону, це означає осередок належить банку. Тому викликаємо метод, який обробляє випадок, коли гравець знаходиться на осередку банку, інакше, осередок належить якомусь іншому гравцеві. Тому викликаємо метод, який обробляє випадок, коли гравець знаходиться на осередку противника.

```
else if (owner.getSide () == Side.NEUTRAL)
playerOnNeutralCell (player, cell);
else
playerOnEnemyCell (player, (Industry) cell);// деактивувавши
крок гравця.
player.deactivateToStep (); // визначаємо наступного гравця.
APlayer nextPlayer;
```

Визначаємо цикл, який крутиться до тих пір, поки поточний гравець буде нейтральним. Нейтральні гравці є службовими і не беруть участь в грі здійснюючи кроки.

```
do {
playerIndex.next ();
nextPlayer = players.get (playerIndex.getCurrent ());}
while (nextPlayer.getSide () == Side.NEUTRAL);
nextPlayer.activateToStep ();}//Активуємо нового гравця.
```

Розглянемо ще декілька методів. Гравець збирає ренту зі свого підприємства:

```
playerGetsRent (player, industry);
break;
case SELL: // Гравець продає власне підприємство
playerSells (player, industry);
break;
//Інакше викидаємо помилку, тому як інші варіанти не передбачені ігровим ядром
default:
throw new RuntimeException ( "Unexpected result:" + result);}}
```

Метод для збору ренти включає у себе отримання вартість ренти з підприємства

```
protected void playerGetsRent (APlayer player, Industry industry) {
int rent = industry.getRent();
```

Метод для продажу підприємства: спочатку отримуємо вартість продажу підприємства, далі додаємо суму продажу в баланс гравця та виставляємо новим власником підприємства банк. Після знімаємо з балансу у банку вартість продажу підприємства:

```
protected void playerSells (APlayer player, Industry industry) {
int sell = industry.getSell ();
player.addCash(sell);
industry.setOwner(players.findBySide (Side.NEUTRAL));
industry.getOwner(). addCash(-sell);}
```

Метод для обробки результатів на осередку банку: перевіряємо, якщо осередок є бонусної, то викликаємо обробник бонусної осередки:

```
protected void playerOnNeutralCell (APlayer player, ACell cell) {
if (cell instanceof BoundCell)
```

```
    playerOnNeutralBoundCell (player, (BoundCell) cell);
    else
    playerOnNeutralIndustry (player, (Industry) cell);}
```

Метод для обробки ситуації, коли гравець перебувати на бонусної клітці: додаємо в баланс гравцеві бонус:

```
    protected void playerOnNeutralBoundCell (APlayer player,
BoundCell cell) {
    player.addCash (cell.getBound ());}
```

Метод для обробки випадку, коли гравець знаходиться на підприємстві банку:

```
    protected void playerOnNeutralIndustry (APlayer player,
Industry industry) {
    NeutralCellResult result = player.onNeutralCell (industry);
    switch (result) {
    case BUY:
```

Метод для обробки покупки підприємства:

```
    protected void playerBuyCell (APlayer player, Industry
industry) {
```

Отримуємо суму покупки. Сума буде змінюватися в залежності від власника:

```
    int buy = industry.getPrice();
```

У гравця відняти суму покупки

```
    player.addCash (-buy);
```

Нинішньому власнику додати в баланс суму покупки. Призначити нового гравця як власника даного підприємства.

```
industry.getOwner (). addCash (buy);
industry.setOwner (player);}
```

І найголовнішим в пакеті «logic» є клас логіка – «ALogic». Це і є ігрове ядро. Логіка відповідає за всі процеси і взаємодії в грі. Отже, варто зауважити, що логіка так само реалізує IStatable. Серед властивостей логіки можна виділити композиційну зв'язок на поле, гравців і гральні кістки. Так само можна помітити, що є індекс гравців. Про нього говорилося раніше.

Наступний пакет «visual» (рис. 15) відповідає за візуалізацію. Тут перебраний набір візуальних компонентів – картки, гравці, поле, чат, інше. Інтерфейс ISoftBounded призначений для макетів. Джава має набір стандартних макетів, завдання яких розміщувати візуальні компоненти згідно певної стратегії.

