Національний університет «Полтавська політехніка імені Юрія Кондратюка»

(повне найменування вищого навчального закладу)

\_Навчально науковий інститут інформаційних технологій та робототехніки\_

(повна назва факультету)

\_Кафедра комп’ютерних та інформаційних технологій і систем\_

(повна назва кафедри)

**Пояснювальна записка**

**до дипломного проекту (роботи)**

\_\_\_\_\_\_\_\_\_\_\_\_\_бакалавра\_\_\_\_\_\_\_\_\_\_\_\_\_

(освітньо-кваліфікаційний рівень)

на тему

Створення апаратно-програмних засобів нейронної мережі для прийняття рішень у покроковому Android-додатку

Виконав: студент 4 курсу, групи 401-ТК

спеціальності

\_\_123 Комп’ютерна інженерія \_

(шифр і назва напряму)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_Власенко М. В.\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Керівник \_\_\_\_\_ Альошин С.П.\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Рецензент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Полтава – 2021 року

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ « ПОЛТАВСЬКА ПОЛІТЕХНІКА ІМЕНІ ЮРІЯ КОНДРАТЮКА»**

**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА РОБОТОТЕХНІКИ**

**КАФЕДРА КОМП’ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ**

**КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА**

**спеціальність 122«Комп’ютерні науки та інформаційні технології»**

**на тему**

**«Створення апаратно-програмних засобів нейронної мережі для прийняття рішень у покроковому Android-додатку»**

**Студента групи 401-ТК Власенка Мирослава Валерійовича**

Керівник роботи кандидат технічних наук, доцент Альошин С.П.

Консультант старший викладач, Демиденко М.І.

Завідувач кафедри доктор технічних наук, професор Головко Г.В.

Полтава – 2021

**РЕФЕРАТ**

Кваліфікаційна робота бакалавра: 64с., 14 малюнків, 8 таблиць, 1 додаток, 14 джерел.

**Об’єкт дослідження**: система адаптивного штучного інтелекту для прийняття рішень у покроковому гральному Android-додатку.

**Мета роботи**: розробити нейронну мережу для реалізації штучного інтелекту з механізмами самонавчання.

**Методи**: розглянути подібні системи штучного інтелекту у гральних додатках та проведено аналіз витрат обчислювальних ресурсів, витрат часу, складності реалізації тощо.

Провести статистичний аналіз оцінок, отриманих у експертів, для сформувати базу навчальної вибірки та тактичної моделі поведінки персонажу під керуванням штучного інтелекту (англ. NPC – Non-Player Character). Таким чином, сформовати початкові коефіцієнти синаптичних зв’язків у нейронній мережі.

Розробити систему самонавчання нейронної мережі, спеціально для прийняття тактичних рішень у покроковій грі із чіткою множиною параметрів та їх незмінною кількістю. Ця система дозволить навчити систему ШІ імітувати поведінку гравця та збільшувати складність гри проти ШІ на протязі усього часу його самонавчання.

Ключові слова: штучний інтелект, покроковий гральний додаток, android-додаток, синаптичні зв’язки, нейронна мережа, тактичне рішення, самонавчання, базова навчальна вибірка, тактична модель поведінки, персонаж під керуванням штучного інтелекту, адаптивний інтелект, прецеденти.

**THE ABSTRACT**

Degree work: 64 pages; 14 figures; 8 tables; one additional section; 14 sources. The graphic part of degree work contains 4 posters.

Object – an adaptive system of artificial intelligence for decision making in the turn-based Android game application.

Purpose of the thesis - to develop a neural network to implement artificial intelligence with self-learning mechanisms.

Were considered similar systems of artificial intelligence in games and analyzed the costs of computing resources, time cost, difficulty of implementation, etc.

Was the statistical analysis of the valuations obtained from experts to form the base of training sample and tactical behavior of non-player characters. Thus were formed the initial synaptic coefficients of a neuron network.

A system of self-learning neural network was developed, especially for tactical decision-making in the turn-based game with a clear set of parameters and a constant amount of these parameters. This system allows you to train the system to simulate the behavior of the AI player and increase the difficulty of the game against the AI over the whole period of his self-training.

ARTIFICIAL INTELLIGENCE, TURN-BASED GAMING APPLICATIONS, ANDROID APPLICATION, SYNAPTIC CONNECTIONS, NEURAL NETWORK, TACTICAL DECISIONS, SELF-LEARNING, BASIC TRAINING SAMPLE, TACTICAL BEHAVIOR MODEL, NON-PLAYER CHARACTER, ADAPTIVE INTELLIGENCE, PRECEDENT.

**ЗМІСТ**

[ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ 7](#_Toc74828551)

[ВСТУП 8](#_Toc74828552)

[РОЗДІЛ 1 9](#_Toc74828553)

[СИСТЕМНИЙ АНАЛІЗ 9](#_Toc74828554)

[1.1 Аналіз завдання 9](#_Toc74828555)

[1.2 Аналіз проблемної області 10](#_Toc74828556)

[1.3 Постановка задачі 11](#_Toc74828557)

[1.4 Можливі області застосування системи 11](#_Toc74828558)

[1.5 Обґрунтування і вибір програмної платформи реалізації програмного засобу………………………………………………………………………………..13](#_Toc74828559)

[РОЗДІЛ 2 15](#_Toc74828560)

[СТРУКТУРНЕ ПРОЕКТУВАННЯ ПРОГРАМНОГО СЕРЕДОВИЩА 15](#_Toc74828561)

[2.1 Опис структури даних 15](#_Toc74828562)

[2.2 Розробка алгоритму 24](#_Toc74828563)

[2.3 Аналіз схеми алгоритму і розробка структури ПЗ 27](#_Toc74828564)

[РОЗДІЛ 3 29](#_Toc74828565)

[РОЗРОБКА ПРОКТНИХ РІШЕНЬ ПО ПРОГРАМНОМУ І ІНФОРМАЦІЙНОМУ ЗАБЕЗПЕЧЕННЮ СИСТЕМИ 29](#_Toc74828566)

[3.1 Основні інформаційні частини ПЗ 29](#_Toc74828567)

[3.1.1 Вихідний код програмного засобу. Вихідний код системи прецедентів, підсистеми нейронної мережі, модуля прийняття рішень – в додаткові А 31](#_Toc74828568)

[3.2 Розробка (опис) інтерфейсу підсистеми 32](#_Toc74828569)

[3.3 Формат зберігання даних в базі прецедентів 33](#_Toc74828570)

[3.4 Опис формату зберігання додаткових даних 36](#_Toc74828571)

[3.5 Опис використовуваної нейронної мережі 37](#_Toc74828572)

[РОЗДІЛ 4 41](#_Toc74828573)

[ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСОБУ 41](#_Toc74828574)

[4.1 Технічне тестування програмного продукту 41](#_Toc74828575)

[4.2 Тестування розробленої системи ШІ 44](#_Toc74828576)

[ВИСНОВКИ 48](#_Toc74828577)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 49](#_Toc74828578)

[ДОДАТОК А 51](#_Toc74828579)

[ВИХІДНИЙ КОД 51](#_Toc74828580)

[Підсистема прецедентів 51](#_Toc74828581)

[Підсистема нейронної мережі 56](#_Toc74828582)

[Підсистема прийняття рішень 61](#_Toc74828583)

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

**ADT** – Android Developer Tools

**GUI** – Графічий інтерфейс користувача (з англ. Graphical User Interface)

**NPC** – Персонаж під управлінням ШІ (з англ. Non-Player Character)

**SDK** – Software Development Kit

**БЗ** – База знань

**Бот** – Ворожий до гравця персонаж під управлінням ШІ (з англ. Bot)

**БП** – База прецедентів

**ГВЧ** – Генератор випадкових чисел (з англ. RNG – Random Number Generator)

**ІШІ** – Ігровой штучний інтелект

**ІЗ** – Інформаційне забезпечення

**ККГ** Колекційна карткова гра (от англ. CCG – Collectible Card Game)

**ОПР** – Особа, приймаюча рішення

**НМ** – Нейронна мережа

**ОП** – Оперативний пристій

**ОС** – Операційна система

**ПК** – Персональний комп'ютер

**ПП** – Програмний продукт

**ПЗ** – Програмний засіб

**ПЕС** – Прецедентна експертна система

**ЕОМ** – Електронна обчислювальна машина

**ЕС** – Експертна система

**МВР** – Мова програмування високого рівня

# ВСТУП

Ігровий штучний інтелект (ІШІ) (англ. *Game artificial intelligence*) - сукупність методів програмування, які використовуються в комп'ютерних іграх для створення ілюзій інтелекту в поведінці персональних комп'ютерів, управління комп'ютерами. Ігровий ШI, крім методу традиційного досвідченого інтелекту, включає такі алгоритми, як теорії управління, робототехніка, комп'ютерна графіка та інформатика загалом.

Реалізація ШI сильно впливає на ігровий процес, системні вимоги та бюджет гри, а розробники балансують між цими вимогами, намагаючись створити цікавий та невимогливий до ресурсів ШІ витрачаючи мінімум часу. Тому підхід до гри ШI серйозно відрізняється від підходу до традиційного ШI - широко застосовуються різного роду спрощення, обмани і емуляції.

Поява нових ігрових жанрів у 1990-х простимулювало використання таких формальних інструментальних засобів штучного інтелекту, як остаточні автомати. Стратегії реального часу ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *RTS*) ставили перед ігровим штучним інтелектом багато нових завдань: неповна інформація, знаходження шляху, приймання рішень у реальному часі та економічне планування [1].

Колекційна карткова гра (ККГ) ([англ](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA),  *Collectible Card Game*) - різновид настільних ігор. ККИ не мають нічого спільного з традиційними азартними картковими іграми. Єдиний загальний елемент - це карта [2].

У колекційних карткових іграх, так само, як і в іграх стратегічного жанру, кожна ігрова сторона знаходиться в умовах неповної інформованості, але завдання побудови ШІ в даному випадку спрощується, так як потрібно прийняття рішень не в реальному часі, а в покроковому режимі.

Смартфони на базі Android дуже популярні: 75% смартфонів світу, працюють на ​​операційній системі Android [3]. Відповідно, і ринок додатків для даної операційної системи дуже великий, але в той же час ніша колекційних карткових ігор тут не насичена - це дає прекрасну можливість виділитися серед конкурентів, наприклад, за рахунок цікавого ігрового процесу проти ШІ.

# РОЗДІЛ 1

# СИСТЕМНИЙ АНАЛІЗ

## Аналіз завдання

Основна мета даної дипломної роботи - рішення задачі прийняття рішень в умовах неповної інформації і жорстких часових обмежень. Розроблена система повинна ефективно приймати рішення в неординарних ситуаціях і мати високу ступінь адаптованості. Для вирішення даного завдання, наочного тестування і дослідження такої системи було прийнято рішення розробити ігровий штучний інтелект для покрокового ігрового Android-додатку на базі нейронної мережі, здатної до самонавчання та імітації поведінки реального гравця.

