МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Львівський національний університет імені Івана Франка
Факультет електроніки та комп'ютерних технологій
Кафедра системного проектування

Допустити до захисту

Завідувач кафедри
_____ доц. Шувар Я. Р.
«___» _____2023 р.

Курсова робота

Магістр
(освітній ступінь)

Використання нейронних мереж для створення шахового рушія

Виконав:
студент групи ФЕС – 42
спеціальності 126 – Інформаційні системи та технології

_____ **Остріков Д. А**.

Науковий керівник:
_____ **асис. Парубочий В. О.**
«____» _____2023 р.

Львів 2023

АНОТАЦІЯ

В курсовій роботі проаналізовано існуючі шахові двигуни, що використовують нейронні мережі, такі як Stockfish та Leela Chess Zero. На основі отриманих знань було створено шаховий двигун та графічний інтерфейс, до якого можна підключити створений двигун. Пошук оптимального ходу в цьому двигуні реалізовано на основі найпростішого алгоритму. Функцію оцінки позиції реалізовано на основі доволі простої з архітектурної точки зору глибокої нейронної мережі прямого поширення, навчання якої зайняло близько 5 годин. Отриману модель було проаналізовано та перевірено в матчі проти шахового двигуна Stockfish 14, який був сконфігурований на 2-ий рівень складності. У висновку наведено можливі способи покращення отриманого шахового двигуна.

ABSTRACT

This diploma work contains analysis of existing chess engines that use neural network, such as Stockfish and Leela Chess Zero. New chess engine and GUI, that can use created engine, were implemented based on the obtained results of the analysis. Searching for the best move was implemented based on the simpliest algorithm. Evaluation function was implemented using quite simple from an architectural point of view deep sequential neural network, which training took nearly 5 hours. Obtained model was analyzised and tested in game versus chess engine Stockfish 14, that had been configured on 2nd level of difficulty. Possible ways to improve created chess engine are described in conclusion section.

ЗМІСТ

ВСТУП 4

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 6

1. ОГЛЯД ІСНУЮЧИХ ШАХОВИХ ДВИГУНІВ 7

1.1. Stockfish NNUE 7

1.1.1. Загальні відомості 7

1.1.2. Структура NNUE 8

1.1.3. Нейронна мережа 12

1.1.4. Переваги Stockfish NNUE 13

1.2. Leela Chess Zero 13

1.2.1. Загальні відомості 13

1.2.2. Історія створення Leela Chess Zero 15

1.2.3. Подання дошки в Leela Chess Zero 16

1.2.4. Нейронна мережа 16

1.2.5. Навчання нейронної мережі 17

1.2.6. WDL оцінка 18

1.3. Порівняння Leela Chess Zero і Stockfish 20

2. СТВОРЕННЯ ВЛАСНОГО ШАХОВОГО ДВИГУНА 22

2.1. Постановка задачі 22

2.2. Оцінка наявних ресурсів та можливостей 22

2.3. Проектування архітектури шахового двигуна 23

2.3.1. Комунікація з графічним інтерфейсом 23

2.3.2. Пошук оптимального ходу 25

2.3.3. Функція оцінювання позиції 25

2.3.4. Програма з графічним інтерфейсом для гри проти двигуна .. 26

2.4. Збір та обробка даних 28
2.5. Архітектура нейронної мережі 29
2.6. Навчання нейронної мережі 35
2.7. Аналіз результатів навчання 36
2.8. Тестування шахового двигуна 38
ВИСНОВКИ 42
ВИКОРИСТАНА ЛІТЕРАТУРА 43

ВСТУП

Останнім часом все більше і більше задач вирішуються з використанням елементів машинного навчання: розпізнавання образів, розпізнавання голосу, прийняття рішень тощо. Все це пов’язано із постійним технологічним прогресом, через що, сьогодні нам доступні в сотні раз потужніші обчислювальні машини. Причини популярності застосування машинного навчання зрозумілі:

- Команда DeepMind довела, що для навчання комп’ютера грі в шахи на надлюдському рівні не потрібно поглиблених знань гри в шахи;
- В деяких випадках використання нейронних мереж значно економить час, адже зменшується необхідність експертних знань для вирішення задачі. Узагальнення необхідних знань перекладається на комп’ютер.
- При вирішенні певних задач даний метод показує кращі результати порівняно із традиційними способами. Прикладом може слугувати шаховий двигун AlphaZero, який перевершив вручну оптимізовану функцію оцінки позиції Stockfish і показав, що навіть у такій добре вивченій грі ще існують невивчені місця.

Актуальність теми полягає у використанні нейронних мереж для вирішення завдань в інтелектуальному аналізі даних. У роботі розглянуто технологію навчання нейронної мережі для оцінки позиції на шаховій дошці, що є актуальним для пошуку раніше не вивчених стратегій, які можна узагальнити і використовувати в інших галузях. Так як найсильніший шаховий двигун можна реалізувати обов’язково із використанням елементів машинного навчання, маючи такий двигун, можна створити різні платформи для професійного аналізу ігор, навчання грі в шахи; для змагань з іншими шаховими двигунами. Отримавши модель, яка показує хороші результати в грі в шахи, можна спробувати її узагальнити так, щоб вона могла навчитись грати не тільки в шахи, а й в ґо (стародавню китайську стратегічну настільну гру).

Об’єктом дослідження є архітектура та принципи роботи існуючих шахових двигунів.

Предметом наукового дослідження є проблема пошуку оптимального ходу в грі шахи.

Мета курсової роботи: створення власного шахового двигуна.

Для досягнення мети поставлено такі завдання:

1. Проведення аналізу літературних джерел з тематики роботи.
2. Аналіз існуючих шахових двигунів, способів оцінки позиції та пошук оптимального ходу.
3. Обґрунтування використаних засобів розробки.
4. Створення програми шахового двигуна, яку можна буде інтегрувати в графічний інтерфейс.
5. Використання нейронної мережі для пошуку оптимального ходу.
6. Порівняння результатів з існуючими двигунами.
7. Створення плану щодо подальших досліджень даної теми.

Робота складається із двох розділів. У першому розглянуто та порівняно два популярних шахові двигуни Stockfish та Leela Chess Zero. У другому розділі було створено шаховий двигун, проаналізовано отримані результати та проведено гру проти Stockfish, сконфігурованого на другий рівень складності.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AB – Alpha-Beta

UCI – Universal Chess Interface

NNUE – Efficiently Updatable Neural Networks

SF – Stockfish

LCZero – Leela Chess Zero

NN – Neural Network

CNN – Convolutional Neural Network

DNN – Deep Neural Network

DCNN – Deep Convolutional Neural Network

SIMD – Single Instruction Multiple Data

UCT – Upper Confidence bounds applied to Trees

L1 – Least Absolute Deviation

L2 – Least Square Errors

WDL – Win Draw Lose

nps – nodes per second

ГП – графічний процесор

ЦП – центральний процесор

1. ОГЛЯД ІСНУЮЧИХ ШАХОВИХ ДВИГУНІВ

1.1. Stockfish NNUE

1.1.1. Загальні відомості

Stockfish NNUE – це версія Stockfish від Хісайорі Нода, яка використовує NNUE (efficiently updatable neural network), ефективно обновлювану нейронну мережу [1]. Незважаючи на приблизно вдвічі меншу швидкість пошуку, Stockfish NNUE стала сильнішою за свій оригінал.

У серпні 2020 року Fishtest показав, що Stockfish NNUE сильніший за класичну версію принаймні на 80 Ело. Того ж місяця Stockfish змінили стандартний метод оцінки позиції на гібридний – NNUE використовується лише в матеріально рівних позиціях, в інших випадках використовується класична оцінка позиції. Навчальний код, як і раніше, залишався в репозиторії Nodchip. Репозиторій Nodchip також надає додаткові інструменти для навчання та розробки мереж NNUE [2]. З часом цей код був замінений на PyTorch.

Зараз в Stockfish доступні як NNUE, так і класична функція оцінки, які можна використовувати для визначення позиції значення, яке згодом використовується в альфа-бета пошуку для знаходження кращого ходу [3].

Класична оцінка розраховує це значення як функцію різних шахових концепцій, розроблених експертами вручну, протестованих та налаштованих за допомогою Fishtest. Fishtest – платформа для тестування двигуна Stockfish шляхом гри між різними версіями цього двигуна. Оцінка NNUE визначає це значення за допомогою нейронної мережі на основі базових вхідних даних. Мережа оптимізована та навчена на оцінках мільйонів позицій при помірній глибині пошуку

Оцінка NNUE була вперше представлена в сьогі, а потім перенесена в Stockfish. Вона може бути ефективно оцінена на центральних процесорах та використовує той факт, що тільки частини нейронної мережі мають бути оновлені після типового шахового ходу.

1.1.2. Структура NNUE

Нейронна мережа складається із чотирьох шарів. Вхідний шар сильно перепараметризований, оскільки крім розміщення фігур на вхід подаються всі можливі ходи обох королів.

Перша NNUE була із використанням структури HalfKP (рис. 1.1, 1.2). Структура HalfKP складається з двох половин, що охоплюють вхідний шар і перший прихований шар, кожна половина вхідного шару пов'язана з одним із двох королів, перехресно з'єднаних із половинами першого прихованого шару. Для кожного розміщення чорного або білого короля, десяти фігур без королів на їх певних клітинках є логічними {0,1} входами, поряд з реліктом від викладання фігур сьогі, 64 x (64 x 10 + 1) = 41024 входи для кожної половини, які множаться на 16-бітний цілий вектор ваги для 256 виходів на половину, загалом 256 x 41 024 = 10 502 144 ваги [1].

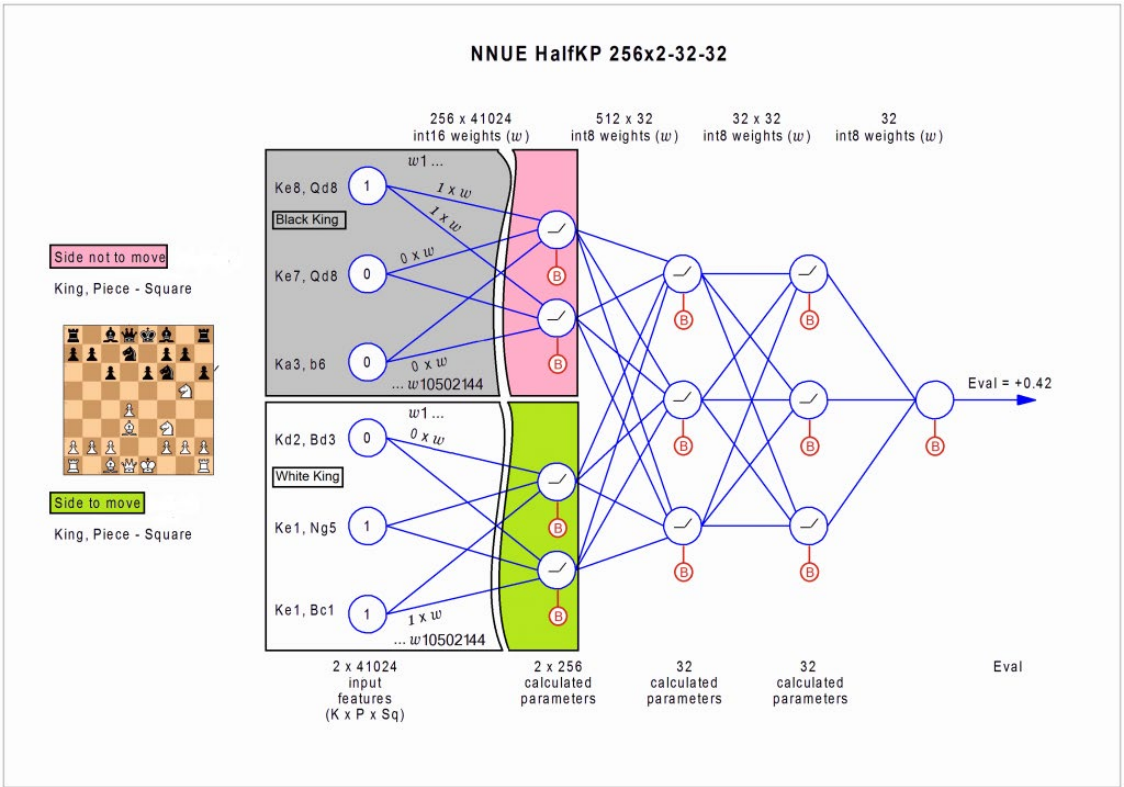

Рис. 1.1. Шари NNUE в дії

Ефективність NNUE зумовлена інкрементним оновленням виходів вхідного шару при виконанні та невиконанні ходів, коли тільки крихітна частина його нейронів має бути розглянута у випадку, якщо немає ходів короля. Інші три шари з вагами 2x256x32, 32x32 і 32x1 обчислювально менш затратні, приховані шари застосовують активацію ReLu, найкраще обчислюються з використанням відповідних SIMD інструкцій, що виконують швидку 8-бітну/16-бітну цілочисельні операції MMX, SSE2 або AVX2 на x86/x86-64, або, якщо є, AVX-512 [4].

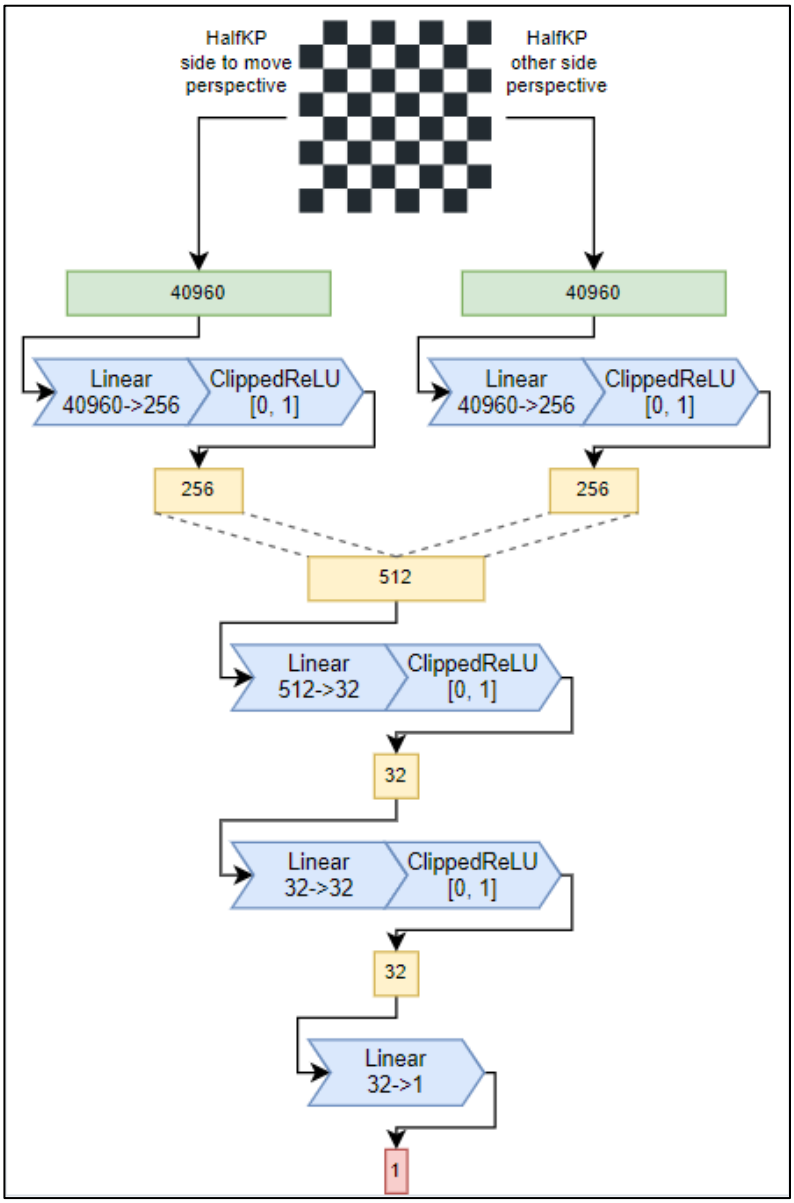

Рис. 1.2. Архітектура HalfKP

Накопичувач має половину «білого короля» і половину «чорного короля», де кожна половина являє собою 256-елементний вектор 16-бітових даних цілочисельного типу, що дорівнює сумі ваг активних ознак та 256-елементного вектора 16-бітового зміщення.