Зазвичай, досить складно домогтися, щоб макет якимось чином враховував побажання компонентів, які він розміщує, наприклад, я хочу, щоб компонент був квадратний, тобто ширина і висота були однакові, або якимось чином дотримувалися пропорції.

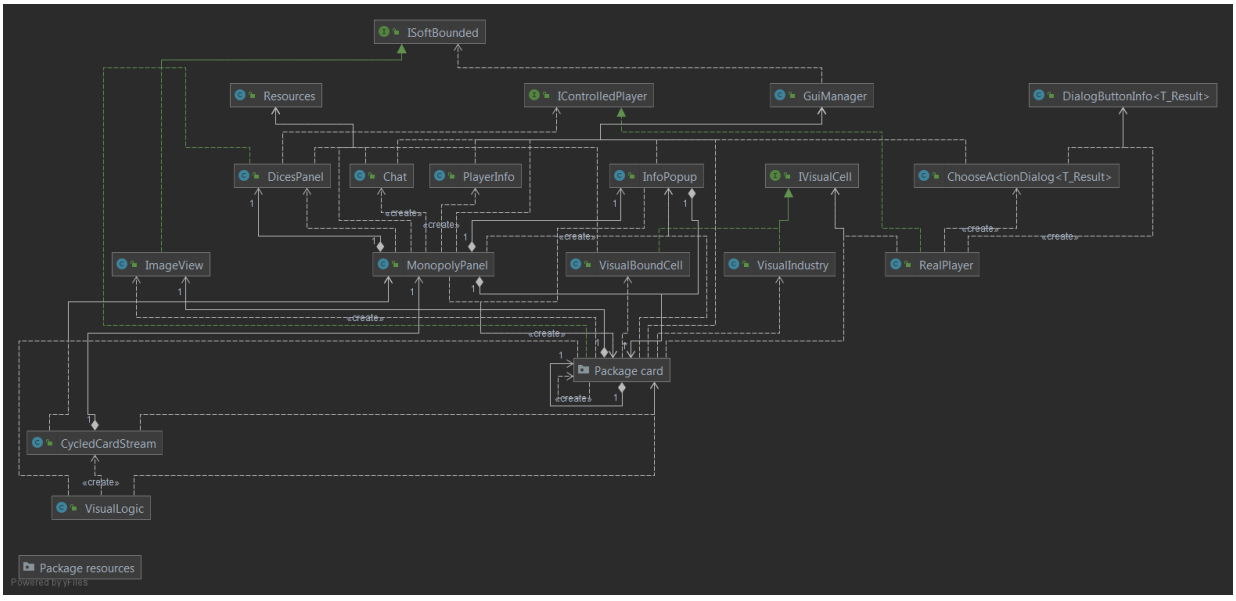


Рисунок 15 – Діаграма пакету Visual

3.7 Опис інтерфейсу гри

Основним «двигуном» в грі є гральні кості. Для даного проекту використовуються два гральних кубіка. Їх приклад та пусте поле представлено на рис 16:

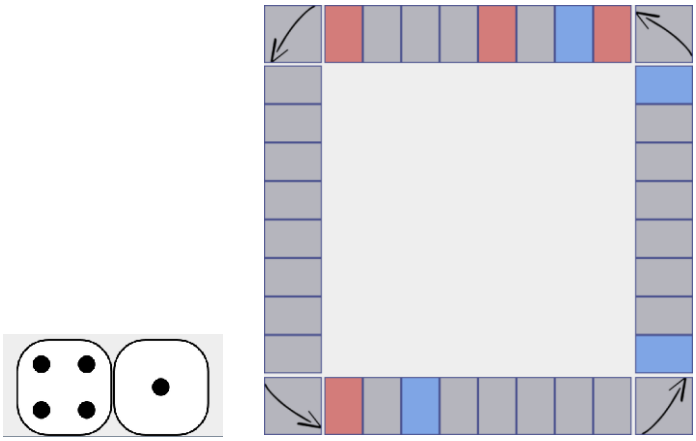


Рисунок 16 – Гральні кубики

Ігрове поле спочатку є пустим, тільки після початку гри на ньому розміщуються підприємства та інформаційні картки і починається гра. Після завершення гри зявляється повідомлення з результатами (рис. 17):