Основними вимогами тут є:

* можливість збору статистики тактичних рішень користувача-гравця і її внесення в БЗ;
* отримана система ШІ повинна бути значно більш гнучкою, ніж її аналог, побудований на базі дерева умов;
* в результаті, пісня 10-12 ігор користувача проти ШІ, відсоток перемог ШІ повинен рухатися до 50% і вище;
* початкові параметри нейронної мережі задаються з урахуванням оцінок, отриманих у 3 експертів, раніше приймаючих участь в тестуванні паперового прототипу ККГ (для формування достатньо сильного базового ШІ);
* час прийняття рішення ігровим ШІ під час його ходу не повинно перевищувати 1 секунду на будь-якому мобільному пристрої під управлінням ОС Android;
* можливість користувача проводити «відкат» бази знань в початковий стан для навчання ШІ з нуля;
* можливість зберігати окрему БЗ для кожного з ігрових персонажів;
* складність гри проти кожного із ботів має бути рівною і збалансованою (необхідно урівняти відсоток перемог і поразок для кожного з існуючих персонажів);
* обсяг кожної БЗ обмежений максимум 10 випадками;
* ШІ на базі нейронної мережі повинен ефективно працювати в умовах неповної інформативності, і його ефективність повинна постійно збільшуватися з часом.

Система штучного інтелекту буде розроблятися для покрокової цифрової ККГ «Board Duels». Було виготовлено модельний прототип ККГ и було проведено його тестування трьома експертами.

В ході проведення тестування прототипу експертами були сформовані основні правила і тактики гри для кожного з існуючих персонажів. Також було виконано балансування сил кожного з цих персонажів, а також кожної з карток, з цілю зменшити часові затрати на балансування на етапі створення програмного продукту.

## Аналіз проблемної області

Штучний інтелект на базі дерева умов має ряд недоліків:

* обмеженість (доволі важко реалізувати таке дерево, яке враховувало б абсолютно всі можливі ситуації в грі і визначало б найбільш доцільну тактику);
* громіздкий код (дерево умов, яке враховує хоча б базові варіанти ігрових ситуацій, являється доволі громіздким і містить велику кількість коду, що повторюється);
* мала гнучкість (дискретність параметрів і окремих рішень, часто велика похибка в прийнятих рішеннях при не великих змінах деяких параметрів);
* слаба адаптивність (такий ШІ дуже важко навчати, і це мало ефективно).

Але, з іншої сторони, при розробці ШІ на базі нейронної мережі виникають наступні проблеми:

* складність реалізації;
* складність базового налаштування мережі (необхідно встановити такі початкові параметри нейронної мережі, щоб проти ШІ на її основі було доволі цікаво грати, а подальше навчання проходило як найбільш ефективно);
* гарна збіжність (на протязі всього часу навчання параметри нейронної мережі повинні наближатися до певних оптимальних значень, і уже через 10-12 ітерацій навчання відсоток перемог проти користувача повинен бути 50% і більше).

## Постановка задачі

Необхідно розробити адаптивну систему прийняття рішень, здатну в умовах не повної інформації і жорстких обмежень в часі приймати максимально ефективні рішення в неординарних ситуаціях. Для вирішення даної задачі було прийнято рішення модернізувати і доповнити систему ігрового ШІ в цифрові ККГ «Board Duels», додавши туди систему самонавчання і ведення бази знань для 2 існуючих ігрових персонажів, з урахуванням вище перелічених вимог. Це дозволить дослідити і протестувати розроблену СПР на практиці і зберегти всю необхідну статистику.

## 1.4 Можливі області застосування системи

Система ШІ що розробляється в даній дипломній роботі має багато спільного з експертними прецедентними системами (ЕПС).

Перші експертні системи (тобто системи першого покоління) представляли собою фактично ускладнені електронні довідники, де прийняття рішень здійснювалося за допомогою перебору варіантів. Вони чудово виконували пошукові задачі (наприклад, пошук необхідних телефонних номерів або формул), але мали мало спільного з інтелектуальною діяльністю. Строго кажучи, «База Знань» пробних систем була фактично базою даних; ці системи відрізнялися від традиційних програм головним чином можливістю для спеціаліста-користувача складати і задіювати для подальших пошуків різного роду складні вибірки.

Доволі швидко розробники ЕС першого покоління зрозуміли, що шлях простого перебору варіантів – довгий і непродуктивний. Людина ніколи не перебирає всі варіанти, вона відкидає явно нереальні. Так виникли ЕС другого покоління, які мали вбудовані алгоритми швидкого знаходження потрібного варіанту, тобто були такими що мали нехай і примітивну, але все ж таки «Базу Знань». Говорячи сучасною мовою, їх БЗ включала Базу даних і алгоритми по роботі з цими даними, закладені експертами-розробниками [4].

Експертні системи третього покоління характерні наявністю дружнього інтерфейсу користувача, а експертні системи четвертого покоління почали розвиватися завдяки значному росту продуктивності комп’ютерів. ЕС четвертого покоління – повністю інакші ЕС. Вони не тільки виконували складні розрахунки і пошуки, а також витягували дані із пам’яті, видавали рекомендації та приймали рішення.

Справжній прорив (а фактично поява тих самих експертних систем, які і працюють з кадрами сьогодні), відбувся в другій половині 90-х років, коли різко збільшився об’єм пам’яті, що зберігалася на дискові комп’ютера, і знизилась її ціна. Це дало можливість спочатку в США (фірма «American Research»), а через деякий час і в Росії, (НПО «ЭТАЛОН», а для громадянських потреб – Центр «Хобби») розробити експертні системи 5-го покоління – прецедентні експертні системи. Такі системи в корені відрізняються від своїх попередників тим, що їхню Базу знань формують не тільки логічні правила, а і «база прецедентів» – результати дослідження реальних людей і інформація про подальший успіх / невдачу їх діяльності.

Іншими словами, прецедентні експертні системи окрім «Бази знань» мають також і «Базу прецедентів» (БП) [4].

Можливі області застосування розроблювальної системи ШІ:

* в ЕС для прийняття транспортних рішень (наприклад, морська справа, де необхідно за обмежений час, з відомим розташуванням морських суден і відомою акваторією, прийняти рішення про необхідність маневру корабля в поточний момент часу);
* в охоронних системах зразку «розумний дім» (наприклад, реалізувати ЕС, яка буде приймати рішення про захисні проти зловмисника, особливістю систем такого роду є те, що зловмисник частіше всього намагається діяти оригінальним способом, не схожим на решту, щоб оманути систему).

В вище перелічених випадках дуже важливо, щоб швидкість прийняття рішення і його ефективність була не нижче, чим у реальних експертів (ОПР), якими дана система навчалась.

Під час виконання даної дипломної роботи я буду розробляти покроковий ігровий додаток для ОС Android. Причини такого вибору вказані нижче:

* практично будь-яку систему ШІ найлегше вивчати і балансувати саме в ігровій формі;
* вважається, що обидві сторони знаходяться в умовах неповної інформативності, а це дозволяє сформувати більш стійку систему ШІ;
* задані часові обмеження на прийняття рішень, і їх ефективності (точність виконання цих умов простіше відстежити також в ігровій формі, користуючись засобами візуалізації);
* на мобільних пристроях з ОС Android існують достатньо жорсткі обмеження обчислювальної потужності, обсяг ОЗП и наданого обсягу місця на ПЗП (це, в свою чергу, дозволяє збільшити акцент на оптимізацію, і сформований алгоритм буде максимально швидкодіючим).

## 1.5 Обґрунтування і вибір програмної платформи реалізації програмного засобу

Для реалізації ПС була вибрана програмна платформа під управлінням ОС Android. Обґрунтування вибору:

* 75% смартфонів в світі працюють на ОС Android;
* частина колекційних карткових ігор на ринку додатків Android мала, хоча популярність цього жанру залишається високою уже декілька десятиліть;
* для платформи Android додатки розробляються на мові програмування Java (що дозволяє легко використувати багато додаткових бібліотек, можливості ООП);
* хороша підтримка розробників зі сторони компанії Google (наявність великої кількості документації і зразків Android-розробки);
* простота публікації програмного продукту (Android-додаток легко публікується в Google Play);
* розробка додатку для мобільної платформи дозволе зробити обов'язковий акцент на швидкодії, при мінімальних затратах ресурсів (тоді розроблений алгоритм буде максимально стійким і роботоздатним практично при будь-якій обчислювальній потужності).

# РОЗДІЛ 2

# СТРУКТУРНЕ ПРОЕКТУВАННЯ ПРОГРАМНОГО СЕРЕДОВИЩА

## 2.1 Опис структури даних

Перед тим, як почати розробку алгоритму, необхідно розглянути особливості предметної області, для якої буде розроблятися система ШІ. Продукт, що розробляється, в якому буде реалізована система ігрового штучного інтелекту з можливістю до самонавчання – це ігровий додаток «Board Duels».

«Board Duels» – це цифрова колекційна карткова гра (ККГ), в якій основна задача користувача-гравця зводиться до того, щоб довести рівень хіп-поінтів (з англ. Hit Points) персонажа суперника до нуля, і при цьому звертати увагу на те, щоб рівень хіт-поінтів власного персонажа не дійшов до нульового порогу. Як тільки рівень хіп-поінтів однієї із сторін досягає нуля – це зараховується, як поразка того гравця, під чиїм управлінням знаходився цей персонаж.

Цієї цілі гравець досягає шляхом використання карт під час свого ходу, атакуючих або захисних.

Крім шкали хіт-поінтів у кожної із сторін є шкала енергії, яка відображає кількість наявної у ігрового персонажа в поточному ході. Бали енергії користувач може витрачувати на спелл-карти (з англ. Spell Card) – їх відмінність від звичайних ігрових карт в тому, що вони являються вміннями ігрового персонажу, кожна карта має свою вартість, яка вимірюється в балах енергії, і при використанні гравцем накладає який небудь ефект на дружнього персонажа або на персонажа опонента.

В ККГ «Board Duels» існує два типи карт можливостей: старші і молодші. Вони всі використовуються для взаємодії з персонажем опонента. Принципова різниця між ними описана в таблиці 2.1.

Таблиця 2.1 – Класифікація карт умінь в ККГ «Board Duels»

|  |  |  |
| --- | --- | --- |
|  | Молодша карта (Minor card) | Старша карта (Major card) |
| Плата за застосування карти | Від 0 до 10 балів енергії | Завжди безкоштовне використання |
| Доступна кількість застосувань | Може застосуватися багато разів, після використання повертається в колоду персонажа в випадкову позицію | Застосовується лише один раз, після чого видаляється із колоди персонажа |
| Можливий ефект після використання | Відбтрає хіт-поінти суперника, або відновлює хіт-поінти дружнього персонажа. Зазвичай, такі карти мають незначний ефект. | Накладає бонусний ефект на дружнього персонажа, такий, що значно збільшує вплив його наступних карт. Також може накладати суттєвий негативний ефект на персонажа суперника. |

Нижче наведена концептуальна схема ККГ «Board Duels» (див. рис. 2.1).

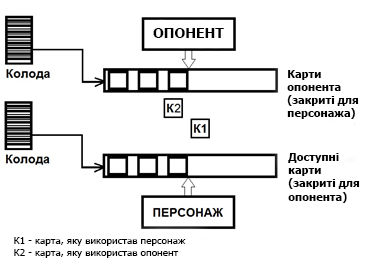


Рисунок 2.1 – Концептуальна схема КГГ «Board Duels»

Розглянемо структуру даних, які будуть використані в додаткові «Board Duels». Всього в цифровій ККГ «Board Duels» можна виділити 2 основних типів сутностей:

* ігровий персонаж (Hero);
* карта можливостей (SpellCard).