Етап «перетворення» оцінки NNUE формує 512-елементний вектор із 8-бітовими числами, де перша половина формується з 256-елементного вектора сторони, чия черга ходити, а друга половина формується з 256-елементного вектора для іншої сторони. На цьому кроці 16-бітові елементи обрізаються до значення від 0 до 127. Це вихідний результат вхідного шару.

Цей 512-елементний вектор 8-розрядних чисел потім множиться на матрицю 32x512 8-бітових ваг, щоб отримати 32-елементний вектор 32-бітових чисел, до якого додається вектор 32-бітових зміщень. Вектор суми ділиться на 64 і обрізається до 32-елементного вектора з 8-бітовими числами від 0 до 127. Це вихідний результат першого прихованого шару.

Отриманий 32-елементний вектор 8-розрядних даних цілого типу множиться на матрицю 8-бітових ваг 32x32, щоб отримати 32-елементний вектор із 32-бітовими даними цілочисельного типу, до якого додається інший вектор 32-бітових зміщень. Ці цілі числа знову діляться на 64 і обрізаються до 32 8-бітових чисел від 0 до 127. Це вихідний результат другого прихованого шару.

Цей 32-елементний вектор із 8-бітовими числами потім множиться на матрицю 1x32 8-бітових ваг (тобто береться добуток двох векторів). Це створює 32-бітове значення, до якого додається 32-бітове зміщення. Це і є результатом вихідного шару.

Результат вихідного шару ділиться на 16 для отримання оцінки NNUE. Оцінка SF потім виконує деякі подальші кроки, такі як додавання бонусу темпу (незважаючи на те, що оцінка NNUE по суті вже враховує сторону для

переміщення на етапі "перетворення") і масштабування оцінки до нуля, оскільки кількість ходів із правила про 50 ходів наближається до 50 [4].

У наступних версіях Stockfish архітектура мережі була вдосконалена Томашем Собчиком та ін. для перспективи з іншого боку, далі використовуючи вертикальний переворот замість повороту HalfKP. Використаний у Stockfish 14 алгоритм HalfKAv2 (рис. 1.3) економить трохи пам’яті, враховуючи надмірність королівських клітинок, використовуючи 11x64x64 = 45056 входів на кожну сторону, відображену на 2x520 лінійному трансформаторі ознак, в подальшому передаючи 8x2 вихід з цього трансформатора ознак напряму в результат вихідного шару для кращого навчання незбалансованих матеріальних конфігурацій [4]. Іншим удосконаленням було використання восьми вихідних підмереж 512x2 → 16 → 32 → 1, дискримінованих по (кількість_фігур - 1) / 4 у діапазоні [0;7].

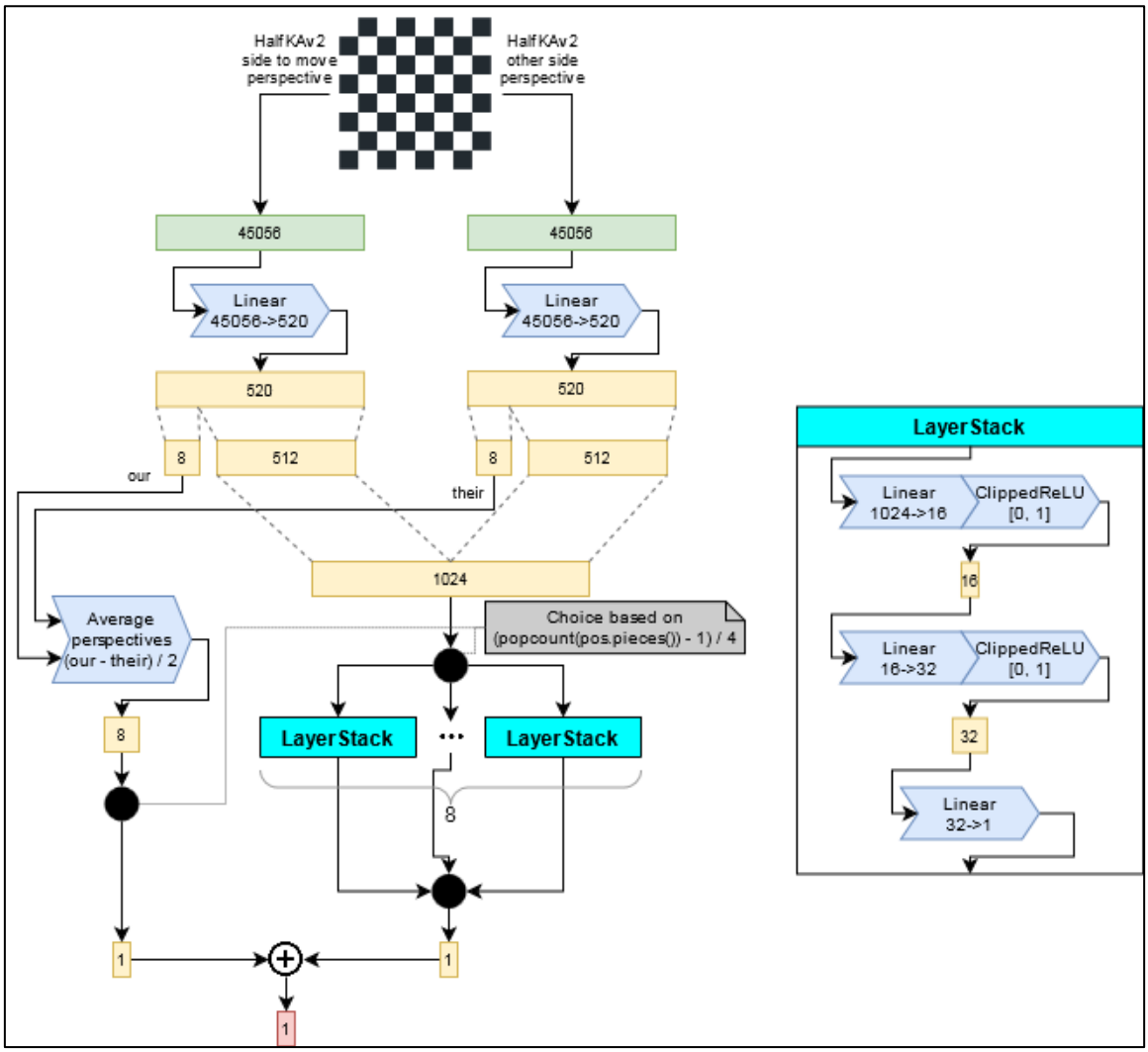

Рис. 1.3. Архітектура HalfKAv2 від Томаша Собчика

1.1.3. Нейронна мережа

Тестові навчені нейронні мережі були побудовані добровольцями та завантажені у Fishtest для тестування. Мережі з хорошими результатами тестування офіційно публікуються на сайті Fishtest із середньою швидкістю два тижні на мережу [6]. Після довгих обговорень найкращого способу публікації мереж у Stockfish, команда розробників вирішила вбудувати мережу за замовчуванням у двійкові файли Stockfish, щоб гарантувати, що NNUE завжди працює, а також забезпечити більшу зручність для користувачів.

Наприкінці 2020 року Гері Лінскотт розпочав реалізацію навчання Stockfish NNUE у PyTorch, використовуючи ресурси GPU для ефективного

навчання мереж. Далі співпраця з командою Leela Chess Zero у лютому 2021 року окупилася, надавши мільярди позицій для навчання нових мереж.

1.1.4. Переваги Stockfish NNUE

Stockfish NNUE працює тільки на CPU, не вимагає дорогих відеокарт, а також необхідності встановлення драйверів та специфічних бібліотек. Таким чином, він набагато простіший в установці (порівняно з двигунами, що використовують глибокі згорткові нейронні мережі, такими як Leela Chess Zero). Використання GPU навіть недоцільно для такої маленької мережі – накладні витрати на запуск ядра.

NNUE потребує невеликих навчальних наборів. Деякі високо результативні мережі можуть бути побудовані зусиллями однієї або кількох людей протягом кількох днів. Не вимагає величезних обчислень на суперкомп'ютері [1].

Використання NNUE не вимагає складних систем, таких як складна модель клієнт-сервер для навчання мереж. Для навчання достатньо одного бінарного файлу з GitHub репозиторію Nodchip [1].

Код NNUE незалежний і може бути легко відокремлений та інтегрований в інші двигуни.

1.2. Leela Chess Zero

1.2.1. Загальні відомості

Leela Chess Zero (LCZero) – адаптація проекту Leela Zero Go Джан-Карло Паскутто для шахів, ініційована та анонсована співавтором Stockfish Гарі Лінскоттом, який уже відповідав за Stockfish Testing Framework під назвою Fishtest. Leela Chess має відкритий вихідний код, що випускається на умовах GPL версії 3 або пізнішої і підтримує UCI. Мета LCZero полягає в тому, щоб побудувати сильну шахову сутність, використовуючи той же тип глибокого

навчання разом з методами дерева пошуку Монте-Карло (MCTS) AlphaZero, як описано в роботах DeepMind 2017 та 2018 років [7], але з використанням розподіленого навчання для ваг глибокої згроткової нейронної мережі [8].

LCZero складається з програми для гри та аналізу партій. Цей виконуваний файл, який і є шаховим двигуном, виконує MCTS і зчитує самонавчальну CNN, ваги якого зберігаються в окремому файлі. LCZero написаний на C++ (початий з C++14, потім оновлений до C++17) і може бути скомпільований для різних платформ. Оскільки глибокі CNN найкраще підходять для масового паралельного виконання на графічних процесорах для виконання всіх точкових продуктів з плаваючою точкою для тисяч нейронів, доцільними цільовими платформами є графічні процесори Nvidia, що підтримують CUDA та бібліотеки CuDNN. GPU, що не підтримують CUDA (AMD), підтримуються через OpenCL, у той час як набагато повільніші бінарні файли розраховані суто для використання на CPU можливі з використанням BLAS, цільовими системами з графічною картою (GPU) або без неї є комп'ютери Linux, Mac OS та Windows, або тільки BLAS – Raspberry Pi [8].

Як і AlphaZero, LCZero оцінює позиції за допомогою нелінійної апроксимації функцій на основі глибокої нейронної мережі, а не лінійної апроксимації функцій, як це використовується в класичних шахових програмах. Ця нейронна мережа приймає на вхід позицію на дошці та видає оцінку позиції (QValue) та вектор ймовірностей ходів (PValue, policy). Після навчання ця мережа комбінується з деревом пошуку Монте-Карло (MCTS), використовуючи стратегію (policy) для звуження пошуку до високоймовірних ходів, і використовуючи значення у поєднанні з політикою швидкого розвороту для оцінки позицій у дереві. Вибір MCTS здійснюється за допомогою варіації покращення PUCT, Predictor + UCT (алгоритм Upper Confidence bounds applied to Trees) [8].

1.2.2. Історія створення Leela Chess Zero

Проект Leela Chess Zero був вперше анонсований на TalkChess.com 9 січня 2018 року [8]. Це було представлення LCZero як шахового двигуна з відкритим вихідним кодом, що самонавчається. Протягом перших кількох місяців навчання LCZero вже досяг рівня гросмейстера, перевершивши силу ранніх релізів Rybka, Stockfish та Komodo, незважаючи на оцінку на порядки меншої кількості позицій завдяки глибокій нейронній мережі у функції оцінки та використання пошуку по дереву Монте-Карло [9] .

У грудні 2018 року AlphaZero опублікувала нову статтю в журналі Science, що розкриває раніше нерозкриті деталі архітектури та параметрів навчання, що використовуються для AlphaZero. Ці зміни незабаром були включені до Leela Chess Zero і збільшили як його силу, так і ефективність навчання.

Двигун був переписаний і ретельно ітерований з моменту його створення, і тепер працює на декількох бекендах, даючи змогу ефективно використовувати різні типи апаратного забезпечення як CPU, так і GPU.

Розробники використали метод навчання з підкріпленням. Це алгоритм машинного навчання, використовуваний навчальною програмою LCZero для максимізації винагороди за допомогою самостійної гри. Користувачі-добровольці запускають LCZero, щоб помогти в навчанні моделі, граючи сотні мільйонів ігор, які надходять до алгоритму підкріплення. Щоб зробити свій внесок у розвиток двигуна LCZero, необхідно завантажити останню версію двигуна, а також клієнт. Клієнт необхідний для підключення до поточного серверу LCZero, де зберігається вся інформація про самостійні шахові партії, для отримання останньої мережі, генерації самостійних партій та завантаження даних навчання назад на сервер.

Для того щоб грати проти двигуна LCZero на машині, необхідні дві компоненти: бінарний файл двигуна і мережа. Бінарний файл двигуна відрізняється від клієнта тим, що клієнт використовується як платформа для

навчання двигуна. Мережа містить функцію оцінки LCZero, необхідну оцінки позицій.

1.2.3. Подання дошки в Leela Chess Zero

Колірна дошка LCZero представлена п'ятьма бітовими дошками (власні фігури, фігури супротивника, ортогональні розсувні фігури, діагональні розсувні фігури та пішаки, включаючи інформацію про прохідних пішаків, закодовану як пішаки 1 та 8 рангу), двома клітинами короля, правом на рокірування та бітовим прапорцем про те, чи перевернута дошка відповідно до кольору, чий зараз хід. Для отримання бітових дошок окремих фігур виконуються деякі операції над множинами, такі як перетин та симетрична різниця.

1.2.4. Нейронна мережа

У той час як AlphaGo використовувала дві несхожі мережі для стратегії та цінності, AlphaZero, як і LCZero, мають спільне "тіло", поєднане з несхожими "головами" політики та цінності. Тіло складається з просторових площин 8x8, використовують залишкові блоки з фільтрами з розміром ядра 3x3. До цього часу використовувалися розміри моделей 64x6, 128x10, 192x15 і 256x20 [8].

Щодо кількості вузлів за секунду в MCTS, менші моделі обчислюються швидше, ніж великі. Вони швидше навчаються і раніше розпізнають прогрес, але вони також раніше насичуються, так що рано чи пізно подальше навчання вже не поліпшить роботу двигуна. Більші та глибокі мережеві моделі покращать сприйнятливість, кількість знань та шаблонів, які можна витягти з навчальних зразків. Як подальше поліпшення LCZero застосовує розширення Squeeze and Excite (SE) до архітектури залишкових блоків. Тіло підключено як до "голови" політики для розподілу ймовірності ходу, так і до "голови"

значення для оцінки позиції (ймовірності виграшу в поточній позиції) до семи попередніх позицій на вхідних площинах [8].