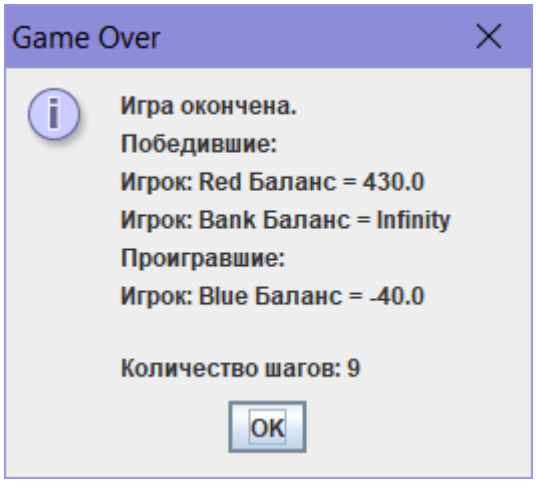


Рисунок 17 – Повідомлення результатів гри

Приклад активної гри представлено на рис. 18.

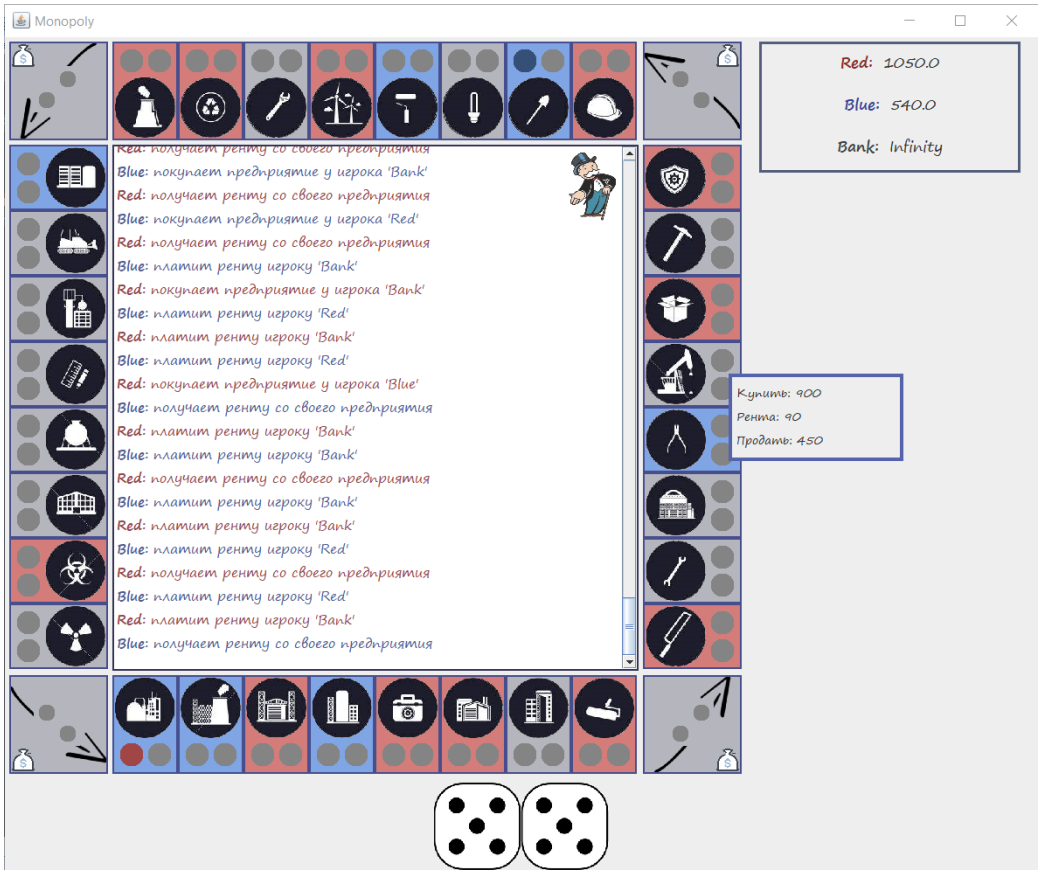


Рисунок 18 – Скрін ходу гри

В середині ігрового поля відображається час гравців та їх дії, як показано на рис. 19:

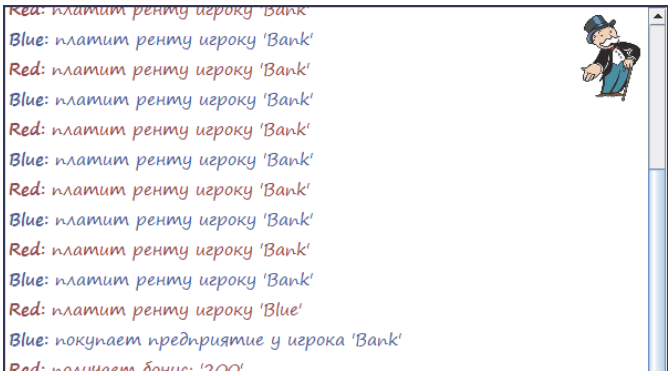


Рисунок 19 – Чат гравців

ВИСНОВКИ

Є безліч різних підходів в машинному навчанні, в тому числі генетичні алгоритми і нейронні мережі. У дипломній роботі реалізовано гру «Монополія» за скороченими правилами. Для гри на базі нейроної мережі створено декілька ботів, які навчаються в ході гри та збільшують свої навички. В роботі була обрана модель нейронної мережі FeedForward.

Взагалі, робота над проектом проходила в декілька етапів. Так, на першому етапі було розглянуто предметну область, виконано аналітичний огляд предметної області і головне розглянуто принципи створення та роботи нейроної мережі за обраним типом.

Проект реалізовано мовою Java, діаграми основних пакетів щодо проекту представлені у описі роботи алгоритму, а також у додатках. Для тестування створено 100 гравців, кожен з яких пройшов гру за різними умовами. Розробляти тести для нейроної мережі займає багато часу, тому успішного боту було вирішено виділити за рахунок кількості отриманих фінансів та часу партії.

Як подальший розвиток гри можна виділити такі доповнення:

- створення більшої кількості гравців для отримання даних тестів;
- доповнення логіки гри повним пакетом правил та умов;
- використання інших алгоритмів для навчання штучного інтелекту.