Кожний ігровий персонаж має в своєму розпорядженні певний набір карт можливостей, який називається колодою персонажа. Вважається, що в кінцевій версії ККГ «Board Duels» в колоді буде знаходитись 12 карт, а в розпорядженні персонажа під час гри може бути не більше 5 карт.

Таблиця 2.2 – Структура даних про ігрового персонажа

|  |  |  |  |
| --- | --- | --- | --- |
| Назва поля | Тип даних | Можливі значення | Опис |
| 1 | 2 | 3 | 4 |
| hp | Integer | Від 0 до max\_hp | Поточний рівень хіт-поінтів (на початку кожної гри поточний рівень рівний максимальному) |
| ep | Integer | Від 0 до max\_ep | Поточний рівень енергії (на початку кожної гри поточний рівень рівний максимальному) |
| max\_hp | Integer | Від 20 до 50 | Максимальний рівень хіт-поінтів |
| max\_ep | Integer | Від 3 до 10 | Максимальний рівень балів енергії |
| regen | Integer | Від 0 до 3 | Рівень регенерації (показує, скільки балів енергії буде відновлювати кожний хід) |
| shield | Integer | Від 0 до 30 | Поточний об’єм поглинання (показує, яка кількість одиниць пошкодження буде поглинена) |

Продовження таблиці 2.2

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| defence | Float | Від 0 до 1 | Рівень захисту персонажа (показує, скільки % отриманих пошкоджень буде проігноровано) |
| defence\_time | Integer | ≥ 0 | Решта (кількість) ходів, протягом яких буде активним захист |
| stun\_time | Integer | ≥ 0 | Решта (кількість) ходів, протягом яких персонаж не зможе використовувати карти |
| damage\_bonus | Integer | Від -10 до 10 | Діючі бонуси до атаки (показує, на скільки одиниць більшими будуть пошкодження, які наносяться використаними картами) |
| damage\_bonus\_time | Integer | ≥ 0 | Кількість ходів що залишилися, протягом яких буде активний бонус до атаки |
| immortality\_time | Integer | ≥ 0 | Кількість ходів зо залишилися, протягом яких персонаж буде невразливий (рівень хіт-поінтів персонажа не зможе бути нижче 1 одиниці) |
| bonus\_regen | Integer | Від -3 до 3 | Діючий бонус регенерації (показує, на скільки одиниць буде збільшено рівень регенерації персонажа) |

Продовження таблиці 2.2

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| costs\_modifier | Integer | Від -5 до 5 | Модифікатор вартості карт (показує, наскільки зміняться затрати на використання карт) |
| costs\_modifier\_time | Integer | ≥ 0 | Решта (кількість) ходів, протягом який модифікатор вартості буде активним |
| HoT\_power | Integer | Від 0 до 10 | Поточний рівень регенерації за хід (показує, скільки хіт-поінтів буде відновлюватися за кожний хід поки ефект буде активним) |
| HoT\_time | Integer | ≥ 0 | Решта (кількість) ходів, протягом яких буде активним ефект періодичної регенерації |
| max\_hp\_bonus | Integer | Від -5 до 10 | Поточна кількість діючих бонусних хіт-поінтів |
| max\_hp\_bonus\_time | Integer | ≥ 0 | Решта (кількість) ходів, протягом яких буде активним бонус хіт-поінтів |
| isPlayer | Boolean | Істина/Брехня | Прапор, який показує чи знаходиться персонаж під управлінням гравця (або ШІ) |
| cards | ArrayList | 12 елементів | Масив даних карт персонажа |
| net | NeuralNetwork | Нейронна мережа в одному екземплярі | Нейронна мережа, яка відноситься до системи ШІ даного персонажа |

Продовження таблиці 2.2

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| dots | ArrayList | Від 0 до 5 елементів | Масив даних негативних ефектів, що діють на персонажа (які періодично наносять йому пошкодження) |

Таблиця 2.3 – Структура даних карти

|  |  |  |  |
| --- | --- | --- | --- |
| Назва поля | Тип  даних | Можливі значення | Опис |
| 1 | 2 | 3 | 4 |
| name | String | - | Назва карти |
| card\_id | Integer | Будь-яке унікальне значення | Ідентифікатор карти |
| isMajor | Boolean | Істина/Брехня | Прапор, який показує тип карти (старша або молодша) |
| isActive | Boolean | Істина/Брехня | Прапор, який показує стан карти (активна або ні) |
| inHands | Boolean | Можливе значення | Прапор, який показує, чи знаходиться дана карта в руках персонажа |
| cost | Integer | Від 0 до 10 | Вартість використання карти (балів енергії) |
| increase\_by\_cost | Boolean | Істина/Брехня | Прапор, який показує чи буде збільшуватися ефект від використання карти, в залежності від затраченої енергії |
| base\_damage | Integer | Від 0 до 10 | Базове пошкодження, яке може бути нанесене картою |

Продовження таблиці 2.3

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| max\_damage | Integer | Від base\_damage до 10 | Максимальне пошкодження, яке може бути нанесене картою |
| activate\_effect\_while\_  getDamage | Boolean | Істина/Брехня | Прапор, який показує, чи буде автоматично активовано ефект карти при отриманні ушкодження |
| effect\_base\_damage | Integer | Від 0 до 10 | Базове ушкодження від додаткового ефекту карти |
| effect\_length | Integer | ≥ 0 | Кількість ходів, протягом яких ефект додаткової карти буде активним |
| effect\_increasing | Integer | Від -3 до 3 | Величина на яку буде збільшена сила ефекту з кожним ходом |
| effect\_base\_healing | Integer | Від 0 до 10 | Базова сила регенерації додаткового ефекту |
| attackHimself | Boolean | Істина/Брехня | Прапор, який чи буде направлена дія карти на дружнього персонажа |
| base\_absorb | Integer | Від 0 до 10 | Базова кількість одиниць пошкодження, які будуть поглинені при використанні карти |
| max\_absorb | Integer | Від base\_absorb до 10 | Максимальна кількість одиниць пошкодження, які можуть бути поглинені при використанні карти |
| base\_healing | Integer | Від 0 до 20 | Базова кількість хіт-поінтів, картою що регенерує |

Продовження таблиці 2.3

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| isNotFatal | Boolean | Істина/Брехня | Прапор, що показує чи буде карта фатальною (якщо ні, тоді кількість хіт-поінтів цілі не може бути нижче 1) |
| removingEffects | Boolean | Істина/Брехня | Прапор, який показує, чи видаляє дана карта решту ефектів |
| isTerminated | Boolean | Істина/Брехня | Прапор, який показує, чи була видалена ця карта з колоди |
| immortality\_length | Integer | ≥ 0 | Кількість ходів, протягом яких карта дає персонажу невразливість до атак |
| max\_hp\_bonus | Integer | Від 0 до 10 | Кількість бонусних хіт-поінтів, які додаються до максимального рівня |
| my\_damage\_bonus | Integer | Від 0 до 5 | Бонус нанесеного картами ушкодження |
| my\_cost\_bonus | Integer | Від -5 до 0 | Бонус вартості карт (наскільки балів енергії меншими будуть затрати на використання карт) |
| my\_healing\_bonus | Integer | Від 0 до 5 | Бонус сили регенерації (наскільки більше хіт-поінтів буде відновлюватися картами) |
| my\_regen\_bonus | Integer | Від 0 до 3 | Бонус регенерації (наскільки балів енергії більше буде відновлювати кожен хід) |
| defence\_value | Integer | Від 0 до 100 | Рівень захисту, який активований цією картою (наприклад 5 – це 5% пошкоджень що ігноруються) |

Продовження таблиці 2.3

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| increase\_by\_hp\_lost | Boolean | Істина/Брехня | Прапор, який показує, чи буде збільшена ефективність даної карти в залежності від втрачених хіт-поінтів |
| ignore\_defence\_  and\_absorbs | Boolean | Істина/Брехня | Прапор, який показує, чи буде дана карта ігнорувати рівень захисту і поглинання цілі |
| stun\_time | Integer | Від 0 до 2 | Кількість ходів, протягом яких ціль буде дезорієнтована (тобто не зможе використовувати карти) |

## 2.2 Розробка алгоритму









## 2.3 Аналіз схеми алгоритму і розробка структури ПЗ

На блок схемі можна побачити, що система ШІ фактично розділена на 4 функціональні частини:

* підсистема аналізу ігрової ситуації;
* підсистема роботи з базою знань (пошук аналогічних випадків в БЗ, запис нових випадків, апроксимація схожих випадків для подальшого навчання нейронної мережі);
* підсистема розрахунку пріоритетів (нейронна мережа);
* підсистема прийняття рішень (вибір карти в залежності від розрахованих пріоритетів).

Найбільш проблематичними являються: підсистема роботи з базою знань і підсистема розрахунку пріоритетів. Підсистема аналізу ігрової ситуації представляє собою ніщо інше, як фрагмент додатку, який конвертує інформацію про ігрових персонажів і картах в зручний для роботи з БЗ і навчання нейронної мережі формат. Також в даній підсистемі проводиться перевірка цих величин для того, щоб, наприклад, виявити переможця и переможеного, і оновити стан всіх активних ефектів.

Додаток буде розроблятися в середовищі Eclipse з допомогою плагіну Android Developer Tools (ADT). Для розробки використаємо мову програмування – Java.

# РОЗДІЛ 3

# РОЗРОБКА ПРОКТНИХ РІШЕНЬ ПО ПРОГРАМНОМУ І ІНФОРМАЦІЙНОМУ ЗАБЕЗПЕЧЕННЮ СИСТЕМИ

## 3.1 Основні інформаційні частини ПЗ

Програмне забезпечення, що розробляється складається із декількох основних компонентів. Структурна схема ПЗ наведена нижче на рисунку 3.1.



Рисунок 3.1 – Структурна схема додатку

Створений для дослідження розробленої системи ШІ програмний продукт складається із декількох основних компонентів:

* база прецедентів (в якій зберігається від 10 до 20 ігрових випадків; для кожного з них там описано усереднені рішення, зіставлені з оцінок експертів);
* підсистема прецедентів (програмний модуль, який замається завантаженням прецедентів з файлу, їх обробкою і пошуком найбільш подібних випадків при заданих параметрах поточної ситуації; підсистема прецедентів в результаті своєї роботи видає 2 найбільш подібних прецедента, які після цього використовуються системою для навчання нейронної мережі для її адаптації до даної ситуації);
* підсистема навчання нейронної мережі (отримує дані від 2 найбільш подібних прецедентів, апроксимує їх, і використовує параметри усередненого нового випадку для навчання одношарової нейронної мережі; в результаті навчання ми отримуємо нейронну мережу, максимально адаптовану до даної ситуації, і ми отримуємо вихідні сигнали, які обробляють численні пріоритети, необхідні для прийняття рішення);
* модуль прийняття рішень (по відомим пріоритетам, отриманими із підсистеми навчання нейронної мережі, даний модуль приймає рішення – будує ланцюг із можливих дій, в залежності від пріоритетів їх категорій, перевіряє можливість виконання кожного з них, і вибирає ту дію, яку можливо виконати зараз, і з максимальним пріоритетом тієї категорії, до якої воно відноситься);
* модуль ігрової механіки (активна частина системи, яку можна назвати «споживачем рішень», – даний модуль виконує ті дії, які були вказані системою прийняття рішень; також модуль ігрової механіки регулярно інформує підсистем навчання нейрона мережі про поточні ситуації, описуючи в числовому вигляді всі її необхідні параметри);
* додаткові ігрові дані (необов’язкова частина розробленої системи, представляє собою набір XML-файлів, в яких зберігаються дані конфігурації для ігрових персонажів і карт, які вони можуть використовувати під час ігрового процесу).