1.2.5. Навчання нейронної мережі

Як і в AlphaZero, Leela Chess Zero не використовує жодних початкових знань, крім правил гри, щоб створити надлюдського гравця. Leela Chess Zero починає навчання з дійсно випадкових ігор проти самого себе, щоб застосувати навчання з підкріпленням на основі результатів цих ігор. Однак існують похідні підходи, такі як Deus X Альберта Сільвера, які намагаються піти коротким шляхом, використовуючи методи контрольованого навчання, такі як подача високоякісних партій, зіграних іншими сильними шахістами, або величезні записи позицій із заданим найчастішим ходом [8]. Неконтрольоване навчання NN зводиться до мінімізації середньоквадратичної помилки втрати значення на виході та втрати політики. Зараз проводяться експерименти щодо навчання «голови» цінності не за результатом гри, а за накопиченим значенням для позиції після дослідження деякої кількості вузлів за допомогою UCT [10].

Розподілене навчання реалізовано за допомогою складної клієнт-серверної моделі. Клієнт, що повністю написаний мовою програмування Go, використовує LCZero для створення партій зіграних із самим собою. Керований сервером, клієнт може завантажити останню мережу, почати гру із самим собою і завантажувати ігри на сервер, який, з іншого боку, регулярно створюватиме і поширюватиме нові ваги нейронної мережі після певної кількості ігор, доступних від учасників. Програмне забезпечення для навчання складається з коду Python, для конвеєра (pipeline) потрібні NumPy і TensorFlow, що працюють на Linux. Сервер написаний на Go разом із Python та shell-скриптами.

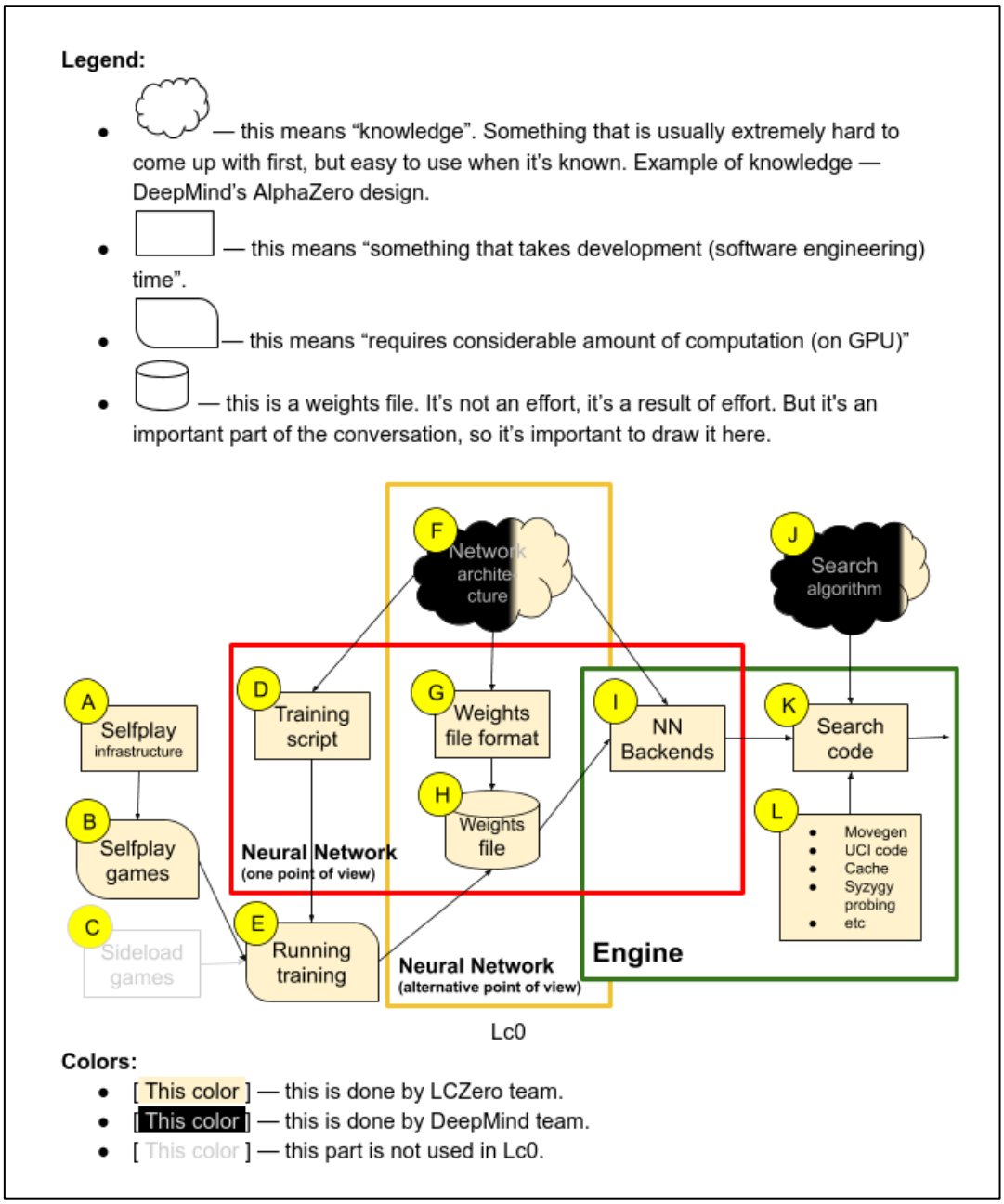

Рис. 1.4. Структура Leela Chess Zero

1.2.6. WDL оцінка

Традиційно, шахові рушії використовували поняття пішак для оцінки позицій. Але можна також проводити оцінку позиції з наступним форматом результату: відсоткова ймовірність перемогти, закінчити в нічию та програти у даній позиції. Команда Lc0 вирішила це реалізувати. Всі мережі, починаючи з липня 2019 року, мають так звану WDL-голову та здатні передбачати ймовірності перемоги, нічиї та поразки окремо [11].

На сьогоднішній день більшість шахових графічних інтерфейсів все ще очікують оцінки в (санти) пішаках, і відобразити за допомогою них ймовірності WDL неможливо. Але дану інформацію можна опрацьовувати за допомогою info команди протоколу UCI. Візуалізація WDL помагає зрозуміти, як LCZero сприймає позицію. Приклад візуалізації цього параметру можна побачити на рис. 1.4.а-в.

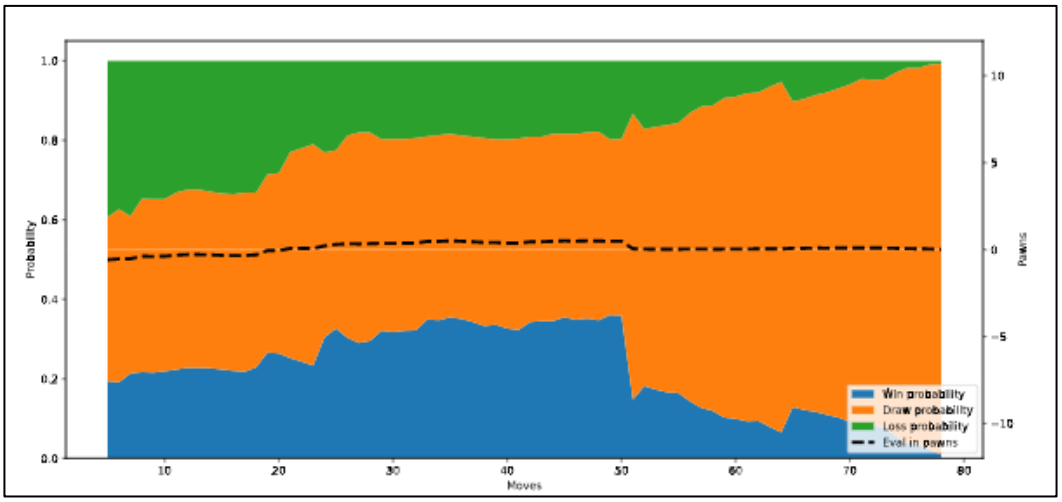

Рис. 1.4.а) TCEC 17 Superfinal Game 53 – LCZero до 50 ходу вважав, що має хороші шанси на перемогу і що шанс на нічию менше 50%

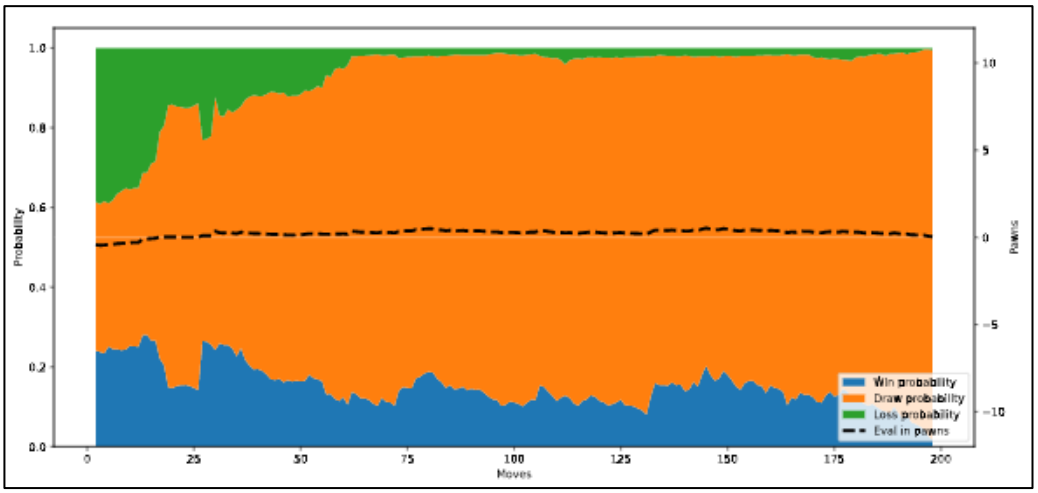

Рис. 1.4.б) TCEC 17 Superfinal Game 71 – LCZero майже від самого початку розумів, що партія закінчиться в нічию

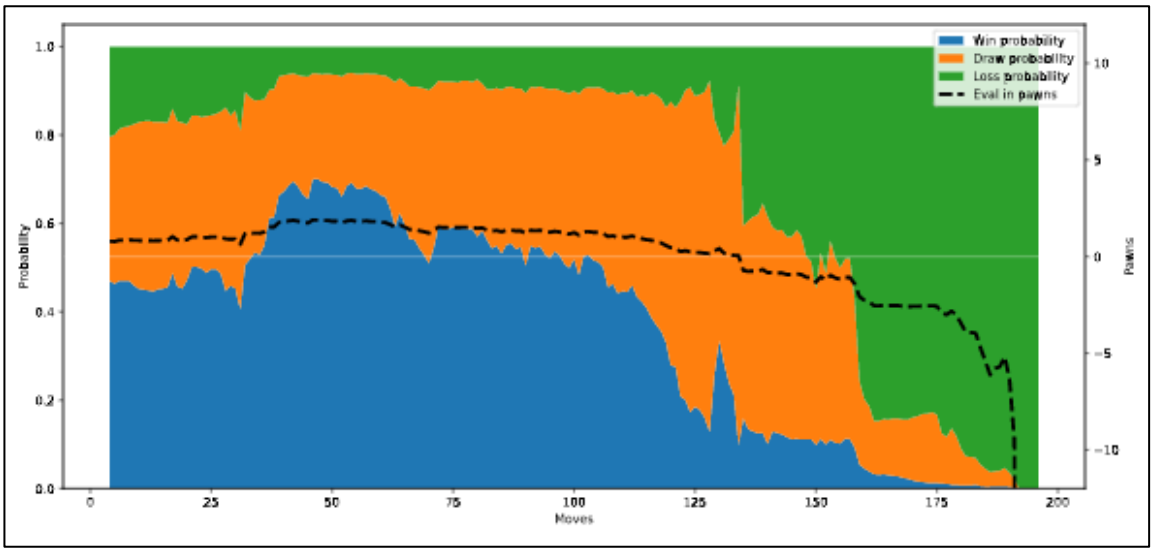

Рис 1.4.в) TCEC 17 Superfinal Game 16 – оцінка WDL набагато раніше показала, що LCZero втрачає контроль над ситуацією

1.3. Порівняння Leela Chess Zero і Stockfish

AlphaZero, LCZero, Stockfish використовують нейронні мережі для оцінки позицій. Але LCZero та AlphaZero також використовують інший алгоритм пошуку. Leela та AlphaZero використовують модифікований пошук дерева Монте-Карло, тоді як усі механізми NNUE використовують традиційну техніку AB-мінімаксного пошуку.

LCZero, в свою чергу, не може використовувати AB-мінімакс. Причиною цього є повільність обчислення: Stockfish NNUE набирає приблизно 100 Mnps, тоді як LCZero – близько 75 Knps. Stockfish може здійснити ефективний пошук AB, оскільки він може переглядати величезну кількість позицій. LCZero повільніший, тому доводиться використовувати алгоритм PUCT.

Stockfish набагато швидший за LCZero, тому що використовує порівняно малу мережу NNUE (близько 50 Кб), тому оцінка не займає багато часу. Мережі LCZero набагато, набагато більші (50-100 МБ). В Stockfish є також частина, яка ефективно оновлюється, яка прискорює роботу двигуна.

Перевага NNUE полягає у відсутності необхідності перераховувати всі обчислення з нуля. LCZero робить це, навіть коли гравці роблять прості повторні ходи, позиція переоцінюється та перераховується. Це дуже затратна робота.

Stockfish NNUE, з іншого боку, турбує лише перерахунок важливих частин мережі і припускає, що частини, які не є важливими, можуть просто повторно використовувати значення з попередньої позиції. Отже, для початку у нього є невелика мережа, і для кожного переміщення потрібно обчислити лише її частину. Оновлення лише частини мережі є досить ризикованим, адже можна пропустити важливу інформацію, проте NNUE перераховує на основі того, чи рухалася фігура відносно обох королів, тож це, ймовірно, допомагає їй не пропустити мат. Крім того, багато проблем вирішуються, переглядаючи 100 мільйонів позицій в секунду.

LCZero-CPU набагато слабший порівняно з NNUE. Це тому, що NNUE невелика мережа, яка ефективно оновлюється. NNUE не потрібні графічні процесори, це навпаки сповільнить обчислення, адже додається затримка при передачі інформації від ЦП до ГП.

Stockfish NNUE рідко програє тактично, оскільки має можливість швидко оцінювати величезну кількість позицій. Він також має більше інформації про позицію, що зберігається в своїй мережі, ніж у старій функції оцінки ручної роботи. Однак LCZero (та інші двигуни з DCNN) все ще мають більше позиційної інформації в своїй оцінці, ніж NNUE. Таким чином, ці двигуни іноді можуть вигравати, розпізнаючи тонкі позиційні особливості.

2. СТВОРЕННЯ ВЛАСНОГО ШАХОВОГО ДВИГУНА

2.1. Постановка задачі

Очікуваний результат цієї курсової роботи – робочий шаховий двигун, який буде грати на рівні шахіста початківця. Для тестування цього, а також для зручності подальших досліджень необхідно створити програму з графічним інтерфейсом, до якої можна буде підключити шаховий двигун, щоб людина могла зіграти проти нього. В межах цієї задачі потрібно буде спроектувати архітектуру шахового двигуна, зібрати та обробити дані для навчання нейронної мережі, вибрати архітектуру нейронної мережі та навчити її. Після завершення попередніх задач варто проаналізувати отримані результати навчання та порівняти шаховий двигун з готовим рішенням.