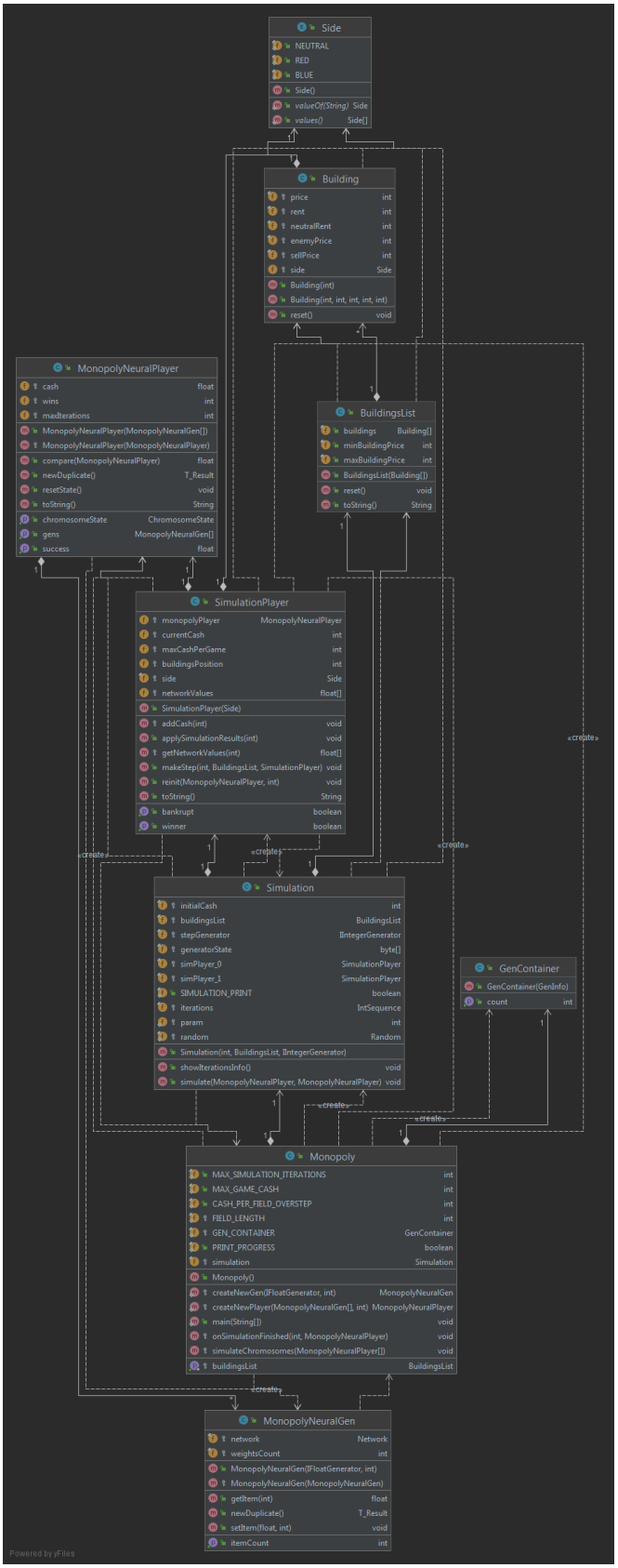
СПИСОК ДЖЕРЕЛ

1. Почему боты в играх не умнеют? URL: https://www.ixbt.com/live/games/pochemu-boty-v-igrah-ne-umneyut.html (дата звернення 10.02.2021).
2. Monopoly. Популярная семейная игра. URL: https://www.marmaladegamestudio.com/games/monopoly/. (дата звернення 20.02.2021).
3. Установка и настройка IntelliJ IDEA. URL: https://geekbrains.ru/posts/intellij_idea_setup. (дата звернення 02.03.2021).
4. Почему IntelliJ IDEA. URL: https://www.jetbrains.com/ru-ru/idea/ (дата звернення 10.03.2021).
5. Java SE . URL: http://java-online.ru/java-basic.xhtml (дата звернення 15.03.2021).
6. Java. URL: http://progopedia.ru/language/java/. (дата звернення 28.03.2021).
7. Библиотека Swing. URL: http://java-online.ru/libs-swing.xhtml (дата звернення 28.03.2021).
8. Шпаргалка по разновидностям нейронных сетей. Часть первая. Элементарные конфигурации. URL: https://tproger.ru/translations/neural-network-zoo-1/ (дата звернення 20.03.2021).
9. Как использовать TensorFlow с Java. URL: https://stackabuse.com/how-to-use-tensorflow-with-java/ (дата звернення 05.04.2021).
10. Что требуется сделать в Java для полноценной поддержки машинного обучения. URL: https://habr.com/ru/company/piter/blog/429596/ (дата звернення 05.04.2021).
11. Установить TensorFlow Java. URL: https://www.tensorflow.org/jvm/install (дата звернення 10.04.2021).

12. Алгоритм машинного обучения Flappy Bird. URL: https://habr.com/ru/post/336612/ (дата звернення 15.04.2021).
13. Нейронные сети, генетические алгоритмы и прочее. Мифы и реальность. URL: https://habr.com/ru/post/321140/ (дата звернення 20.04.2021).
14. Генетический алгоритм. Просто о сложном. URL: https://habr.com/ru/post/128704/ (дата звернення 22.04.2021).

Д О Д А Т К И

Додаток А –
Діаграма класів



Додаток Б –
Діаграма пакету Logic