На рисунку 3.2 представлена діаграма класів створюваної ККГ.

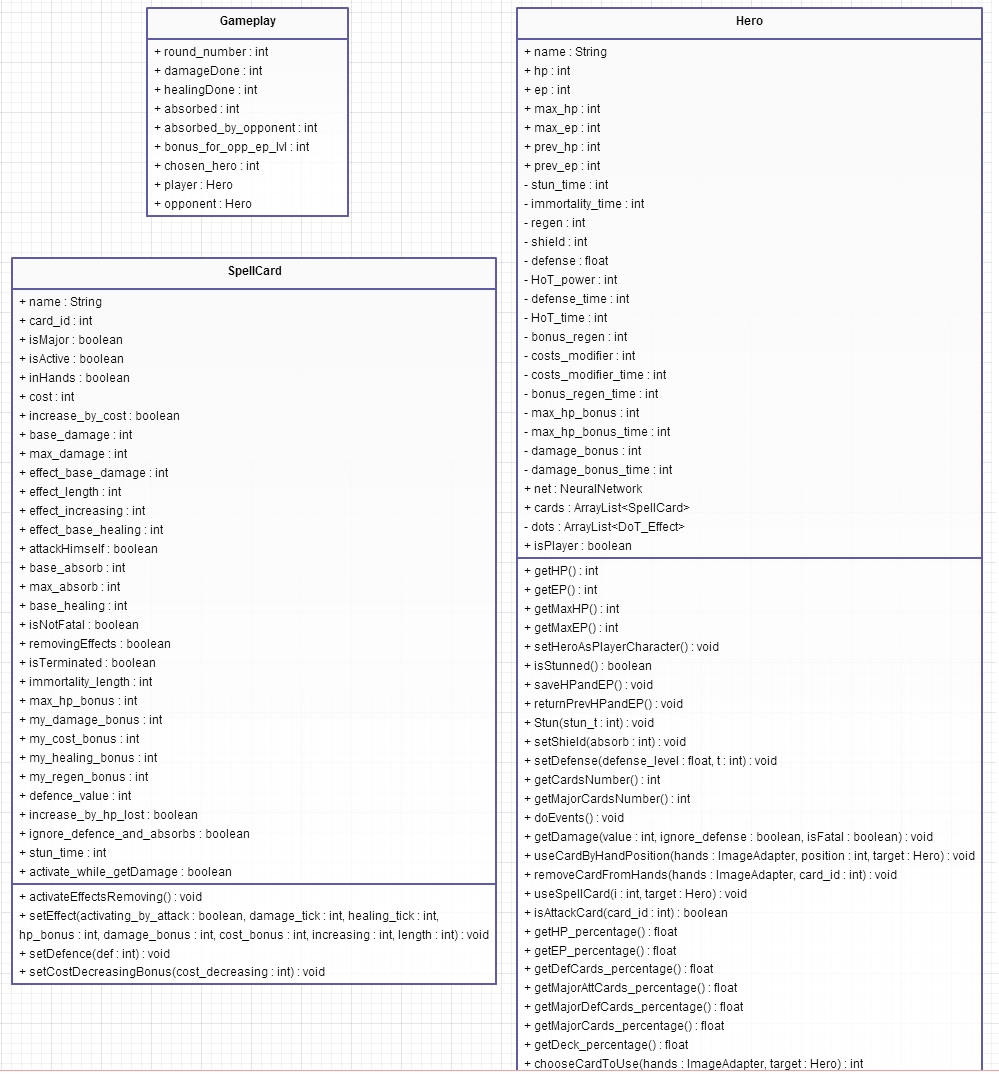


Рисунок 3.2 – Діаграма класів ККГ «Board Duels»

## 3.1.1 Вихідний код програмного засобу. Вихідний код системи прецедентів, підсистеми нейронної мережі, модуля прийняття рішень – в додаткові А

Вихідний код підсистеми прецедентів, підсистеми нейронної мережі, модуля прийняття рішень – в додаткові А.

## 3.2 Розробка (опис) інтерфейсу підсистеми

Для додатку був розроблений графічний інтерфейс користувача. В процесі його розробки було зроблено акцент на те, щоб інтерфейс був максимально інформативним і інтуїтивно зрозумілим в управлінні. Основним ускладненням було жорстке обмеження по розміру екрану: необхідно було розмістити всі необхідні компоненти по площі екрану мобільного пристрою, так щоб вони не перекривали один одного, і не псувалося сприйняття інтерфейсу користувачем.

Також елементи інтерфейсу були організовані таким чином, щоб витрачалося як можна менше ресурсів і було не важко його модифікувати в майбутніх версіях.

Знімки екрану додатку наведені на рисунках 3.3 і 3.4.



Рисунок 3.3 – Графічний інтерфейс додатку



Рисунок 3.4 – Графічний інтерфейс користувача

На рисунку 3.4 показано момент використання гравцем карти, коли додаток починає відображати її анімацію і виводити текстову інформацію про результат використання карти.

## 3.3 Формат зберігання даних в базі прецедентів

База прецедентів представляє собою документ в форматі XML, зразок документу наведено нижче, на рисунку 3.5.



Рисунок 3.5 – База прецедентів для ККГ «Board Duels»

На даному рисунку показана база прецедентів, яка містить в собі лише один випадок. В якості початкової БП в ККГ «Board Duels» використовується БП з десятьма випадками. В процесі навчання системи ШІ база прецедентів змінюється, і туди також можуть додаватися нові випадки, для більш точного відбору параметрів для нейронної мережі і прийняття наступних рішень.

Розглянемо структуру документа більш детально, в якому зберігається база прецедентів для ККГ. Документ має деревоподібну структуру, де корінним елементом являється елемент «cases». В середину нього вкладені прецеденти, вони можуть бути записані в будь-якому порядку. Структура прецеденту описана в таблиці 3.1.

Таблиця 3.1 – Структура прецеденту.

|  |  |  |
| --- | --- | --- |
| Елемент | Тип вкладених даних | Опис |
| <input> | Float | Вхідний сигнал (описує один із параметрів поточної ситуації) |
| <output> | Float | Еталонний вихідний сигнал (для базової вибірки прецедентів – вихідні сигнали були отримані шляхом проведення експертного оцінювання) |
| <potency> | Float | Потенціал прецедентів (показує, в якій частині всіх ігор, де була отримана перемога, був використаний даний прецедент) |
| <frequency> | Float | Частота прецеденту (показує, як часто зустрічався даний прецедент) |

Дана структура являється універсальною і може бути використана для будь-якої системи і будь-якої предметної області. Використовуючи дані про вхідні сигнали, можна знаходити того прецеденту із бази, який точніше всього описує поточну ситуацію, і використати еталонне рішення для повторного навчання нейронної мережі. В розробленій системі ШІ повторне навчання відбувається лише в тому випадку, якщо отримане рішення не можливо реалізувати, а також в самому початку, коли завантажується базовий прецедент для ініціалізації. В інших ситуаціях, якщо в базі знайдено потрібний прецедент, і його рішення являється достатньо ефективним (його ефективність можна оцінити шляхом оцінки потенціалу і частоти прецеденту) – то система ШІ просто копіює готове рішення із знайденого прецеденту. В іншому випадку, коли знайдений прецедент пов’язаний з неефективним рішенням, або з’являється не часто – тоді проводиться повторне навчання нейронної мережі, і пошук рішення для поточної ситуації.

Розглянемо більш детально конкретну систему прецедентів, яка була використана в ККГ «Board Duels». Всі сигнали, величини яких зберігаються в прецедентах, являються нормалізованими і приймають значення від 0 до 1.

## 3.4 Опис формату зберігання додаткових даних

Окрім інформації про прецеденти, яка необхідна для повторного навчання нейронної мережі і для пошуку рішень, для програмного продукту необхідне зберігання додаткової інформації, пов’язаної зі специфікою розроблюваного Android-додатку.

Так як для тестування системи ШІ була розроблена покрокова Android-гра, і ключовим елементом цієї гри являється карта – тому потрібно детально описати структуру даних карти.

Опис кожної з карт ККГ «Board Duels» зберігається в окремому документі XML. Зразок документу наведено на рисунку 3.6.

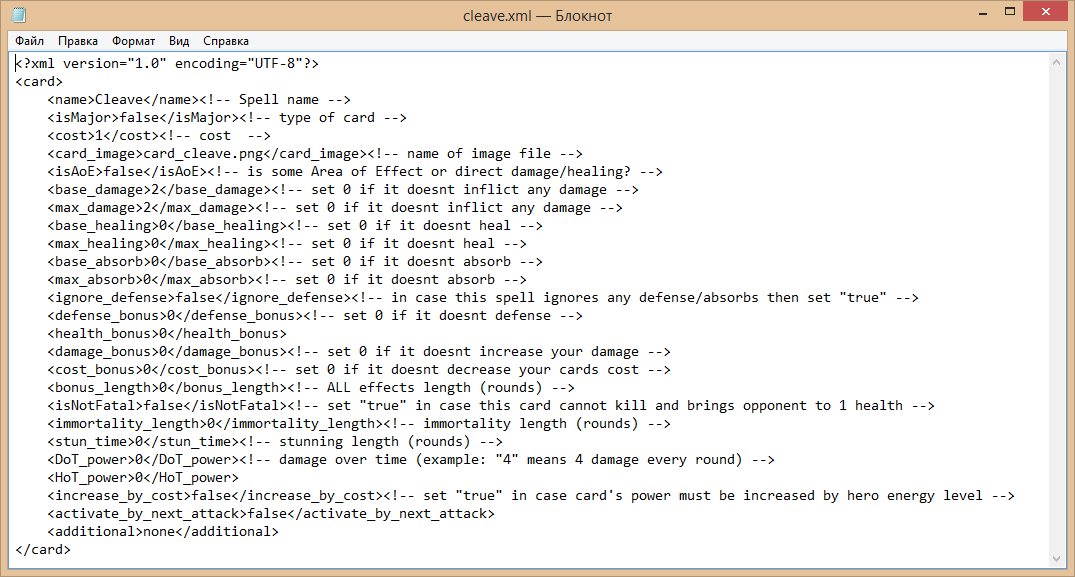


Рисунок 3.6 – Зразок XML-документу, який описує карту в ККГ «Board Duels»

Також в ККГ «Board Duels» зберігаються дані про ігрових персонажів. Зразок документу, який описує параметри персонажа, показано на рисунку 3.7.