2.2. Оцінка наявних ресурсів та можливостей

Через обмеження в потужностях обчислювальної машини, єдиним варіантом було використання онлайн сервісів для навчання нейронної мережі. Серед проаналізованих сервісів було виділено наступні: Google Colab, Google Cloud і AWS. Можливості AWS не були розглянуті детально, тому що не було достатньо часу, щоб розібратись із незнайомим сервісом.

Google Colab – онлайн сервіс для програмування на мові Python, чудово підходить для спеціалістів напрямку «Data Sience» та «Machine Learning», оскільки за допомогою цього сервісу можна швидко і легко створювати та навчати моделі різних розмірів. І хоча є обмеження на безкоштовне користування сервісом – 6 годин використання GPU (графічного процесора), є варіант платної підписки із збільшенням ресурсів та часу користування [12]. Із переваг також можна виділити мобільність та доступність, адже можна зайти з будь-якого комп’ютера в будь-якому місці.

Google Cloud (Google Compute Engine) – потужний онлайн сервіс, який надає обчислювальні можливості різних розмірів та чудове комерційне

рішення. Сервіс надає демо-період на три місяці і 300$ для використання певного функціоналу, проте щоб використовувати GPU або TPU, потрібно змінити квоту. Це можна зробити наступним чином: а) поповнивши рахунок; б) написавши в службу підтримки [13]. Також величезна кількість функціоналу ускладнює інтерфейс, що робить роботу з цим сервісом не дуже інтуїтивно зрозумілою. Щоб не витрачати час, на освоєння цього потужного інструмента, було вирішено вибрати Google Colab.

2.3. Проектування архітектури шахового двигуна

2.3.1. Комунікація з графічним інтерфейсом

Після аналізу сторонніх двигунів, було вирішено забезпечити підтримку UCI (Universal Chess Interface). Це відкритий протокол для комунікації, який забезпечує комунікацію між шаховим двигуном та графічним інтерфейсом. Особливість цього протоколу полягає в тому, що він надає можливість не зберігати стан гри в шаховому двигуні.

Переваги:

- відсутність інформації про стан ігри на стороні шахового двигуна. Це зменшує кількість проблем синхронізації між шаховим графічним інтерфейсом та двигуном;
- шахова система (об’єднання шахового двигуна та графічного інтерфейсу), працює стабільніше;
- відсутня потреба в зберіганні файлу з налаштуваннями шахового двигуна, адже конфігурація відбувається за допомогою команд протоколу;
- простота для програмістів в розробці та підтримці двигунів на основі UCI протоколу;
- зручність пошуку проблем в роботі двигуна: легко почати гру з певної позиції, легко знайти потрібну позиції в лог-файлах;

- майже усі нові двигуни підтримують цей протокол;
- майже усі шахові графічні інтерфейси підтримують UCI.

UCI використовує довгу алгебраїчну форму запису ходів та «0000» як позначення нульового ходу. Приклад ходів: e2e4, e7e5, e1g1 (коротке рокірування білого короля), e7e8q («коронація» / заміна пішака на ферзя)

Для цього зі сторони графічного інтерфейсу повинен бути реалізований необхідний мінімум для повноцінної гри шахи з шаховим двигуном, а також зі сторони двигуна потрібно реалізувати цей мінімальний інтерфейс.

Цю реалізацію можна знайти в додатку Б. Комунікацію з двигуном, запущеним в терміналі, можна побачити на рис. 2.1.а та 2.1.б.

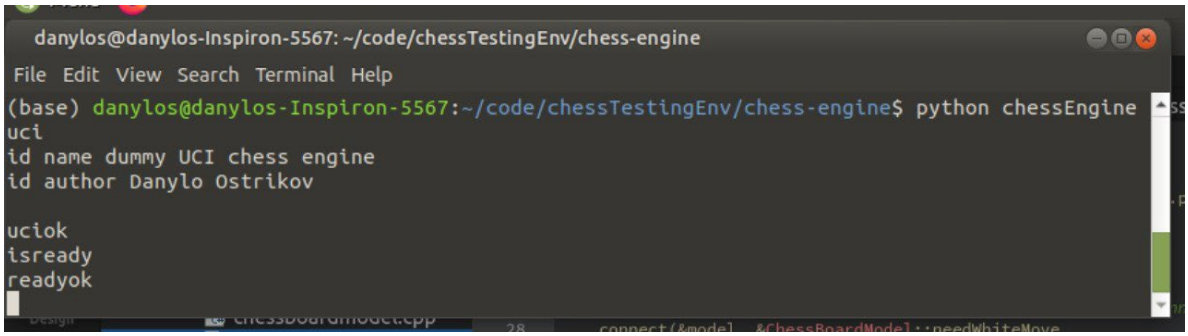

Рисунок 2.1.а) запуск та ініціалізація двигуном через UCI команду ‘uciok’ та ‘isready’

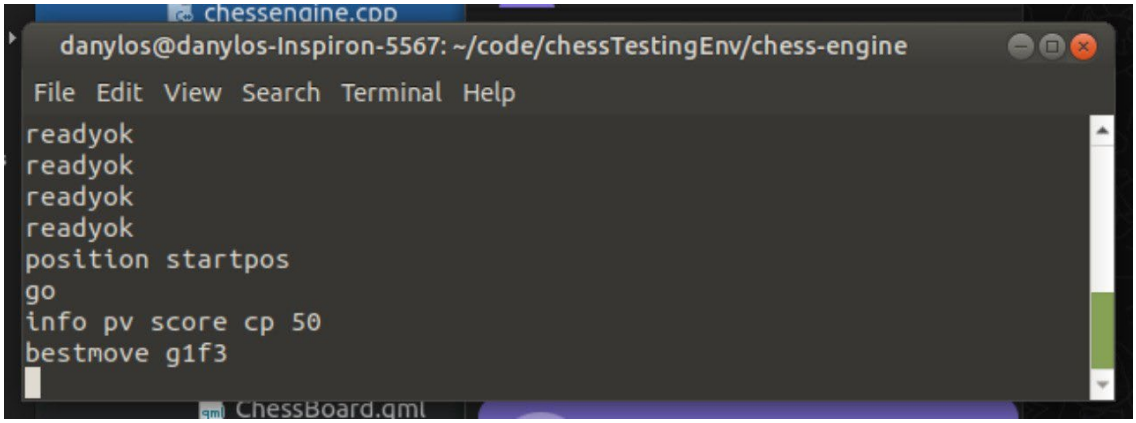

Рисунок 2.1.б) запит на пошук оптимального ходу за допомогою комбінації UCI команд ‘position’ та ‘go’

2.3.2. Пошук оптимального ходу

Створення ефективного алгоритму пошуку оптимального ходу достатньо складна задача, тому в межах цієї курсової роботи було вирішено реалізувати найпростіший алгоритм: коли ми отримуємо на вхід позицію на дошці, ми генеруємо всі легальні ходи, оцінюємо кожну із можливих позицій після даного півкроку та вибираємо найкращу для нас. Цей варіант не передбачає аналіз позицій, що можуть виникнути на дошці, після декількох ходів. Проте, вибір та реалізація алгоритму для пошуку оптимального ходу з врахуванням подальших можливих ходів, було вирішено опрацювати в наступній роботі. В межах цієї курсової роботи, наведеного алгоритму для досягнення поставленої цілі цілком достатньо.

2.3.3. Функція оцінювання позиції

Функція оцінювання позиції – це одна з найважливіших компонент шахового двигуна. В традиційних двигунах, функція оцінки шахових позицій базується на критеріях, визначених експертами та покращеними експерментальним шляхом. Такі двигуни, займали перше місце аж до 2017 року. В 2017 році, двигун Stockfish програв AlphaZero в матчі із 100 партій [14].

Функції оцінки позиції використовують наступні критерії:

- безпека короля,
- матеріальна цінність наявних фігур,
- активність фігур,
- структури пішаків.

Отже, можна виділити наступну задачу: створення нейронної мережі для оцінки шахової позиції, яка враховує безпеку короля, матеріальну цінність фігур, активність фігур та структури пішаків. Зрозуміло, що це не просте завдання і така модель буде: а) архітектурно складною, б) складною для

навчання. Тому в межах цієї роботи, розглянуто спробу створення моделі, яка враховуватиме принаймні матеріальну цінність наявних фігур, а також, можливо, активність фігур.

У цій роботі ключовим є використання нейронної мережі для оцінки позицій, для цього було спроектовано та навчено модель, про що написано в підрозділах 2.5 та 2.6.

2.3.4. Програма з графічним інтерфейсом для гри проти двигуна

Для тестування роботи двигуна, використовуючи засоби Qt Creator та мову програмування C++, була створена програма з графічним інтерфейсом для гри в шахи.

Було вирішено працювати над власним графічним інтерфейсом замість використання існуючих через наступні причини:

- планується в майбутньому створити мобільних додаток на тему шах;
- подальше удосконалення графічного інтерфейсу та перетворення на потужних програмний засіб для аналізу шахових двигунів, проведення порівняння між двигунами та збирання відповідної статистики;
- повторне використання коду програмування шахових правил при перенесені реалізації шахового двигуна з мови програмування Python на мову C++ для покращення швидкодії шахового двигуна.

Створений графічний інтерфейс можна побачити на рис. 2.2.а-в.

Рисунок 2.2.а) стартове вікно програми

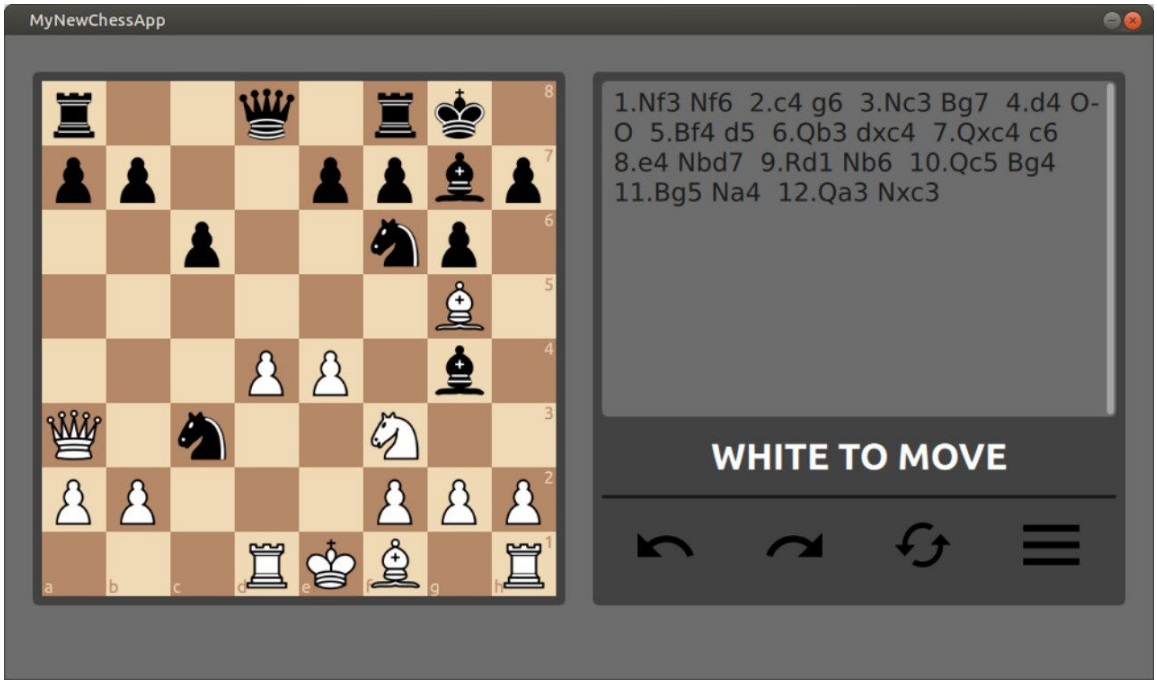

Рисунок 2.2.б) вікно програми посеред гри

Рис. 2.2.в) вікно програми при завершені гри

2.4. Збір та обробка даних

Можна виділити та розглянути наступні сервіси з базами шахових партій:

- YottaBase – сервіс зберігає одну й найбільших у світ базу шаховий партій. Переваги: сервіс безкоштовний; наявність достатньої кількості фільтрів, щоб отримати необхідну базу шахових партій; можливість викачати у форматі PGN. Недоліки: ліміт на безкоштовне використання; партії не проаналізовані шаховими двигунами.
- Chess365 – містить велику кількість партій, а також окремо виділену базу «Masters Database», що містить партії гравців із найвищим рейтингом Ело. Недоліки: необхідний функціонал – платний.
- База партій CCRL – найбільша база шахових партій, зіграних комп’ютерними програмами. Містить статистику шахових позицій, яка складається із відсоткового відношення перемог/нічий/поразок. Містить коментарі до партій, проте формат коментарів не дуже зручний для обробки, тому було вирішено не використовувати цю базу партій.

- База партій Lichess – база зіграних на платформі Lichess партій. Містить 3 367 175 192 партій. Близько 6% партій містять аналіз проведений шаховим двигуном Stockfish [15]. Містить список відомих проблем в базі з партіями. Містить набори партій різних варіацій правил гри в шахи – стандартні шахи, антишахи, атомні шахи, шахи 960, шахи «Crazyhouse», орда, король гори та інші. Набори партій розділені по місяцям, що досить зручно. Головна перевага цієї бази партій – вона цілком безкоштовна.

В результаті аналізу ресурсів для підготовки навчального датасету було обрано платформу Lichess. Використана база налічувала близько 86 мільйонів партій. Так як корисними були лише проаналізовані партії, непотрібні партії було відкинуто. Тривалість кожної з партій в середньому – 40 ходів, тобто в результаті обробки були виділено приблизно по 80 різних позицій з кожної партії. Кожна з позицій зберігалась як окремий запис у форматі FEN у базі даних SQLite для зручності отримання позицій і їх оцінок при навчанні моделі. Кожен запис в базі даних містить наступні поля: ідентифікатор позиції, FEN позиції, бінарне кодування позиції, оцінка позиції. Формат бінарного кодування розглянуто в наступному підрозділі. В результаті, отриманий набір містить 37 164 639 неунікальних позицій.

2.5. Архітектура нейронної мережі

В результаті попереднього аналізу наявної літератури та готових рішень було вирішено передавати в нейронну мережу наступну інформацію:

1) розміщення кожної фігури на дошці;
2) який гравець зараз повинен ходити;
3) можливість сторін зробити рокірування;
4) можливість взяття на проході в даній позиції;
5) розміщення пішака, який в попередньому півході перемістився на дві клітинки, тобто, може бути ціллю для взяття на проході;

6) скільки вже було зроблено ходів;
7) скільки було зроблено ходів відповідно до правила про 50 ходів.

Для кодування позиції фігур було вирішено використати популярну технологію зберігання інформації про шахову позицію – кодування за допомогою бітборд [16]. Бітборд – це масив із нулів та одиниць (рис. 2.3). Бітборд широко використовується в програмуванні шахових двигунів, адже це дає можливість ефективно використовувати кеш процесора, а також прискорює генерацію можливих позицій, так як використовуються різні «магічні» бітові операції [16, 17]. В даному випадку, кожен тип фігур кодується за допомогою масиву із 64 бітів, які репрезентують гральну дошку (рис. 2.4). У верхньому лівому краю розміщена перша координата, у нижньому правому – остання координата. При чому кожна наступна координата розміщена праворуч попередньої, у випадку, якщо координата розміщена на правому краю дошки, то наступною вважається крайня ліва координата, розміщена на один рядок нижче. Тобто, якщо спів ставити наші координати із шаховими координатами у нас вийде наступне: координаті “a8” відповідає бітборд індекс “0”, “b8” => “1”, “h8” => “7”, “a7” => “8” тощо. У результаті, перша частина вхідного масиву є розміром 64x6x2 та має наступну структуру:

- перші 64 біти – розміщення білого короля;
- другі 64 біти – розміщення білих ферзів;
- треті 64 біти – розміщення білих тур;
- четверті 64 біти – розміщення білих офіцерів;
- п’яті 64 біти – розміщення білих коней;
- шості 64 біти – розміщення білих пішаків;
- сьомі 64 біти – розміщення чорних короля;
- восьмі 64 біти – розміщення чорних ферзів;
- дев’яті 64 біти – розміщення чорних тур;
- десяті 64 біти – розміщення чорних офіцерів;
- одинадцяті 64 біти – розміщення чорних коней;

- дванадцяті 64 біти – розміщення чорних пішаків;

У додаткових 5 байтах була закодована інформація про те, скільки ходів вже було зроблено, кількість ходів відповідно до правила про 50 ходів, можливість взяття на проході та можливість рокірування.

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Рис. 2.3. Індексація координат в бітборді

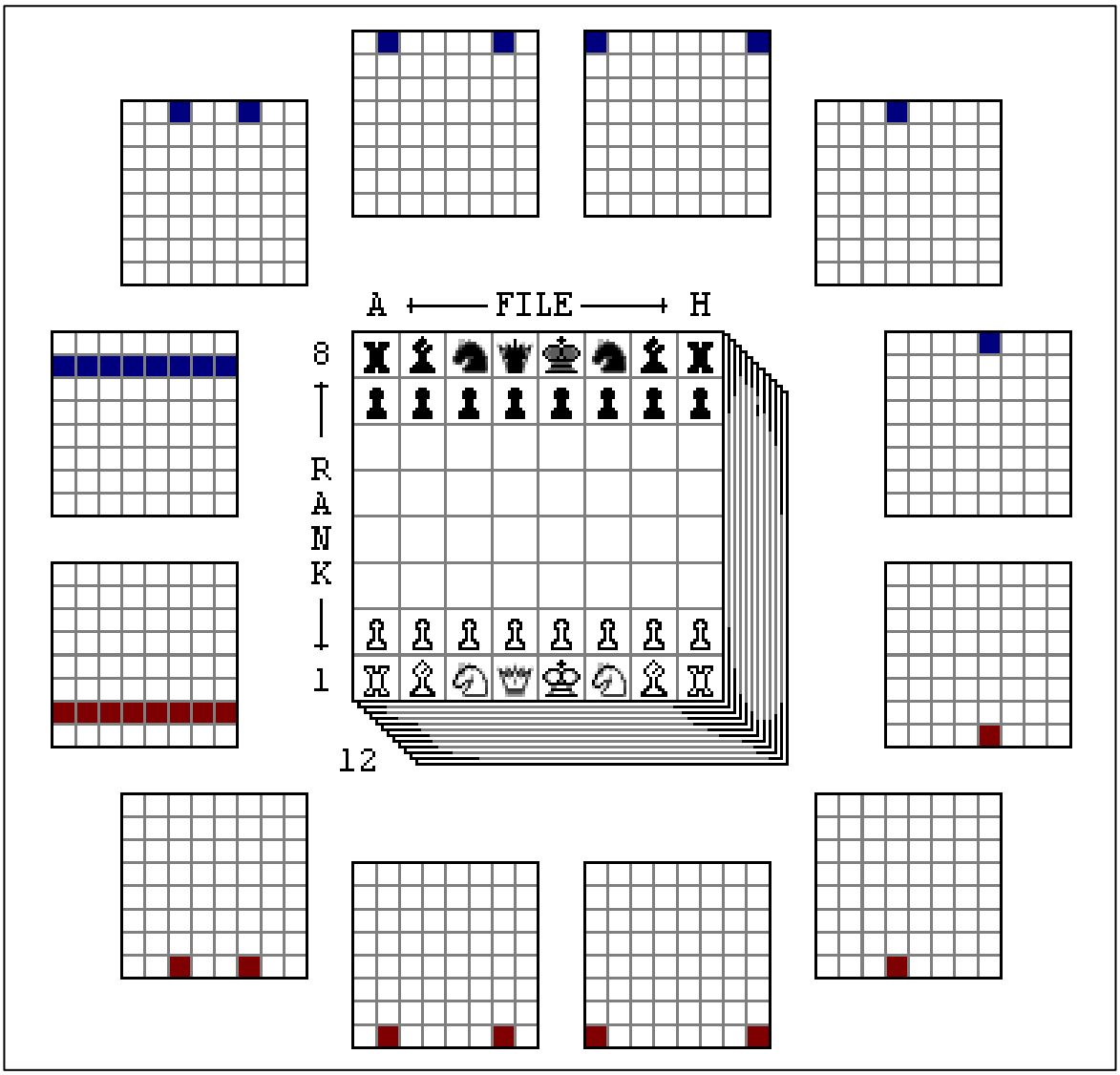

Рис. 2.4. Кодування шахової позиції із використанням бітборд

Створена нейронна мережа (рис. 2.5) яка складається з наступних шарів:

1) вхідний шар: 808 нейронів;
2) 6 прихованих шари по 808 нейронів;
3) вихідний шар: результат – десяткове число в межах приблизно [-15; 15].

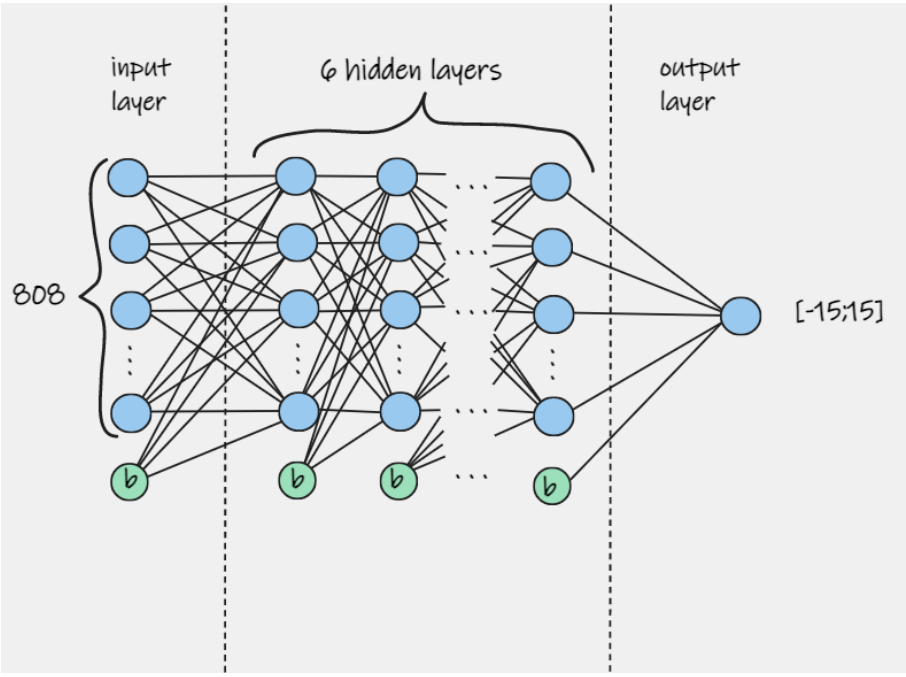

Рис. 2.5. Архітектура нейронної мережі

На виході, ми отримуємо оцінку позиції в межах [-15; 15], де від'ємне значення вказує на домінацію чорного гравця в даній позиції, нульове вказує на нічию, відповідно, додатне значення вказує на наявність переваги у білого гравця. Це число означає перевагу в пішаках. Обмеження оцінки в межах [-15; 15] було необхідним для того, щоб позбавитись від неприємних наслідків від цілком виграшних позицій для одної зі сторін, адже в даному випадку не важливо, щоб отримана модель оцінювала в такий же спосіб як це робить, наприклад, Stockfish. Основна вимога – потрібно, щоб в результаті можна було зробити висновок, яка зі сторін має перевагу. Діапазону [-15; 15] нам для цього достатньо:

- [-15; 10): чорні можуть поставити мат через n ходів і білі цьому не можуть завадити;
- [10; 4): чорні виграють зі значною перевагою;
- [4; -1): чорні виграють з незначною перевагою;
- [-1; 1]: сили гравців майже рівні (на дошці нічийна шахова позиція);
- [1; 4): білі виграють з незначною перевагою;

- [4; 10): білі виграють зі значною перевагою;
- [10; 15]: білі можуть поставити мат через n ходів і чорні цьому не можуть завадити.

В цій моделі передавальною функцією (функцією активації) було обрано зрізаний лінійний вузол (ReLU) через його простоту, біологічну правдоподібність та швидкість обчислення [18]. Інші види функції активації в даній моделі не були використані та досліджені.

Опис структури нейронної мережі та кількість навчальних параметрів можна побачити на рис. 2.6 та 2.7.

```
EvaluationModel(
  (seq): Sequential(
    (linear-0): Linear(in_features=808, out_features=808, bias=True)
    (relu-0): ReLU()
    (linear-1): Linear(in_features=808, out_features=808, bias=True)
    (relu-1): ReLU()
    (linear-2): Linear(in_features=808, out_features=808, bias=True)
    (relu-2): ReLU()
    (linear-3): Linear(in_features=808, out_features=808, bias=True)
    (relu-3): ReLU()
    (linear-4): Linear(in_features=808, out_features=808, bias=True)
    (relu-4): ReLU()
    (linear-5): Linear(in_features=808, out_features=1, bias=True)
  )
)
```

Рис. 2.6. Опис структури нейронної мережі

```
==========================================================================
Layer (type:depth-idx)                   Param #
==========================================================================
EvaluationModel                          --
├─Sequential: 1-1                        --
│    └─Linear: 2-1                       653,672
│    └─ReLU: 2-2                         --
│    └─Linear: 2-3                       653,672
│    └─ReLU: 2-4                         --
│    └─Linear: 2-5                       653,672
│    └─ReLU: 2-6                         --
│    └─Linear: 2-7                       653,672
│    └─ReLU: 2-8                         --
│    └─Linear: 2-9                       653,672
│    └─ReLU: 2-10                        --
│    └─Linear: 2-11                      809
==========================================================================
Total params: 3,269,169
Trainable params: 3,269,169
Non-trainable params: 0
==========================================================================
```

Рис. 2.7. Кількість навчальних параметрів мережі

2.6. Навчання нейронної мережі

Навчання проводилось на платформі Google Colab, де безкоштовно можна використовувати GPU (NVIDIA Tesla K80) [19]. Попри ліміт на безкоштовне використання GPU – шість годин на день, навчання було успішно завершене. Даних ресурсів було достатньо, щоб пройти одну епоху навчання по базі позицій, яка містить всього 37164639 унікальних попередньо проаналізованих двигуном Stockfish позицій.

Для навчання нейронної мережі було використано алгоритм оптимізації функції втрат на основі градієнтного спуску – Adam [20]. Було використано навчання через пакети з даними із 512-ма позиціями в одному пакеті. Навчання проводилось із початковим коефіцієнтом швидкості навчання рівному 1e-3.

Навчання в результаті зайняло близько п’яти годин (рис. 2.8). Отриманий програмний код легко змінити, щоб збільшити кількість кількість прихованих шарів або ж розмір пакету (див. Додаток В), що дає можливість дослідити ефективність навчання мережі з різними початковими параметрами.

```
Using native 16bit precision.
GPU available: True, used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]

  | Name | Type       | Params
-----------------------------------
0 | seq  | Sequential | 2.0 M
-----------------------------------
2.0 M     Trainable params
0         Non-trainable params
2.0 M     Total params
7.847     Total estimated model params size (MB)
/usr/local/lib/python3.7/dist-packages/pytorch_lightning/utilities/data.py:72: UserWarning: Your `IterableDataset`
  "Your `IterableDataset` has `__len__` defined."
Epoch 0: 50%  36040/72588 [2:01:18<2:03:01, 4.95it/s, loss=1.74, v_num=nt-4]
```

Рис. 2.8. Процес навчання моделі

Використана функція втрат – $L(a) = |a|$. Залежність значення похибки від кількості вивчених позицій можна побачити на рис. 2.9.

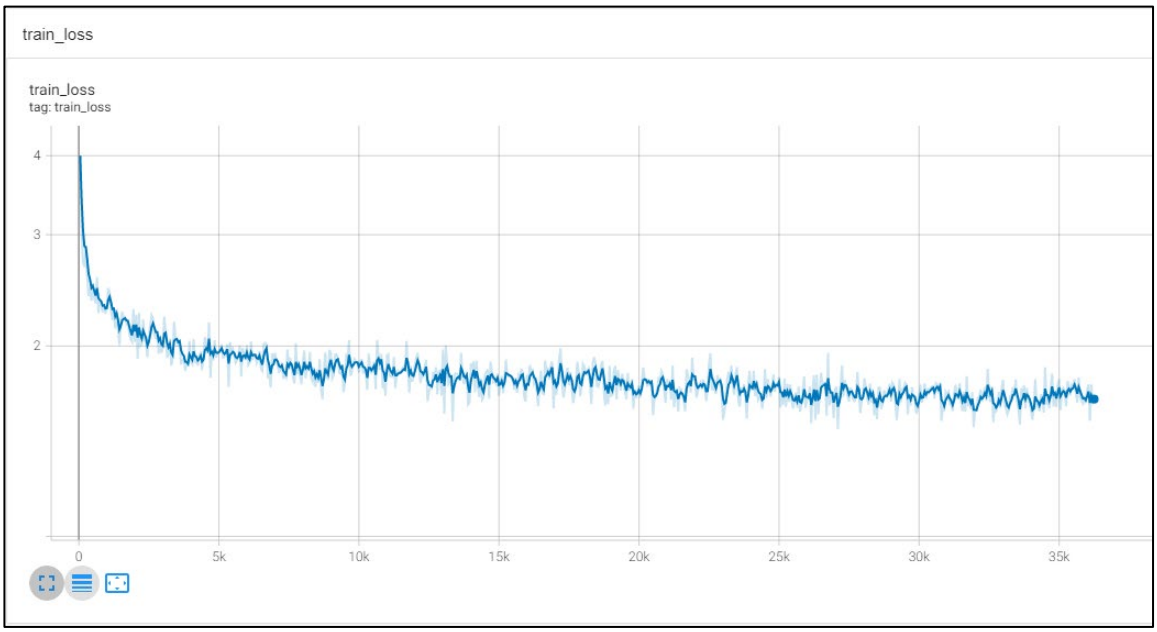

Рис. 2.9. Графік функції втрат

2.7. Аналіз результатів навчання

В цій роботі використано загальноприйняту оцінку вартості фігур [21] (див. таб. 1). Причому ця оцінка не враховує королів при порівнянні кількості матеріалу сторін, адже на дошці постійно є два короля, взяття короля означає завершення гри.