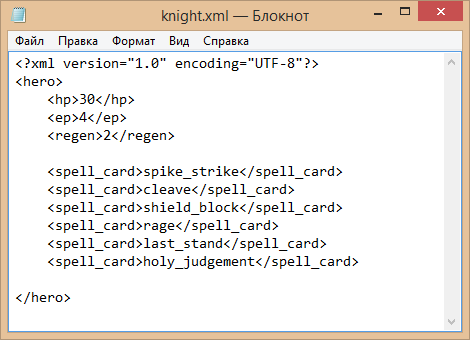


Рисунок 3.7 – XML-документ, який описує параметри персонажа «Лицар»

## 3.5 Опис використовуваної нейронної мережі

В розроблюваному в цілях тестування ШІ ігровому додатку «Board Duels» використовується одношарова нейронна мережа, з 14 вхідними і 4 вихідними сигналами.

Нейронна мережа містить в собі 4 нейрона. Кожний штучний нейрон представляє собою суматор вхідних сигналів і порогову передаточну функцію наступного виду:

(3.1)



, де ε – поріг чутливості нейрону (у використовуваній ККГ «Board Duels» нейронній мережі дана величина рівна 0,05 і являється однаковою для всіх нейронів).

Для кожного j-го нейрона вхідний сигнал *Xj* являється сумою добутків кожного з 14 вхідних сигналів мережі відповідний синаптичну вагу aij (який в розробленому алгоритмі навчання напряму залежить від пріоритету і-го вхідного сигналу для j-го нейрону). Формула (3.2), по якій проводиться розрахунок вхідного сигналу нейрона, наведено нижче:

(3.2)



Сенс вхідних і вихідних сигналів описано в таблицях 3.2 – 3.3.

Таблиця 3.2 – Сенс кожного з вхідних сигналів нейронної мережі в ККГ «Board Duels»

|  |  |
| --- | --- |
| Номер вхідного сигналу | Опис |
| 1 | 2 |
| 0 | Рівень здоров’я ігрового персонажа |
| 1 | Рівень здоров’я опонента |
| 2 | Рівень енергії ігрового персонажа |
| 3 | Рівень енергії опонента |
| 4 | % карт в руках у опонента (в руках їх може бути не більше 5) |
| 5 | % наповненості колоди (максимум 12 карт) |
| 6 | % наповненості колоди опонента |
| 7 | % атакуючих карт в руках |
| 8 | % карт захисного характер в руках |
| 9 | % старших атакуючих карт в руках |
| 10 | % старших карт захисного характеру в руках |
| 11 | % карт в руках у ігрового персонажа |

Продовження таблиці 3.2

|  |  |
| --- | --- |
| 1 | 2 |
| 12 | % відсутньої кількості балів здоров’я в опонента |
| 13 | % відсутньої кількості балів здоров’я в ігрового персонажа |

Таблиця 3.2 – Сенс кожного з вихідних сигналів нейронної мережі в ККГ «Board Duels»

|  |  |
| --- | --- |
| Номер вихідного сигналу | Опис |
| 0 | Пріоритет атаки |
| 1 | Пріоритет захисту |
| 2 | Пріоритет використання старшої атакуючої карти |
| 3 | Пріоритет використання старшої карти захисту |

На рисункові 3.8 показано схему нейронної мережі яку було використано в роботі.

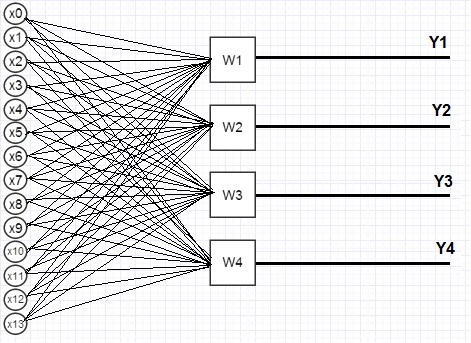


Рисунок 3.8 – Схема нейронної мережі, яку було використано в ККГ «Board Duels»

Був розроблений спеціальний алгоритм навчання нейронної мережі – по пріоритетам сигналів. Вважається, що нам відомі пріоритети кожного з вхідних сигналів – тим більше він буде впливати на загальну суму сигналів, вхідних в нейрон, і навпаки. Таким чином, можна достатньо просто вирахувати коефіцієнти синаптичних зв’язків, при заданих пріоритетах вхідних сигналів.

Алгоритм навчання детально описано в розділі 2.2.

В таблиці 3.4 можна побачити, які пріоритети у вхідних сигналів, для кожного з нейронів мережі.

Таблиця 3.4 – Задані пріоритети вхідних сигналів для нейронної мережі в ККГ «Board Duels»

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Нейрон | X0 | X1 | X2 | X3 | X4 | X5 | X6 | X7 | X8 | X9 | X10 | X11 | X12 | X13 |
| W1 | 5 | 12 | 11 | 4 | 2 | 7 | 3 | 10 | 1 | 8 | 6 | 9 | 13 | 0 |
| W2 | 2 | 8 | 4 | 12 | 11 | 3 | 10 | 1 | 9 | 5 | 6 | 7 | 0 | 13 |
| W3 | 0 | 13 | 4 | 1 | 3 | 8 | 2 | 11 | 7 | 12 | 5 | 10 | 9 | 6 |
| W4 | 1 | 6 | 4 | 11 | 12 | 9 | 10 | 5 | 8 | 2 | 7 | 3 | 0 | 13 |

Дані пріоритети було вибрано шляхом отримання оцінок від експертів: їм було надано список критеріїв, які буде враховувати майбутня система прийняття рішень, і експерти оцінили численно важливість кожного із них для визначеного типу дій (4 типи дій і 14 критеріїв, по яким можна оцінювати ситуацію в процесі її аналізу).

# РОЗДІЛ 4

# ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСОБУ

## 4.1 Технічне тестування програмного продукту

На момент завершення роботи над дипломним проектом, була випущена версія 1.4.2 програмного продукту «Board Duels», і опублікована на Google Play. На рисунку 4.1 наведено звіт завантажень програмного продукту, таком там можна побачити кількість помилок, які виникли в ході тестування.

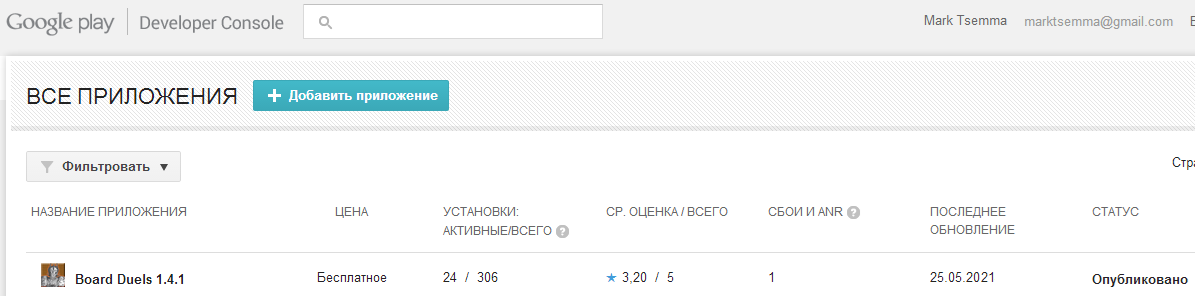


Рисунок 4.1 – Статистика завантажень програмного продукту

За останні 6 місяців існування програмного продукту відбувся 1 збій, на той момент (19 січня 2021 року) була опублікована версія 1.3 (рисунок 4.2).

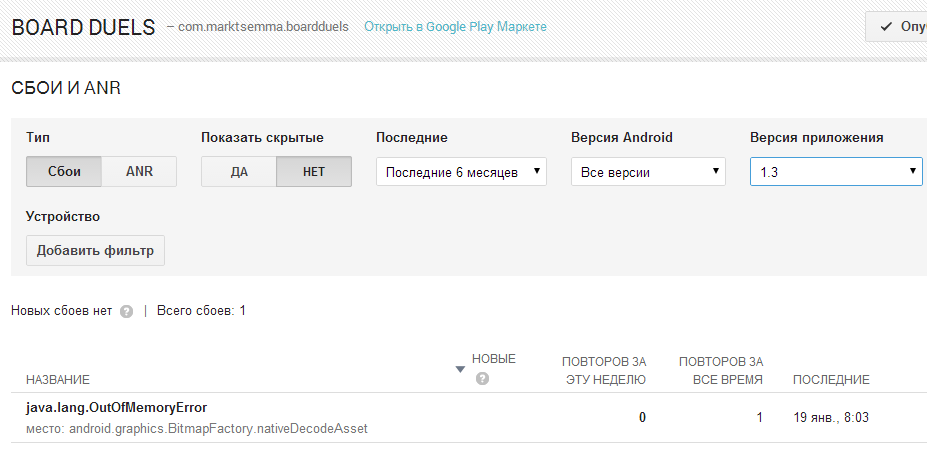


Рисунок 4.2 – Статистика збоїв програмного продукту

Більш детальний опис помилки наведено нижче, на рисункові 4.3. Проаналізувавши дану інформацію, виведену в консолі, можна зробити висновок, що причиною помилки було переповнення оперативної пам’яті пристрою.

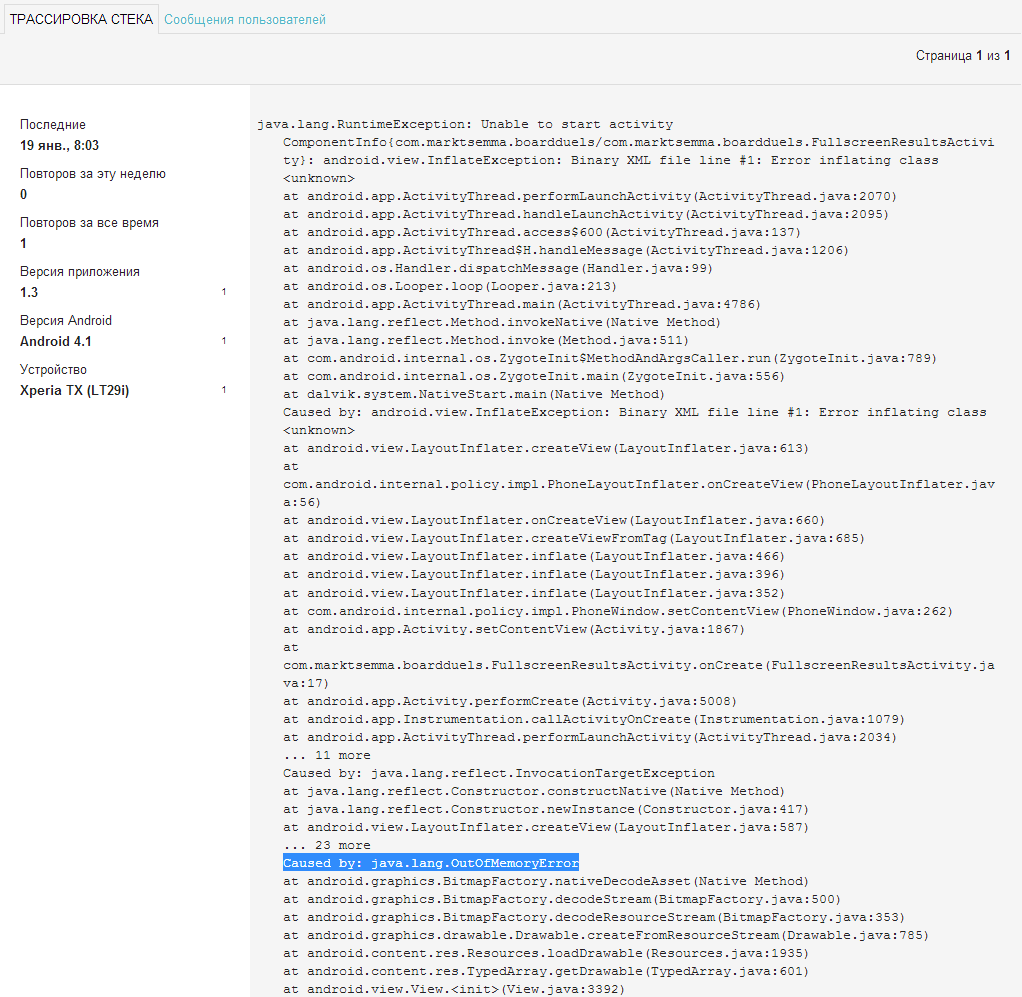


Рисунок 4.3 – Трасувальна сітка під час виникнення помилки

В тестуванні приймали користувачі з різних країн світу, і на різних пристроях. Більш детальна інформація вказана на рисунках 4.4 та 4.5.

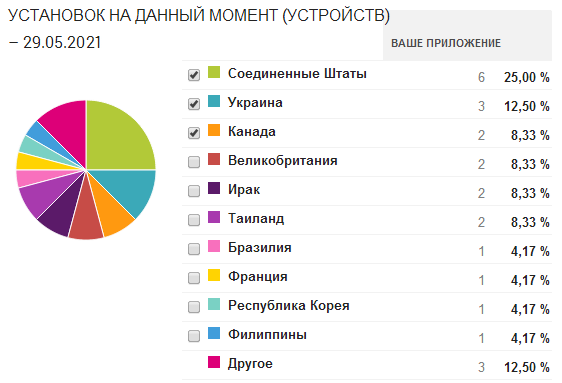


Рисунок 4.4 – Статистика інсталяції програмного продукту

Позиції лідерів по установці займають США та Україна.

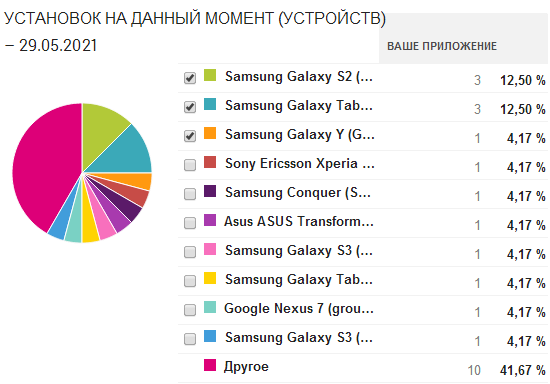


Рисунок 4.5 – Пристрої, на яких проводилося тестування

## 4.2 Тестування розробленої системи ШІ

В результаті тестування розробленого Android-додатку сформовано журнал стану системи ШІ на кожному з ходів. В даному журналі зберігається ключова інформація про стан системи ШІ:

* використані персонажем під управлінням ШІ карт (їх назви);
* знайдені системою прецедентів найбільш схожі випадки (точніше всього описуючі поточну ситуацію);
* значення вхідних сигналів нейронної мережі (описують ситуацію на поточному ході);
* отримані з нейронної мережі вихідні сигнали;
* величини синаптичних ваг кожної з нейронів на поточному ході.

Записи з даними про ключові ходи персонажа під управлінням ШІ для тестової гри показані в таблиці 4.1. Сенс кожного вхідного і вихідного сигналу нейронної мережі описаний в підрозділі 3.5 «Опис використовуваної нейронної мережі».

Таблиця 4.1 – Журнал тестової гри в цифровій ККГ «Board Duels»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № раунду | Вхідні сигнали (параметри поточної ситуації) | Найбільш схожі прецеденти з бази | Кількість доступних карт | Обчислені мережею пріоритети | Використовувана системою ШІ карта |
| 1 | 2 | 3 | 4 | 5 | 6 |

Продовження таблиці 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 0.83  1.0  1.0  0.5  0.5  0.5  0.67  0.0  0.25  0.0  0.5  0.75  0.0  0.17 | 4, 2 | 3 | 0.29  0.30  0.14  0.20 | Last Stand (major def.)  +  Rage  (major att.) |
| 2 | 0.78  1.0  1.0  0.13  0.5  0.5  0.67  0.25  0.25  0.0  0.25  0.75  0.0  0.22 | 4, 3 | 3 | 0.29  0.25  0.16  0.20 | Holy Judgment (major att.)  +  Cleave  (minor att.) |

Продовження таблиці 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 5 | 0.67  0.43  0.4  0.13  0.75  0.67  0.5  0.5  0.0  0.0  0.0  0.5  0.57  0.33 | 7, 6 | 3 | 0.27  0.32  0.18  0.17 | Spike Strike (minor att.)  +  Cleave  (minor att.) |
| 10 | 0.37  0.1  0.2  0.13  0.75  0.67  0.5  0.25  0.25  0.0  0.0  0.5  0.9  0.63 | 7, 8 | 3 | 0.22  0.33  0.17  0.16 | Shield Block  (minor def.) |

За результатами тестування можна оцінити ефективність ШІ в різних ситуаціях:

* ШІ віддає перевагу використанню старших карт, і використовує відразу, коли вони доступні (так як вони не несуть затрат балів енергії), і тоді залишається достатньо балів енергії для використання необхідних молодших карт;
* якщо серед наявних карт ШІ не знаходить найбільш пріоритетної в даній ситуації, тоді він вибирає другу за пріоритетом, і так далі. Саме тому в ситуаціях, коли пріоритет атаки був вищим ніж захисту, але ШІ використав карту захисту – це тому що не було підходящої атакувальної карти для використання в поточному раунді;
* база прецедентів, складається всього з 10 випадків проявляє себе достатньо ефективно, і 2 найбільш схожих прецедентів, які знаходить система ШІ, описують будь-яку ситуацію з високою точністю;
* завдяки тому що, модуль прийняття рішень не вибирає якесь конкретне одне рішення, а будує ланцюг можливих дій – дана система є достатньо гнучкою і добре адаптованою для будь-яких випадків, і завжди знаходиться яке-небу рішення;
* можна припустити, що об’єм бази прецедентів повинен бути більшим, в випадках, коли нейронна мережа приймає на вхід велику кількість сигналів (тобто для розробленої системи, щоб зберегти точність прийняття рішення – необхідно формувати кількість прецедентів, пропорційне кількості вхідних сигналів).

# ВИСНОВКИ

В ході написання даної курсової роботи була розроблена адаптивна система ШІ на базі одношарової нейронної мережі і бази прецедентів. Також був розроблений спеціальний алгоритм навчання нейронної мережі за відомими пріоритетами вхідних сигналів.

Дана система ШІ була протестована в покроковому ігровому додатку на платформі Android, з заданими часовими обмеженнями, а також обмеження на використання ресурсів. Додатково було проведено аналіз ринку ігрових Android-додатків, і був опублікований в дипломній роботі програмний продукт «Board Duels». В якості місця публікації був вибраний магазин Android-додатків Google Play.

В панелі розробника Google Play Developer Console було проведено статичне дослідження завантажень опублікованого додатку, і було проведено його відкрите тестування серед користувачів Google Play. Таким чином, були виявлені деякі помилки і збої, які виникали в ході роботи програми на різних мобільних пристроях під управлінням ОС Android.

Розроблена в дипломній роботі система ШІ може також бути використана в інших областях, з ситуаціями різного роду, в якості експериментальної системи. Наприклад, при вирішенні транспортних задач в морському ділі, де необхідно обрати необхідний маневр для морського судна, за обмежений час і з високою точністю. Також таку систему прийняття рішень можна ефективно використати в охоронних системах типу «розумний дім», де зловмисник часто може здійснювати неординарні дії, і система повинна бути добре адаптованою і гнучкою, щоб зуміти прийняти рішення в найбільш короткий термін.

В ігровій індустрії дана система (в якості системи ШІ) показала себе достойно, і при невеликих затратах ресурсів здатна достатньо ефективно протистояти реальному гравцю в покрокових іграх. Розроблене система ШІ спеціально призначена для ситуацій, де не можливо отримати повну і точну інформацію, і при цьому необхідно за короткий час знайти рішення, не менш ефективне якби його приймала людина.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Игровой искусственный интеллект [Электронный ресурс] / Wikimedia Foundation, Inc.. – Режим доступу: www/ URL: <http://ru.wikipedia.org/wiki/Игровой_искусственный_интеллект>. – с.1

2. Коллекционная карточная игра [Электронный ресурс] / Wikimedia Foundation, Inc.. – Режим доступу: www/ URL: http://ru.wikipedia.org/wiki/Коллекционная\_карточная\_игра. – с.1

3. Данные Strategy Analytics о рынке смартфонов за третий квартал 2013 года [Электронный ресурс] / OSZone.net. – Режим доступу: www/ URL: http://www.oszone.net/22504/Dannie\_Strategy\_Analytics. – с.1

4. Теория построения экспертных систем [Электронный ресурс] / Supermak.narod.ru, – Режим доступу: www/ URL: <http://supermak.narod.ru/page1_1.htm>. – с.1

5. ДСан ПиН 3.3.2-007-98. Государственные санитарные правила и нормы. Гигиенические требования к организации работы с визуальными дисплейными терминалами электронно-вычислительных машин. [Электронный ресурс] / Режим доступу : www/ URL: http://ot.kture.kharkov.ua/?page\_id=2766/ - Киев, Введ. 10.12.1998.

6. ДК - 003 - 95. Государственный классификатор профессий. [Электронный ресурс] : Утв. Госстандартом Украины 27.07.1995. / Режим доступу : www/ URL: http://posada.com.ua/useful/employer/5/12/ - 15.05.2012.

7. ГОСТ 12.0.003-74. Опасные и вредные производственные факторы классификации. [Электронный ресурс] : Утв. Госстандартом СССР 18.11.74 №2551 / Режим доступу : www/ URL: http://bud-inform.com.ua/norms/view/ssbt/370/ - 16.05.2012.

8. НПАОП 40.1-1.21-98. Правила безопасной эксплуатации электроустановок потребителей. [Электронный ресурс] : Утв. Госнадзорохрантруда Украины 10. 02. 98 №93/2533 / Режим доступу : www/ URL: http://www.dnaop.com/html/2029.html - 16.05.2012.

9. НПАОП 40.1-1.32-01. Правила устройства электроустановок. Электрооборудование специальных установок. [Электронный ресурс] : Утв. Министерства труда и социальной политики Украины 10. 02. 98 №272 / Режим доступу : www/ URL: http://3umf.com/doc/3134/ - 16.05.2012.