Таблиця 1

Цінність шахових фігур

Фігура	Цінність
Пішак	1
Кінь	3
Слон	3
Тура	5
Ферзь	9

Для порівняння відхилення прогнозованої оцінки позиції від суто матеріальної оцінки позиції було реалізовану функцію *material_for_board()* (рис. 2.10). Середнє відхилення прогнозованої оцінки позиції від матеріальної оцінки позиції становить близько п'яти пішаків. Це не означає, що навчання пройшло невдало і модель погано справляється зі задачею. Із цього результату можна зробити висновок, що модель використовує не лише знання про наявність матеріалу для прогнозування оцінки позиції, а щось іще – наприклад, розміщення фігур.

```
import chess

MATERIAL_LOOKUP = {chess.KING:0,chess.QUEEN:9,
                   chess.ROOK:5,chess.BISHOP:3,
                   chess.KNIGHT:3,chess.PAWN:1}

def material_for_board(board):
  eval = 0.0
  for sq, piece in board.piece_map().items():
    mat = MATERIAL_LOOKUP[piece.piece_type]
    if piece.color == chess.BLACK:
      mat = mat * -1
    eval += mat
  return eval
```

Рис. 2.10. Функція для матеріальної оцінки позиції

Середнє відхилення від навчальної вибірки навченої моделі становить 1,6 пішаків. Враховуючи, що аналіз позиції для навчальної вибірки було проведено шаховим двигуном Stockfish із певною глибиною пошуку, майже неможливо навчити дану модель так, щоб вона узагальнила аналіз можливих наступних ходів, та ще й робила це правильно. Це вже буде не навчання, а перенавчання. Можливо, в даному випадку похибка значна, тому що навчена модель слабка у виявлені тактичних прийомів (із аналізу роботи шахового двигуна LCZero можна вважати, що це нормальне явище).

Із графіка функції втрат на рис. 2.9 видно, що модель можна було продовжувати навчати для покращення результатів, хоча швидкість навчання вже була досить низька.

2.8. Тестування шахового двигуна

Для визначення точного рейтингу шахового двигуна можна скористатись сервісом послугами сервісу Computer Chess Rating Lists (CCRL) [22]. Проте це доволі затратний процес. Для перевірки створеного шахового двигуна було проведено партію проти шахового двигуна Stockfish 14, сконфігурованого на другий рівень складності. Stockfish грав чорними фігурами. У результаті створений шаховий двигун отримав перемогу через те, що Stockfish здався на 151 ході (рис. 2.11). Процес пошуку найкращого ходу створеним шаховим двигуном можна побачити в додатку А.

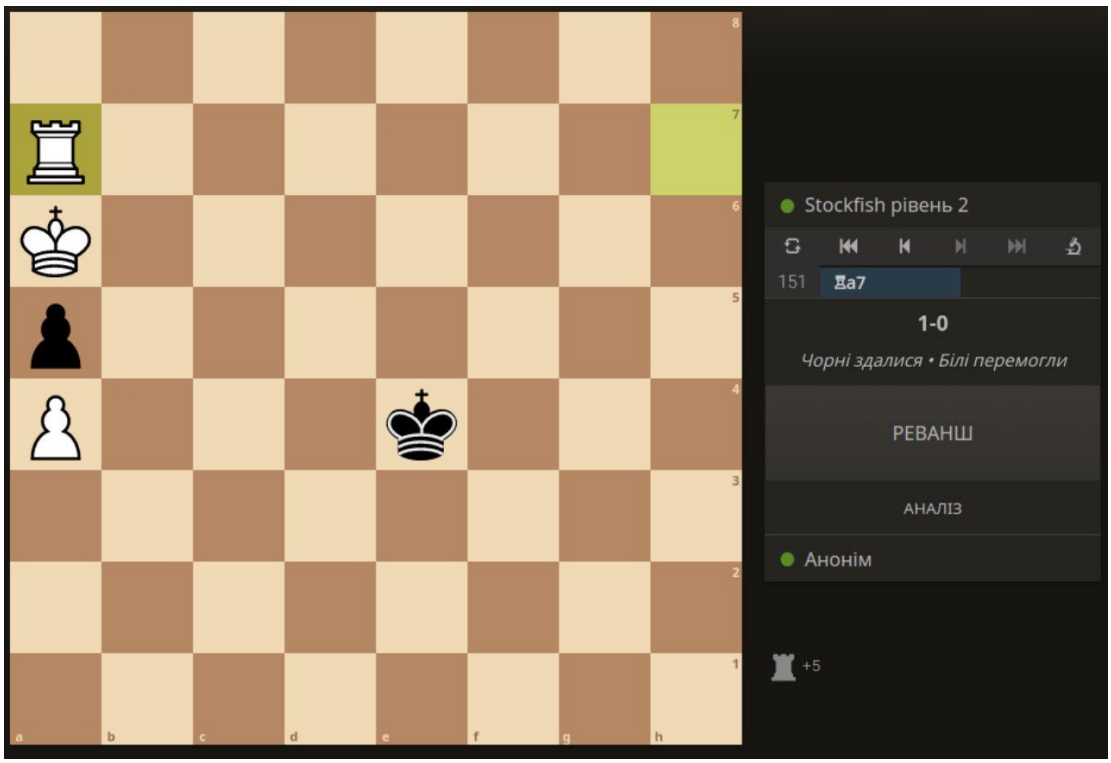

Рис. 2.11. Результат партії створеного шахового двигуна проти Stockfish на 2 рівні складності

При аналізі партії можна побачити, що хоча створений двигун не використовує книжкову базу дебютів, на початку гри (рис. 2.12) він прагне отримати контроль над центром дошки, що є однією із основних стратегій гри [23].

Створений шаховий двигун часто робить однотипні помилки (рис. 2.13, 2.14): він вирішує взяти фігуру противника, не зважаючи на те, що вона є під захистом. Це можна виправити додаванням альфа-бета відсічення [24].

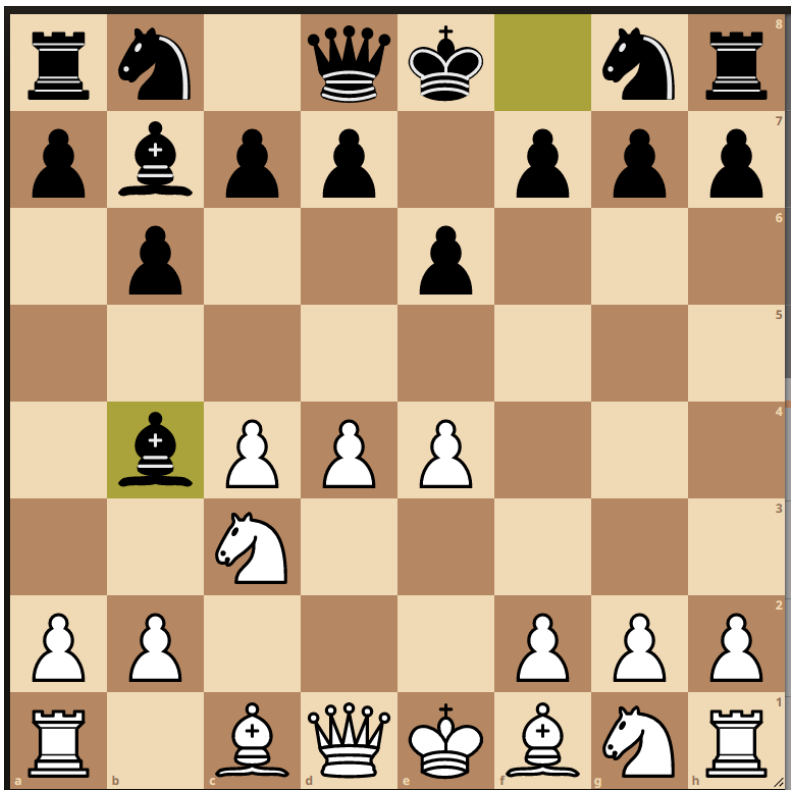

Рис. 2.12. Позиція на дошці після 4 ходу

Рис. 2.13. Білі забрали чорного коня на h4 турою, не зважаючи на те, що кінь був захищений ферзем

Рис. 2.14. Білі забрали чорного пішака на h7, поставивши свою туру під удар

Після аналіз зіграної партії було виявлену критичну слабкість створеного двигуна: приблизно після 12-го ходу, коли у білих повністю виграна позиція, двигун перестав покращувати свою позицію та гра затягнулась до 151-го ходу (рис. 2.15). Причиною цього було введення обмеження на вихідну оцінку позиції в межах [-15;15]. І це зрозуміло, хоча і не дуже очевидно. Адже виграшу позицію можна поступово покращувати, поки не буде поставлено мат опоненту.

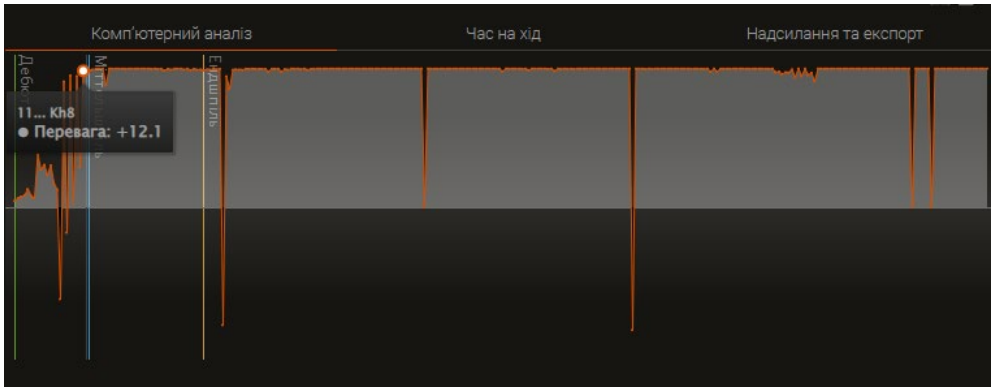

Рис. 2.15. Перевага білого гравця в залежності від ходу

ВИСНОВКИ

В ході курсової роботи був створений простий шаховий двигун та графічний інтерфейс, який в подальшому можна буде покращити, додавши наступний функціонал:

- підключення двох шахових двигунів та запуску гри між ними;
- збір статистики зіграних ігор для подальшого аналізу.

Також можна виділити наступні кроки для оптимізації та покращення роботи шахового двигуна:

- дослідження ефективності навчання нейронної мережі при різній кількості прихованих шарів, різній кількості нейронів в шарах та при різних значеннях гіперпараметрів: швидкості навчання та розміру пакету з даними;
- дослідження ефективності навчання з більш параметризованим вхідним шаром, наприклад, враховуючи кількість полів біля королів, які є під атакою та захистом інших фігур;
- дослідження та оптимізація функції пошуку оптимального ходу, аналіз та застосування різних алгоритмів, таких як альфа-бета відсічення, дерево пошуку Монте-Карло тощо;
- аналіз навчального набору на наявність помилок та детальна попередня обробка навчальних даних, створення власної специфічної навчальної вибірки, яка містить позиції, які відповідають певним критеріям.

Аналіз готових шахових двигунів показав, що задача створення потужного шахового двигуна є актуальною і по нині. Це складна задача, і хоча є велика кількість готових рішень та наукових робіт на цю тему, все ж є невивчені місця. Наприклад, створення гібридного двигуна із використанням NNUE та DCNN.

ВИКОРИСТАНА ЛІТЕРАТУРА

1. Stockfish NNUE [Electronic resource]. Chess Programming Wiki. Available from: https://www.chessprogramming.org/Stockfish_NNUE.
2. Nodchip's Stockfish forked repository [Electronic resourse]. Nodschip's GitHub repository. Available from: https://github.com/nodchip/Stockfish.
3. Introducing NNUE Evaluation [Electronic resourse]. Stockfish – Open Source Chess Engine. Available from: https://stockfishchess.org/blog/2020/introducing-nnue-evaluation.
4. NNUE PyTorch [Electronic resource]. Glinscott's GitHub repository. Available from: https://github.com/glinscott/nnue-pytorch.
5. NNUE Question - King Placements [Electronic resource]. TalkChess. Available from: http://www.talkchess.com/forum3/viewtopic.php?f=7&t=75506&start=9.
6. One year of NNUE [Electronic resource]. Official-stockfish's GitHub repository. Available from: https://github.com/official-stockfish/Stockfish/discussions/3628#discussioncomment-1047323.
7. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm [Electronic resource]. arXiv. Available from: https://arxiv.org/abs/1712.01815.
8. Leela Chess Zero [Electronic resource]. Chess Programming Wiki. Available from: https://www.chessprogramming.org/Leela_Chess_Zero.
9. Announcing lczero [Electronic resource]. TalkChess. Available from: http://www.talkchess.com/forum3/viewtopic.php?t=66280.
10. Lessons From AlphaZero (part 4): Improving the Training Target [Electronic resource]. Vish Abrams's Medium. Available from: https://medium.com/oracledevs/lessons-from-alphazero-part-4-improving-the-training-target-6efba2e71628.
11. Win-Draw-Loss evaluation [Electronic resource]. Lc0 blog. Available from:

https://lczero.org/blog/2020/04/wdl-head.
12. Colaboratory [Electronic resource]. Google. Available from: https://research.google.com/colaboratory/faq.html.
13. Google Compute Engine FAQ [Electronic resource]. Google Cloud. Available from: https://cloud.google.com/compute/docs/faq.
14. Google's AlphaZero Destroys Stockfish In 100-Game Match [Electronic resource]. Chess.com. Available from: https://www.chess.com/news/view/google-s-alphazero-destroys-stockfish-in-100-game-match.
15. Open database [Electronic resource]. Lichess.org. Available from: https://database.lichess.org.
16. Bitboards [Electronic resource]. Chess Programming Wiki. Available from: https://www.chessprogramming.org/Bitboards.
17. Bitboard Board-Definition [Electronic resource]. Chess Programming Wiki. Available from: https://www.chessprogramming.org/Bitboard_Board-Definition.
18. Rectifier [Electronic resource]. Wikipedia, the free encyclopedia. Available from: https://en.wikipedia.org/wiki/Rectifier_(neural_networks).
19. Does Google Colab Use My Gpu? [Electronic resource]. GraphicsCardsAdvisor. Available from: https://graphicscardsadvisor.com/does-google-colab-use-my-gpu/.
20. Adam: A Method for Stochastic Optimization [Electronic resource]. arXiv. Available from: https://arxiv.org/abs/1412.6980.
21. Chess piece relative value [Electronic resource]. Wikipedia, the free encyclopedia. Available from: https://en.wikipedia.org/wiki/Chess_piece_relative_value.
22. Welcome to the CCRL [Electronic resource]. Computer Chess Rating Lists website. Available from: http://www.computerchess.org.uk/ccrl.
23. Як грати в шахи - правила + 7 перших кроків [Електронний ресурс] – Режим доступу: https://www.chess.com/uk/pravila-shakhi.