10. ДСН 3.3.6.042-99. Санитарные нормы микроклимата производственных помещений. [Электронный ресурс] : Утв. МОЗ Украины 01.12.1999 / Режим доступу : www/ URL: http://budstandart.com/read/document/show/3105034/id/ 3096960?submenu=10516 - 16.05.2012.

11. НАПБ Б.03.002-2007. Нормы определения категорий помещений, зданий и наружных установок по взрывопожарной и пожарной опасности. [Электронный ресурс] : Режим доступу : www/ URL: http://dbn.at.ua/load/21-1-0-315 - 16.05.2012.

12. ДБН В.1.1.7-2002. Пожарная безопасность объектов строительства. [Электронный ресурс] : Утв. Госстроем Украины 03.12.2012 №88 / Режим доступу : www/ URL: [http://dbn.at.ua/load/normativy/dbn/1-1-0-88 - 16.05.2012](http://dbn.at.ua/load/normativy/dbn/1-1-0-88%20-%2016.05.2012).

13. Android за 24 часа. Программирование приложений под операционную систему Google / Лорен Дэрси, Шейн Кондер, - Рид Групп, 2011. – 409 с.

14. Генетические алгоритмы, искусственные нейронные сети и проблемы виртуальной реальности / Г.К. Вороновкий, К.В. Махотило, С.Н. Петрашев, С.А. Сергеев. – Х.: ОСНОВА, 1997. – 112 с.

# ДОДАТОК А

# ВИХІДНИЙ КОД

## Підсистема прецедентів

**public** **class** Precedents {

**public** **static** **int** *cases\_counter* = 0;

**public** **final** **static** ArrayList<BoardDuelsCase> *loaded\_cases* =

**new** ArrayList<BoardDuelsCase>();

**class** BoardDuelsCase

{

**private** **final** **float**[] inputs;

**private** **final** **float**[] outputs;

**private** **float** potency;

**private** **float** frequency;

**private** **int** usage\_counter;

**private** **final** **int** id;

**public** BoardDuelsCase(**int** \_id, **float**[] inp, **float**[] out, **float** freq, **float** pot)

{

id = \_id;

inputs = Arrays.*copyOf*(inp, inp.length);

outputs = Arrays.*copyOf*(out, out.length);

frequency = freq;

potency = pot;

}

**public** **int** getId()

{

**return** id;

}

**public** **void** calculateStats(**boolean** isVictory)

{

frequency += usage\_counter / *cases\_counter*;

**if** (isVictory){

potency += 1 / *cases\_counter*;

}

}

**public** **boolean** isEffective(**float** neuro\_net\_potential)

{

**return** (0.7\*potency + 0.3\*frequency) > neuro\_net\_potential;

}

**public** **float**[] getInputs()

{

**return** **this**.inputs;

}

**public** **float**[] getOutputs()

{

**return** **this**.outputs;

}

**public** **float** getFullPotency()

{

**return** 0.7f\*potency + 0.3f\*frequency;

}

**public** **void** setPotency(**float** p, **float** freq)

{

**this**.potency = p;

**this**.frequency = freq;

}

}

**public** **void** calculateStats(**boolean** isVictory)

{

**for**(**int** i = 0; i < *loaded\_cases*.size(); i++)

{

*loaded\_cases*.get(i).calculateStats(isVictory);

}

}

**private** **void** copyFile(InputStream in, OutputStream out) **throws** IOException {

**byte**[] buffer = **new** **byte**[4096];

**int** read;

**while**((read = in.read(buffer)) != -1){

out.write(buffer, 0, read);

}

}

//Функция, которая переносит стандартную базу прецедентов из архива приложения на //карту памяти мобильного устройства

**public** **void** resetCases(Context call\_context) **throws** IOException, ParserConfigurationException, SAXException

{

InputStream fis =(InputStream)

call\_context.getAssets().open("cases.xml");

File sdPath = Environment.*getExternalStorageDirectory*();

// добавляем свой каталог к пути

sdPath = **new** File(sdPath.getAbsolutePath() + "/BoardDuels");

sdPath.mkdirs();

// формируем объект File, который содержит путь к файлу

File sdFile = **new** File(sdPath, "cases.xml");

sdFile.createNewFile();

OutputStream out = **new** FileOutputStream(sdFile);

copyFile(fis, out);

fis.close();

fis = **null**;

out.flush();

out.close();

out = **null**;

loadCases(call\_context);

}

//Функция сохранения прецедентов в файл

**public** **void** saveCases(Context call\_context) **throws** IOException, SAXException, ParserConfigurationException

{

DocumentBuilderFactory dbf = DocumentBuilderFactory.*newInstance*();

DocumentBuilder db = dbf.newDocumentBuilder();

Document doc = db.newDocument();

Element root = doc.getDocumentElement();

Element RootElement = doc.createElement("cases");

**for** (**int** i = 0; i < *loaded\_cases*.size(); i++)

{

BoardDuelsCase current\_case = *loaded\_cases*.get(i);

Element case\_element = doc.createElement("case");

case\_element.setAttribute("id", Integer.*toString*(i));

**for** (**int** j = 0; j < current\_case.inputs.length; j++)

{

Element input\_element = doc.createElement("input");

input\_element.appendChild(doc.createTextNode(Float.*toString*(current\_case.inputs[j])));

input\_element.setAttribute("id", Integer.*toString*(j));

case\_element.appendChild(input\_element);

}

**for** (**int** k = 0; k < current\_case.outputs.length; k++)

{

Element output\_element = doc.createElement("output"); output\_element.appendChild(doc.createTextNode(Float.*toString*(current\_case.outputs[k])));

output\_element.setAttribute("id", Integer.*toString*(k));

case\_element.appendChild(output\_element);

}

Element potency\_element = doc.createElement("potency");

potency\_element.appendChild(doc.createTextNode(Float.*toString*(current\_case.potency)));

case\_element.appendChild(potency\_element);

Element frequency\_element = doc.createElement("frequency");

frequency\_element.appendChild(doc.createTextNode(Float.*toString*(current\_case.frequency)));

case\_element.appendChild(frequency\_element);

RootElement.appendChild(case\_element);

}

doc.appendChild(RootElement);

File sdPath = Environment.*getExternalStorageDirectory*();

sdPath = **new** File(sdPath.getAbsolutePath() + "/BoardDuels");

// создаем каталог

sdPath.mkdirs();

// формируем объект File, который содержит путь к файлу

File sdFile = **new** File(sdPath, "cases.xml");

**try** {

// открываем поток для записи

BufferedWriter bw = **new** BufferedWriter(**new** FileWriter(sdFile));

// пишем данные

bw.write(doc.getTextContent());

// закрываем поток

bw.close();

Log.*d*("bdLog", "Файл записан на SD: " + sdFile.getAbsolutePath());

} **catch** (IOException e) {

e.printStackTrace();

}

}

//Функция загрузки прецедентов из файла

**public** **void** loadCases(Context call\_context) **throws** ParserConfigurationException, IOException, SAXException

{

DocumentBuilderFactory dbf = DocumentBuilderFactory.*newInstance*();

DocumentBuilder db = dbf.newDocumentBuilder();

String filename = "cases";

File sdPath = Environment.*getExternalStorageDirectory*();

// добавляем свой каталог к пути

sdPath = **new** File(sdPath.getAbsolutePath() + "/BoardDuels");

// формируем объект File, который содержит путь к файлу

File sdFile = **new** File(sdPath, "cases.xml");

FileInputStream fis = **new** FileInputStream(sdFile);

Document doc = db.parse(fis);

Element root = doc.getDocumentElement();

**float**[] inp = **new** **float**[14];

**float**[] out = **new** **float**[4];

**int** cases\_counter = -1;

NodeList cases = root.getChildNodes();

**for**(**int** i = 0; i < cases.getLength(); i++)

{

**if**(cases.item(i).getNodeName().equalsIgnoreCase("case"))

{

cases\_counter++;

Element current\_case = (Element)cases.item(i);

NodeList case\_data = current\_case.getChildNodes();

**int** inp\_counter = 0;

**int** out\_counter = 0;

**for**(**int** j = 0; j < case\_data.getLength(); j++)

{

**if**(case\_data.item(j).getNodeName().equalsIgnoreCase("input"))

{

inp[inp\_counter] = **new** Float(case\_data.item(j).getTextContent());

inp\_counter++;

}

**if**(case\_data.item(j).getNodeName().equalsIgnoreCase("output"))

{

out[out\_counter] = **new** Float(case\_data.item(j).getTextContent());

out\_counter++;

}

}

**float** p = **new** Float(((Element)cases.item(i)).getElementsByTagName("potency").item(0).getTextContent());

**float** freq = **new** Float(((Element)cases.item(i)).getElementsByTagName("frequency").item(0).getTextContent());

**final** BoardDuelsCase c = **new** BoardDuelsCase(cases\_counter,inp,out,freq,p);

*loaded\_cases*.add(c);

}

}

}

//Функция для поиска наиболее похожего случая в базе прецедентов

**public** **int** findClosestCase(**float**[] inp)

{

**float** delta = 0.0f;

**float** deltas = 0.0f;

**for**(**int** i = 0; i < *loaded\_cases*.size(); i++)

{

**for**(**int** j = 0; j < 14; j++)

{

deltas += Math.*abs*(inp[j] - *loaded\_cases*.get(i).inputs[j]);

}

delta = deltas / 14;

**if**(delta <= 0.05f)

{

**return** i;

}

}

**return** -1;

}

//Функция, возвращающая номера двух наиболее приближенных случаев

**public** **int**[] findTwoClosestCases(**float**[] inp)

{

**int**[] result = **new** **int**[2];

result[0] = -1;

result[1] = -1;

**float** delta = 0.0f;

**float** min\_delta = 1.0f;

**float** deltas = 0.0f;

**for**(**int** i = 0; i < *loaded\_cases*.size(); i++)

{

deltas = 0.0f;

**for**(**int** j = 0; j < 14; j++)

{

deltas += Math.*abs*(inp[j] - *loaded\_cases*.get(i).getInputs()[j]);

}

delta = deltas / 14.0f;

**if**(delta <= min\_delta)

{

result[0] = i;

min\_delta = delta;

}

}

delta = 0.0f;

deltas = 0.0f;

min\_delta = 1.0f;

**for**(**int** i = 0; i < *loaded\_cases*.size(); i++)

{

deltas = 0.0f;

**for**(**int** j = 0; j < 14; j++)

{

deltas += Math.*abs*(inp[j] - *loaded\_cases*.get(i).getInputs()[j]);

}

delta = deltas / 14.0f;

**if**(delta <= min\_delta)

{

**if**(i != result[0])

{

result[1] = i;

min\_delta = delta;

}

}

}

Random r = **new** Random();

**if** (result[0] < 0)

result[0] = r.nextInt(*loaded\_cases*.size());

**if** (result[1] < 0)

result[1] = result[0];

**return** result;

}

//Данная функция возвращает экземпляры двух наиболее похожих случаев

**public** BoardDuelsCase[] getClosestCases(**float**[] inp)

{

BoardDuelsCase[] cases = **new** BoardDuelsCase[2];