24. Alpha–beta pruning [Electronic resource]. Wikipedia, the free encyclopedia. Available from: https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning

ДОДАТОК А. ПРОЦЕС ПОШУКУ ХОДІВ

У цьому додатку наведена інформація, збережена у процесі пошуку створеним шаховим двигуном ходів у матчі проти двигуна Stockfish. Лог файл обрізаний з метою економії місця. Після 15 ходу, оцінка позиції зі сторони створеного шахового двигуна нижче 15 практично не опускалась. Вміст лог-файлу:

count of possible moves: 20
g1h3 -> -0.20, g1f3 -> 0.22, b1c3 -> 0.11, b1a3 -> -0.38, h2h3 -> 0.08, g2g3 -> 0.16, f2f3 -> -0.40, e2e3 -> 0.04, d2d3 -> 0.07, c2c3 -> 0.05, b2b3 -> -0.08, a2a3 -> 0.10, h2h4 -> -0.29, g2g4 -> -0.95, f2f4 -> -0.22, e2e4 -> 0.13, d2d4 -> 0.23, c2c4 -> 0.30, b2b4 -> -0.43, a2a4 -> -0.04
best move: c2c4, eval: 0.2954216003417969
r n b q k b n r
p p p p p p p p
. . . .
. . . .
. . P . . . . .
. . . .
P P . P P P P P
R N B Q K B N R
rnbqkbnr/pppppppp/8/8/2P5/8/PP1PPPPP/RNBQKBNR b KQkq c3 0 1
b6
count of possible moves: 22
g1h3 -> 0.29, g1f3 -> 0.58, d1a4 -> 0.05, d1b3 -> 0.12, d1c2 -> 0.25, b1c3 -> 0.60, b1a3 -> -0.07, c4c5 -> -0.55, h2h3 -> 0.49, g2g3 -> 0.57, f2f3 -> 0.01, e2e3 -> 0.36, d2d3 -> 0.37, b2b3 -> 0.34, a2a3 -> 0.50, h2h4 -> 0.17, g2g4 -> -0.16, f2f4 -> 0.02, e2e4 -> 0.33, d2d4 -> 0.57, b2b4 -> 0.16, a2a4 -> 0.20
best move: b1c3, eval: 0.5952143669128418
r n b q k b n r
p . p p p p p p
. p . . . . . .
. . . .
. . P . . . . .
. . N . . . . .
P P . P P P P P
R . B Q K B N R
rnbqkbnr/p1pppppp/1p6/8/2P5/2N5/PP1PPPPP/R1BQKBNR b KQkq - 1 2
e6
count of possible moves: 26
c3d5 -> -0.19, c3b5 -> -0.25, c3e4 -> -0.42, c3a4 -> -0.30, c3b1 -> -0.14, g1h3 -> 0.40, g1f3 -> 0.60, d1a4 -> 0.25, d1b3 -> 0.23, d1c2 -> 0.45, a1b1 -> 0.38, c4c5 -> -0.51, h2h3 -> 0.53, g2g3 -> 0.52, f2f3 -> 0.06, e2e3 -> 0.48, d2d3 -> 0.43, b2b3 -> 0.38, a2a3 -> 0.51, h2h4 -> 0.36, g2g4 -> 0.08, f2f4 -> 0.07, e2e4 -> 0.65, d2d4 -> 0.71, b2b4 -> 0.20, a2a4 -> 0.14
best move: d2d4, eval: 0.7090036273002625
r n b q k b n r
p . p p . p p p

. p . . p . . .
. . . . . .
. . P P . . . .
. . N . . . . .
P P . . P P P P
R . B Q K B N R
rnbqkbnr/p1pp1ppp/1p2p3/8/2PP4/2N5/PP2PPPP/R1BQKBNR b KQkq d3 0 3
Bb7
count of possible moves: 33
c3d5 -> -0.10, c3b5 -> -0.11, c3e4 -> -0.56, c3a4 -> -0.26, c3b1 -> -0.22, g1h3 -> 0.40, g1f3 -> 0.67, e1d2 -> -0.92, d1a4 -> 0.08, d1d3 -> 0.21, d1b3 -> 0.38, d1d2 -> 0.29, d1c2 -> 0.40, c1h6 -> 0.31, c1g5 -> 0.29, c1f4 -> 0.69, c1e3 -> 0.46, c1d2 -> 0.43, a1b1 -> 0.44, d4d5 -> 0.59, c4c5 -> -0.09, h2h3 -> 0.54, g2g3 -> 0.42, f2f3 -> 0.33, e2e3 -> 0.61, b2b3 -> 0.17, a2a3 -> 0.61, h2h4 -> 0.24, g2g4 -> 0.02, f2f4 -> -0.04, e2e4 -> 0.82, b2b4 -> 0.06, a2a4 -> 0.18
best move: e2e4, eval: 0.8225789070129395
r n . q k b n r
p b p p . p p p
. p . . p . . .
. . . . . .
. . P P P . . .
. . N . . . . .
P P . . . P P P
R . B Q K B N R
rn1qkbnr/pbpp1ppp/1p2p3/8/2PPP3/2N5/PP3PPP/R1BQKBNR b KQkq e3 0 4
Ne7
count of possible moves: 40
c3d5 -> 0.38, c3b5 -> 0.43, c3a4 -> 0.30, c3e2 -> 0.50, c3b1 -> 0.13, g1h3 -> 0.92, g1f3 -> 1.47, g1e2 -> 1.47, f1d3 -> 1.50, f1e2 -> 0.94, e1e2 -> 0.02, e1d2 -> -0.37, d1h5 -> 0.52, d1g4 -> 0.67, d1a4 -> 0.55, d1f3 -> 0.72, d1d3 -> 0.83, d1b3 -> 0.80, d1e2 -> 0.82, d1d2 -> 0.89, d1c2 -> 0.93, c1h6 -> 1.33, c1g5 -> 0.96, c1f4 -> 1.15, c1e3 -> 1.08, c1d2 -> 0.93, a1b1 -> 1.07, e4e5 -> 0.20, d4d5 -> 1.20, c4c5 -> 0.70, h2h3 -> 1.00, g2g3 -> 0.95, f2f3 -> 0.81, b2b3 -> 0.64, a2a3 -> 0.98, h2h4 -> 1.04, g2g4 -> 0.65, f2f4 -> 0.63, b2b4 -> 0.46, a2a4 -> 0.76
best move: f1d3, eval: 1.502136468887329
r n . q k b . r
p b p p n p p p
. p . . p . . .
. . . . . .
. . P P P . . .
. . N B . . . .
P P . . . P P P
R . B Q K . N R
rn1qkb1r/pbppnppp/1p2p3/8/2PPP3/2NB4/PP3PPP/R1BQK1NR b KQkq - 2 5
Ng6
count of possible moves: 42
d3e2 -> 0.63, d3c2 -> 0.98, d3f1 -> 0.52, d3b1 -> 0.86, c3d5 -> 0.71, c3b5 -> 0.13, c3a4 -> 0.45, c3e2 -> 0.74, c3b1 -> 0.07, g1h3 -> 0.99, g1f3 -> 1.51, g1e2 -> 1.59, e1e2 -> 0.10, e1d2 -> -0.43, e1f1 -> 0.19, d1h5 -> 0.86, d1g4 -> 0.63, d1a4 -> 0.36, d1f3 -> 0.62, d1b3 -> 0.71, d1e2 -> 0.87, d1d2 -> 0.90, d1c2 -> 0.92, c1h6 -> 0.97, c1g5 -> 0.86, c1f4 -> 0.84, c1e3 -> 1.39, c1d2 -> 1.01, a1b1 -> 1.07, e4e5 -> 0.14, d4d5 -> 1.01, c4c5 -> 0.37, h2h3 -> 0.81, g2g3 -> 0.84, f2f3 -> 0.59, b2b3 -> 0.50, a2a3 -> 0.96, h2h4 -> 1.03, g2g4 -> 0.14, f2f4 -> 1.07, b2b4 -> 0.33, a2a4 -> 0.57
best move: g1e2, eval: 1.589677333831787

r n . q k b . r
p b p p . p p p
. p . . p . n .
. . . . . .
. . P P P . . .
. . N B . . . .
P P . . N P P P
R . B Q K . . R
rn1qkb1r/pbpp1ppp/1p2p1n1/8/2PPP3/2NB4/PP2NPPP/R1BQK2R b KQkq - 4 6
Bb4
count of possible moves: 32
d3c2 -> 1.06, d3b1 -> 1.15, e2f4 -> 0.82, e2g3 -> 0.98, e2g1 -> 0.43, h1g1 -> 0.74, h1f1 -> 0.98, e1d2 -> -0.15, e1f1 -> 0.49, d1a4 -> 0.79, d1b3 -> 1.00, d1d2 -> 1.11, d1c2 -> 1.13, c1h6 -> 1.23, c1g5 -> 0.90, c1f4 -> 1.07, c1e3 -> 1.45, c1d2 -> 1.23, a1b1 -> 1.10, e1g1 -> 1.56, e4e5 -> 0.77, d4d5 -> 0.74, c4c5 -> 0.72, h2h3 -> 1.04, g2g3 -> 0.87, f2f3 -> 1.19, b2b3 -> 0.53, a2a3 -> 1.37, h2h4 -> 1.61, g2g4 -> 0.38, f2f4 -> 1.46, a2a4 -> 0.86
best move: h2h4, eval: 1.6120541095733643
r n . q k . . r
p b p p . p p p
. p . . p . n .
. . . . . .
. b P P P . . P
. . N B . . . .
P P . . N P P .
R . B Q K . . R
rn1qk2r/pbpp1ppp/1p2p1n1/8/1bPPP2P/2NB4/PP2NPP1/R1BQK2R b KQkq h3 0 7
Nxh4
count of possible moves: 33
d3c2 -> 0.18, d3b1 -> 0.38, e2f4 -> 0.38, e2g3 -> 0.38, e2g1 -> -0.29, h1h4 -> 5.77, h1h3 -> -0.12, h1h2 -> -0.27, h1g1 -> -0.15, h1f1 -> -0.14, e1d2 -> -1.22, e1f1 -> 0.15, d1a4 -> 0.53, d1b3 -> 0.28, d1d2 -> 0.70, d1c2 -> 0.57, c1h6 -> 0.96, c1g5 -> 0.47, c1f4 -> 0.67, c1e3 -> 0.85, c1d2 -> 0.78, a1b1 -> 1.11, e1g1 -> 0.56, e4e5 -> 0.47, d4d5 -> 0.14, c4c5 -> 0.31, g2g3 -> 0.06, f2f3 -> 0.74, b2b3 -> -0.21, a2a3 -> 0.92, g2g4 -> -0.08, f2f4 -> 0.75, a2a4 -> 0.14
best move: h1h4, eval: 5.768982887268066
r n . q k . . r
p b p p . p p p
. p . . p . . .
. . . . . .
. b P P P . . R
. . N B . . . .
P P . . N P P .
R . B Q K . . .
rn1qk2r/pbpp1ppp/1p2p3/8/1bPPP2R/2NB4/PP2NPP1/R1BQK3 b Qkq - 0 8
0-0
count of possible moves: 35
h4h7 -> 7.02, h4h6 -> 5.50, h4h5 -> 6.32, h4g4 -> 6.34, h4f4 -> 6.08, h4h3 -> 6.76, h4h2 -> 5.95, h4h1 -> 5.83, d3c2 -> 6.50, d3b1 -> 6.60, e2f4 -> 6.90, e2g3 -> 6.30, e2g1 -> 5.66, e1d2 -> 5.44, e1f1 -> 6.53, d1a4 -> 5.66, d1b3 -> 5.87, d1d2 -> 6.54, d1c2 -> 6.63, c1h6 -> 6.68, c1g5 -> 6.53, c1f4 -> 6.68, c1e3 -> 6.63, c1d2 -> 6.55, a1b1 -> 6.43, e4e5 -> 7.00, d4d5 -> 5.45, c4c5 -> 5.96, g2g3 -> 5.92, f2f3 -> 5.95, b2b3 -> 5.56, a2a3 -> 6.30, g2g4 -> 5.64, f2f4 -> 6.04, a2a4 -> 5.91
best move: h4h7, eval: 7.02123498916626

r n . q . r k .
p b p p . p p R
. p . . p . . .
. . . . . .
. b P P P . . .
. . N B . . . .
P P . . N P P .
R . B Q K . . .
rn1q1rk1/pbpp1ppR/1p2p3/8/1bPPP3/2NB4/PP2NPP1/R1BQK3 b Q - 0 9
Bxc3+
count of possible moves: 5
e1f1 -> 4.22, e2c3 -> 7.81, d1d2 -> 4.69, c1d2 -> 3.81, b2c3 -> 7.85
best move: b2c3, eval: 7.851337909698486
r n . q . r k .
p b p p . p p R
. p . . p . . .
. . . . . .
. . P P P . . .
. . P B . . . .
P . . . N P P .
R . B Q K . . .
rn1q1rk1/pbpp1ppR/1p2p3/8/2PPP3/2PB4/P3NPP1/R1BQK3 b Q - 0 10
Nc6
count of possible moves: 36
h7h8 -> 9.18, h7g7 -> 12.69, h7h6 -> 7.70, h7h5 -> 8.52, h7h4 -> 8.77, h7h3 -> 9.13, h7h2 -> 8.35, h7h1 -> 7.83, d3c2 -> 8.62, d3b1 -> 8.80, e2f4 -> 9.78, e2g3 -> 8.91, e2g1 -> 8.13, e1d2 -> 7.74, e1f1 -> 8.19, d1a4 -> 6.33, d1b3 -> 7.35, d1d2 -> 8.28, d1c2 -> 8.22, c1h6 -> 7.96, c1g5 -> 8.71, c1f4 -> 8.38, c1e3 -> 8.54, c1a3 -> 8.01, c1d2 -> 7.61, c1b2 -> 7.86, a1b1 -> 8.57, e4e5 -> 8.90, d4d5 -> 6.83, c4c5 -> 7.24, g2g3 -> 7.02, f2f3 -> 6.84, a2a3 -> 6.93, g2g4 -> 7.08, f2f4 -> 7.29, a2a4 -> 7.54
best move: h7g7, eval: 12.692252159118652
r . . q . r k .
p b p p . p R .
. p n . p . . .
. . . . . .
. . P P P . . .
. . P B . . . .
P . . . N P P .
R . B Q K . . .
r2q1rk1/pbpp1pR1/1pn1p3/8/2PPP3/2PB4/P3NPP1/R1BQK3 b Q - 0 11
Kh8
count of possible moves: 35
g7g8 -> 11.94, g7h7 -> 12.16, g7f7 -> 9.98, g7g6 -> 11.25, g7g5 -> 11.12, g7g4 -> 12.90, g7g3 -> 12.24, d3c2 -> 11.60, d3b1 -> 11.93, e2f4 -> 14.97, e2g3 -> 14.34, e2g1 -> 12.49, e1d2 -> 11.00, e1f1 -> 11.38, d1a4 -> 9.38, d1b3 -> 10.69, d1d2 -> 12.41, d1c2 -> 11.62, c1h6 -> 11.53, c1g5 -> 14.26, c1f4 -> 13.09, c1e3 -> 12.40, c1a3 -> 11.57, c1d2 -> 11.03, c1b2 -> 10.85, a1b1 -> 11.40, e4e5 -> 14.42, d4d5 -> 10.30, c4c5 -> 10.76, g2g3 -> 9.91, f2f3 -> 9.90, a2a3 -> 10.60, g2g4 -> 9.08, f2f4 -> 9.35, a2a4 -> 10.04
best move: e2f4, eval: 14.972040176391602
r . . q . r . k
p b p p . p R .
. p n . p . . .