**int**[] closest = findTwoClosestCases(inp);

**if** (closest[1] >= 0 && closest[0] >= 0)

{

cases[0] = *loaded\_cases*.get(closest[0]);

cases[1] = *loaded\_cases*.get(closest[1]);

}

**return** cases;

}

}

## Підсистема нейронної мережі

public class NeuralNetwork {

public float X[];

public float Y[];

public Neuron W[];

public static class Neuron {

private float threshold;

private float[] x;

private float[] a;

private int[] my\_inputs;

public Neuron() {

threshold = 0.0f;

x = new float[1];

a = new float[1];

a[0] = 1.0f;

}

public float getSynapseWeight(int s)

{

return a[s];

}

public void getSignals(float signals[])

{

for(int i = 0;i < x.length;i++)

x[i] = signals[my\_inputs[i]];

}

//Функция установки входных сигналов нейрона

public void setInputs(int[] inputs)

{

my\_inputs = Arrays.copyOf(inputs, inputs.length);

}

//Функция создания нейрона сети

public void createNeuron(int axons, float threshold) {

x = new float[axons];

a = new float[axons];

my\_inputs = new int[axons];

this.threshold = threshold;

refreshSynapses();

}

//Функция для установки одного из входных сигналов

public void setInputSignal(int index, float value) {

x[index] = value;

}

//Функция установки величины синаптического веса

public void setSynapseSensivity(int index, float value) {

a[index] = value;

}

//Функция сброса синаптических весов (они становятся равными)

public void refreshSynapses() {

for (int i = 0; i < x.length; i++) {

x[i] = 0.0f;

a[i] = (float) 1 / x.length;

}

}

//Функция обучения нейрона

public void doBaseLearning(float inputs[], float model\_out,

int[] input\_priorities, float epsilon) {

int p\_sum = 0;

float my\_Y = 0.0f;

int most\_important = 0;

int max\_inp = 0;

float max\_input = -1;

int max\_p = -1;

float step = 2.0f - 2\*epsilon;

getSignals(inputs);

for (int i = 0; i < x.length; i++) {

p\_sum += input\_priorities[i];

if (x[i] > max\_input)

{

max\_inp = i;

max\_input = x[i];

}

if (input\_priorities[i] > max\_p) {

most\_important = i;

max\_p = input\_priorities[i];

}

}

for (int i = 0; i < x.length; i++) {

setSynapseSensivity(i, (float) input\_priorities[i] / p\_sum);

my\_Y += a[i] \* x[i];

}

while (Math.abs(model\_out - my\_Y) > epsilon) {

if (my\_Y < model\_out) {

for (int i = 0; i < x.length; i++) {

if (x[i] <= epsilon) {

a[max\_inp] += (float) (step / x.length) \* a[i] \* 0.5;

a[most\_important] += (float) (step / x.length) \* a[i] \* 0.5;

a[i] \*= (1 - (float) (step / x.length));

}

}

} else if (my\_Y > model\_out) {

for (int i = 0; i < x.length; i++) {

for (int k = 0; k < x.length; k++) {

if (i != k) {

if (x[i] < (x[k]+epsilon)) {

a[i] += (float) (step / x.length) \* a[k];

a[k] \*= (1 - (float) (step / x.length));

}

}

}

}

}

step \*= 0.25;

if(step <= (epsilon / x.length))break;

my\_Y = 0.0f;

for (int i = 0; i < x.length; i++) {

my\_Y += a[i] \* x[i];

}

}

my\_Y = 0.0f;

for (int i = 0; i < x.length; i++) {

my\_Y += a[i] \* x[i];

}

}

//Функция, возвращающая величину выходного сигнала нейрона(с учетом порога)

public float getSignal() {

float charge = 0.0f;

for (int i = 0; i < x.length; i++)

charge += (float)(x[i] \* a[i]);

if (charge < threshold)

return 0;

else

return charge;

}

};

//Конструктор нейронной сети

public NeuralNetwork(int neurons, int inputs)

{

W = new Neuron[neurons];

for(int i = 0;i < neurons;i++){

W[i] = new Neuron();

W[i].createNeuron(inputs, 0.0f);

}

Y = new float[neurons];

X = new float[inputs];

}

//Функция для задания массива входных сигналов сети

public void setInputs(float[] x)

{

X = Arrays.copyOf(x, x.length);

}

//Функция для назначения определенных входов сети указанному нейрону

public void assignInputs(int neuron\_number, int[] inputs)

{

W[neuron\_number].createNeuron(inputs.length, 0.0f);

W[neuron\_number].setInputs(Arrays.copyOf(inputs,inputs.length));

}

//Функция запуска процесса обучения для указанного нейрона с указанием его //приоритетов и эталонного выходного сигнала

public void startLearning(int neuron, float model\_Y, int[] priorities)

{

W[neuron].doBaseLearning(X,model\_Y, priorities, 0.05f);

}

//Функция запуска нейронной сети для получения новых выходных сигналов

public void launch()

{

for(int i = 0;i < W.length;i++)

{

W[i].getSignals(X);

Y[i] = W[i].getSignal();

}

}

}

## Підсистема прийняття рішень

**public** **int** chooseCardToUse(ImageAdapter hands,Hero target)

{

//По умолчанию, в начале хода все приоритеты действий равны

**float** attack\_p = 1.0f;

**float** defense\_p = 1.0f;

**float** major\_att\_p = 1.0f;

**float** major\_def\_p = 1.0f;

//Вызываем функцию переобучения нейронной сети, чтобы //адаптировать ее к текущей игровой ситуации и получить //приоритеты действий

reLearn(target);

//Считываем полученные от нейронной сети приоритеты

attack\_p = net.Y[0];

defense\_p = net.Y[1];

major\_att\_p = net.Y[2];

major\_def\_p = net.Y[3];

//Формируем цепочку действий в порядке их текущей важности

**boolean** isAttack = attack\_p > defense\_p;

**boolean** isMajorAtt = (major\_att\_p >= major\_def\_p) && isAttack && (major\_att\_p > attack\_p\*0.5);

**boolean** isMajorDef = (major\_def\_p > major\_att\_p) && !isAttack && (major\_def\_p > defense\_p\*0.5);

**float** major\_cards = getMajorAttCards\_percentage() + getMajorDefCards\_percentage();

//Здесь начинается, собственно, принятие решения и выбор //используемой карты в текущем ходу

**for**(**int** i = (cards.size()-1);i >= 0;i--)

{

SpellCard current\_card = cards.get(i);

**if**(major\_cards > 0.0f)

{

**if**(current\_card.isMajor && isAttackCard(i) == isMajorAtt && current\_card.inHands && current\_card.isActive)

{

useSpellCard(i,target);

FullscreenGameplayActivity.*echo*("SPELL CARD USED:\n" + current\_card.name);

removeCardFromHands(hands,i);

*turns\_history\_writer*.addUsedCardInfo(current\_card.name);

**return** current\_card.card\_id;

}

**else**

{

**if**(current\_card.isMajor && !isAttackCard(i) && isMajorDef && current\_card.inHands && current\_card.isActive)

{

useSpellCard(i,target);

FullscreenGameplayActivity.*echo*("SPELL CARD USED:\n" + current\_card.name);

removeCardFromHands(hands,i);

*turns\_history\_writer*.addUsedCardInfo(current\_card.name);

**return** current\_card.card\_id;

}

}

}

**else**

{

**if**(isAttackCard(i) == isAttack && current\_card.inHands && current\_card.isActive && (ep >= current\_card.cost))

{

useSpellCard(i,target);

FullscreenGameplayActivity.*echo*("SPELL CARD USED:\n" + current\_card.name);

removeCardFromHands(hands,i);

*turns\_history\_writer*.addUsedCardInfo(current\_card.name);

**return** current\_card.card\_id;

}

}

}

//in case if the most useful card wasn't found - AI will use any active card if he has enough energy

**for**(**int** i = 0;i < cards.size();i++)

{

**if**(cards.get(i).inHands && cards.get(i).isActive && (ep >= cards.get(i).cost))

{

useSpellCard(i,target);

FullscreenGameplayActivity.*echo*("SPELL CARD USED:\n" + cards.get(i).name);

removeCardFromHands(hands,i);

*turns\_history\_writer*.addUsedCardInfo(cards.get(i).name);

**return** cards.get(i).card\_id;

}

}

FullscreenGameplayActivity.*AI\_step\_passed* = **true**;

**return** -1;

}

Далее приведен листинг функции reLearn(), которая вызывается вышеуказанной функцией для запуска переобучения нейронной сети. Перед началом, собственно, переобучения сети, следующая функция обращается к подсистеме прецедентов, запрашивая найти наиболее похожие случаи. После этого полученные случаи аппроксимируются, и параметры усредненного прецедента передаются системе обучения нейронной сети.

**public** **void** reLearn(Hero target)

{

**float** inputs[] = **new** **float**[14];

//Ниже описан смысл каждого из входов нейронной сети в

//игровом приложении Board Duels

inputs[0] = getHP\_percentage();

inputs[1] = target.getHP\_percentage();

inputs[2] = getEP\_percentage();

inputs[3] = target.getEP\_percentage();

inputs[4] = target.getCards\_percentage();

inputs[5] = getDeck\_percentage();

inputs[6] = target.getDeck\_percentage();

inputs[7] = getAttCards\_percentage();

inputs[8] = getDefCards\_percentage();

inputs[9] = getMajorAttCards\_percentage();

inputs[10] = getMajorDefCards\_percentage();

inputs[11] = getCards\_percentage();

inputs[12] = 1 - inputs[1];

inputs[13] = 1 - inputs[0];

//Здесь мы получаем наиболее похожие случаи из базы прецедентов

BoardDuelsCase[] found\_cases = *precedents*.getClosestCases(inputs);

**float**[] found\_outputs = **new** **float**[4];

//Далее 2 найденных случая аппроксимируются

**for**(**int** i = 0; i < 4; i++)

{

found\_outputs[i] = 0.5f\*(found\_cases[0].getOutputs()[i] + found\_cases[1].getOutputs()[i]);

}

//Назначаются входы для нейронной сети и передаются параметры //усредненного прецедента

**int** all\_inputs[] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13};

net.setInputs(inputs);

net.assignInputs(0, all\_inputs);

net.assignInputs(1, all\_inputs);

net.assignInputs(2, all\_inputs);

net.assignInputs(3, all\_inputs);

net.startLearning(0, found\_outputs[0], **new** **int**[]{1,3,3,-1,-2,2,1,3,-2,2,2,2,3,-5});

net.startLearning(1, found\_outputs[1], **new** **int**[]{1,2,1,3,3,1,3,-1,2,2,2,2,-2,3});

net.startLearning(2, found\_outputs[2], **new** **int**[]{-2,3,1,-1,1,2,1,3,2,3,2,3,2,2});

net.startLearning(3, found\_outputs[3], **new** **int**[]{-1,2,1,3,3,2,3,2,2,1,2,1,-2,3});

//После переобучения сети она запускается для получения //актуальных выходных сигналов

net.launch();

//Производится запись выполненных действий в журнал

*turns\_history\_writer*.addTurnInfo(net, found\_cases[0].getId(), found\_cases[1].getId());

}