. . . . . .
. . P P P N . .
. . P B . . . .
P . . . . P P .
R . B Q K . . .
r2q1r1k/pbpp1pR1/1pn1p3/8/2PPPN2/2PB4/P4PP1/R1BQK3 b Q - 2 12
Ne5
count of possible moves: 41
g7g8 -> 14.46, g7h7 -> 14.26, g7f7 -> 10.96, g7g6 -> 14.29, g7g5 -> 14.17, g7g4 -> 14.31, g7g3 -> 14.46, f4g6 -> 14.17, f4e6 -> 10.10, f4h5 -> 14.68, f4d5 -> 11.19, f4h3 -> 11.83, f4e2 -> 10.74, d3e2 -> 11.46, d3c2 -> 13.26, d3f1 -> 11.39, d3b1 -> 14.05, e1e2 -> 13.60, e1d2 -> 14.15, e1f1 -> 14.51, d1h5 -> 14.65, d1g4 -> 14.66, d1a4 -> 10.42, d1f3 -> 14.76, d1b3 -> 11.08, d1e2 -> 14.21, d1d2 -> 13.20, d1c2 -> 12.64, c1e3 -> 14.35, c1a3 -> 14.06, c1d2 -> 13.15, c1b2 -> 14.62, a1b1 -> 14.17, d4e5 -> 14.98, d4d5 -> 7.93, c4c5 -> 13.26, g2g3 -> 10.88, f2f3 -> 10.16, a2a3 -> 11.33, g2g4 -> 9.39, a2a4 -> 11.13
best move: d4e5, eval: 14.982900619506836
r . . q . r . k
p b p p . p R .
. p . . p . . .
. . . . P . . .
. . P . P N . .
. . P B . . . .
P . . . . P P .
R . B Q K . . .
r2q1r1k/pbpp1pR1/1p2p3/4P3/2P1PN2/2PB4/P4PP1/R1BQK3 b Q - 0 13
Bd5
count of possible moves: 41
g7g8 -> 14.98, g7h7 -> 15.05, g7f7 -> 15.02, g7g6 -> 15.06, g7g5 -> 15.11, g7g4 -> 15.12, g7g3 -> 15.15, f4g6 -> 14.97, f4e6 -> 14.96, f4h5 -> 15.05, f4d5 -> 14.92, f4h3 -> 15.01, f4e2 -> 15.02, d3e2 -> 15.03, d3c2 -> 15.04, d3f1 -> 14.99, d3b1 -> 15.03, e1e2 -> 15.00, e1d2 -> 15.02, e1f1 -> 14.98, d1h5 -> 14.94, d1g4 -> 15.02, d1a4 -> 14.62, d1f3 -> 15.05, d1b3 -> 15.04, d1e2 -> 15.06, d1d2 -> 15.07, d1c2 -> 15.02, c1e3 -> 15.02, c1a3 -> 15.03, c1d2 -> 15.03, c1b2 -> 14.97, a1b1 -> 15.01, e4d5 -> 14.93, c4d5 -> 14.94, c4c5 -> 15.06, g2g3 -> 15.01, f2f3 -> 15.00, a2a3 -> 15.04, g2g4 -> 14.70, a2a4 -> 15.10
best move: g7g3, eval: 15.147058486938477
r . . q . r . k
p . p p . p . .
. p . . p . . .
. . . b P . . .
. . P . P N . .
. . P B . . R .
P . . . . P P .
R . B Q K . . .
r2q1r1k/p1pp1p2/1p2p3/3bP3/2P1PN2/2PB2R1/P4PP1/R1BQK3 b Q - 2 14

ДОДАТОК Б. КОД ПРОГРАМИ

Код створення, навчання та збереження шахової мережі:

```
!wget https://storage.googleapis.com/chesspic/datasets/2022-03-31-lichess-evaluations-37MM.db.gz
!gzip -f -d "2022-03-31-lichess-evaluations-37MM.db.gz"
!rm "2022-03-31-lichess-evaluations-37MM.db.gz"

!pip install peewee pytorch-lightning

from peewee import *
import base64

db = SqliteDatabase('2022-03-31-lichess-evaluations-37MM.db')

class Evaluations(Model):
  id = IntegerField()
  fen = TextField()
  binary = BlobField()
  eval = FloatField()

  class Meta:
    database = db

  def binary_base64(self):
    return base64.b64encode(self.binary)
db.connect()
LABEL_COUNT = 37164639
# LABEL_COUNT = 1000000
print(LABEL_COUNT)
eval = Evaluations.get(Evaluations.id == 1)
print(eval.binary_base64())

import os
import torch
import numpy as np
from torch import nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader, IterableDataset, random_split
import pytorch_lightning as pl
from random import randrange
import time
from collections import OrderedDict

class EvaluationDataset(IterableDataset):
  def __init__(self, count):
    self.count = count
  def __iter__(self):
    return self
  def __next__(self):
    idx = randrange(self.count)
    return self[idx]
  def __len__(self):
    return self.count
  def __getitem__(self, idx):
    eval = Evaluations.get(Evaluations.id == idx+1)
    bin = np.frombuffer(eval.binary, dtype=np.uint8)
    bin = np.unpackbits(bin, axis=0).astype(np.single)
    eval.eval = max(eval.eval, -15)
    eval.eval = min(eval.eval, 15)
```

```
        ev = np.array([eval.eval]).astype(np.single)
        return {'binary':bin, 'eval':ev}

    dataset = EvaluationDataset(count=LABEL_COUNT)

    # Start tensorboard.
    %reload_ext tensorboard
    %tensorboard --logdir lightning_logs/

    import time
    from collections import OrderedDict

    class EvaluationModel(pl.LightningModule):
      def __init__(self,learning_rate=1e-3,batch_size=1024,layer_count=10):
        super().__init__()
        self.batch_size = batch_size
        self.learning_rate = learning_rate
        layers = []
        for i in range(layer_count-1):
          layers.append((f"linear-{i}", nn.Linear(808, 808)))
          layers.append((f"relu-{i}", nn.ReLU()))
        layers.append((f"linear-{layer_count-1}", nn.Linear(808, 1)))
        self.seq = nn.Sequential(OrderedDict(layers))

      def forward(self, x):
        return self.seq(x)

      def training_step(self, batch, batch_idx):
        x, y = batch['binary'], batch['eval']
        y_hat = self(x)
        loss = F.l1_loss(y_hat, y)
        self.log("train_loss", loss)
        return loss

      def configure_optimizers(self):
        return torch.optim.Adam(self.parameters(), lr=self.learning_rate)

      def train_dataloader(self):
        dataset = EvaluationDataset(count=LABEL_COUNT)
        return DataLoader(dataset, batch_size=self.batch_size, num_workers=2, pin_memory=True)

    configs = [
            #  {"layer_count": 4, "batch_size": 64},
            #  {"layer_count": 4, "batch_size": 512},
             {"layer_count": 6, "batch_size": 1024},
             ]
    for config in configs:
      version_name = f'{int(time.time())}-batch_size-{config["batch_size"]}-layer_count-{config["layer_count"]}'
      logger = pl.loggers.TensorBoardLogger("lightning_logs", name="chessml", version=version_name)
      trainer = pl.Trainer(gpus=1,precision=16,max_epochs=1,auto_lr_find=True,logger=logger)
      model =
EvaluationModel(layer_count=config["layer_count"],batch_size=config["batch_size"],learning_rate=1e-3)
      # trainer.tune(model)
      # lr_finder = trainer.tuner.lr_find(model, min_lr=1e-6, max_lr=1e-3, num_training=25)
      # fig = lr_finder.plot(suggest=True)
      # fig.show()
      trainer.fit(model)
      break

    from torchvision import models
    from torchsummary import summary

    model = torch.load("chessAiModel")
```

```
print(model)

from IPython.display import display, SVG
from random import randrange

SVG_BASE_URL = "https://us-central1-spearsx.cloudfunctions.net/chesspic-fen-image/"

def svg_url(fen):
  fen_board = fen.split()[0]
  return SVG_BASE_URL + fen_board

def show_index(idx):
  eval = Evaluations.select().where(Evaluations.id == idx+1).get()
  batch = dataset[idx]
  x, y = torch.tensor(batch['binary']), torch.tensor(batch['eval'])
  y_hat = model(x)
  loss = F.l1_loss(y_hat, y)
  print(f'Idx {idx} Eval {y.data[0]:.2f} Prediction {y_hat.data[0]:.2f} Loss {loss:.2f}')
  print(f'FEN {eval.fen}')
  display(SVG(url=svg_url(eval.fen)))

for i in range(5):
  idx = randrange(LABEL_COUNT)
  show_index(idx)

# need to do better on "tactics" like 700756

idx = 196153
for i in range (31):
  show_index(idx + i)

import chess

MATERIAL_LOOKUP =
{chess.KING:0,chess.QUEEN:9,chess.ROOK:5,chess.BISHOP:3,chess.KNIGHT:3,chess.PAWN:1}

def avg(lst):
    return sum(lst) / len(lst)

def material_for_board(board):
  eval = 0.0
  for sq, piece in board.piece_map().items():
    mat = MATERIAL_LOOKUP[piece.piece_type]
    if piece.color == chess.BLACK:
      mat = mat * -1
    eval += mat
  return eval

def guess_zero_loss(idx):
  eval = Evaluations.select().where(Evaluations.id == idx+1).get()
  y = torch.tensor(eval.eval)
  y_hat = torch.zeros_like(y)
  loss = F.l1_loss(y_hat, y)
  return loss

def guess_material_loss(idx):
  eval = Evaluations.select().where(Evaluations.id == idx+1).get()
  board = chess.Board(eval.fen)
  y = torch.tensor(eval.eval)
  y_hat = torch.tensor(material_for_board(board))
  loss = F.l1_loss(y_hat, y)
  return loss
```

```
def guess_model_loss(idx):
  eval = Evaluations.select().where(Evaluations.id == idx+1).get()
  batch = dataset[idx]
  x, y = torch.tensor(batch['binary']), torch.tensor(batch['eval'])
  y_hat = model(x)
  loss = F.l1_loss(y_hat, y)
  return loss

zero_losses = []
mat_losses = []
model_losses = []
for i in range(1000):
  idx = randrange(LABEL_COUNT)
  zero_losses.append(guess_zero_loss(idx))
  mat_losses.append(guess_material_loss(idx))
  model_losses.append(guess_model_loss(idx))
print(f'Guess Zero Avg Loss {avg(zero_losses)}')
print(f'Guess Material Avg Loss {avg(mat_losses)}')
print(f'Guess Model Avg Loss {avg(model_losses)}')

torch.save(model, "chessAiModel1")
torch.save(model.state_dict(), "chessAiModel_state1")
```

Код файлу “chessengine.cpp”, який реалізує комунікацію із шаховим двигуном по протоколу UCI:

```
#include "chessengine.h"

#include <iostream>
#include <QThread>
#include <QTimer>

ChessEngine::ChessEngine(QObject *parent, ChessBoardModel &model, QString path)
   : QObject(parent)
   , mModel(model)
   , mProcess(parent)
   , mState(ChessEngineState::Uninitialized)
{
   mProcess.setProgram(path);
   mProcess.setProgram("./chessEngineStockfish");
   mProcess.setProgram("/home/danylos/code/chessTestingEnv/chess-engine/engine.py");
   mProcess.start();

   if (!mProcess.waitForStarted())
   {
      qDebug() << "Failed to start chess engine" << mProcess.program() << "error:" << mProcess.errorString();
      assert(false);
   }

   connect(&model, &ChessBoardModel::madeMove,
         this, &ChessEngine::testLoggingMoves, Qt::UniqueConnection);
   connect(&model, &ChessBoardModel::needWhiteMove,
         this, &ChessEngine::testRequestForWhiteMove, Qt::UniqueConnection);
   connect(&model, &ChessBoardModel::needBlackMove,
         this, &ChessEngine::testRequestForBlackMove, Qt::UniqueConnection);
   connect(&mProcess, &QProcess::readyReadStandardOutput,
         this, &ChessEngine::handleOutput, Qt::UniqueConnection);
   connect(&mProcess, &QProcess::readyReadStandardError,
         this, &ChessEngine::handleErrorOutput, Qt::UniqueConnection);
```

```
    mProcess.write("uci\n");
    mProcess.write("isready\n");
    mProcess.waitForBytesWritten();
}

void ChessEngine::testRequestForWhiteMove()
{
}

void ChessEngine::testRequestForBlackMove()
{
    QString positionComand = "position startpos";
    if (movesHistory.isEmpty() == false)
    {
        positionComand += " moves ";
        positionComand += movesHistory.join(' ');
    }

    positionComand += "\n";
    qDebug() << "to engine:" << positionComand;
    qDebug() << mProcess.write(positionComand.toUtf8());
    mState = ChessEngineState::LookingForMove;
    qDebug() << "changed state to LookingForMove";
    qDebug() << mProcess.write("go movetime 1000\n");
}

void ChessEngine::testLoggingMoves(std::string move)
{
    movesHistory.push_back(move.c_str());
    qDebug() << "Cathed move:" << move.c_str() << "all moves:" << movesHistory;
}

void ChessEngine::handleOutput()
{
    QString out = mProcess.readAllStandardOutput();
    switch (mState)
    {
    case ChessEngineState::Uninitialized:
    {
        if (!out.contains("uciok"))
        {
            qDebug() << "didn't receive any uciok here";
            return;
        }
        mState = ChessEngineState::Dormant;
        qDebug() << "changed state to Dormant";
        break;
    }
    case ChessEngineState::LookingForMove:
    {
        if (!out.contains("bestmove"))
        {
            qDebug() << "didn't receive any bestmoves here";
            return;
        }
        QStringList lines = out.split('\n');
        for (const auto &line : lines)
        {
            if (line.contains("bestmove"))
            {
                mState = ChessEngineState::Dormant;
                QString move = line.split(' ').at(1);
                qDebug() << "found a best move:" << move;
```

```
            if (mModel.makeMove(move.toStdString()) == false)
            {
                qCritical() << "Illegal move from engine";
                testRequestForBlackMove();
            }
        }
    }
    break;
  }
  default:
  {
    qDebug() << "Cannot handle output while in this state ->" << mState;
  }
  }
}

void ChessEngine::handleErrorOutput()
{
  QString out = mProcess.readAllStandardError();
  qCritical() << out;
}
```

Код файлу chessboard.cpp, який містить внутрішню репрезентацію дошки та перевірку ходів відповідно до правил:

```
#include "doctest/doctest.h"
#include "chessboard.h"
#include "piecemove.h"
#include "utils.h"
#include <QDebug>
#include <iostream>
#include <string_view>

static const int INVALID_COORDINATES = BOARD_LAST_INDEX + 1;

ChessMove::ChessMove(const std::string &longAlgebraicNotation)
{
  std::string from(longAlgebraicNotation.substr(0, 2));
  std::string to(longAlgebraicNotation.substr(2, 4));
  mFrom = Coordinates(from);
  mTo = Coordinates(to);
}

Chess::Coordinates ChessMove::getFrom() const
{
  return mFrom;
}

Chess::Coordinates ChessMove::getTo() const
{
  return mTo;
}

enum PieceMoveDirection
{
    Right = 1,
    Left = -Right,
    Down = 8,
    Up = -Down,
    DownRight = Down + Right,
```

```
        UpLeft = -DownRight,
        DownLeft = Down + Left,
        UpRight = -DownLeft,
};

bool isPawnOnStartPosition(Chess::Coordinates coordinates, Chess::PieceColor pieceColor);
bool isKingOnStartPosition(Chess::Coordinates coordinates, Chess::PieceColor pieceColor);

ChessPiece::ChessPiece(Chess::PieceColor color
          , Chess::PieceType type
          , Chess::Coordinates coordinates)
    : mColor{color}
    , mType{type}
    , mCoordinates{coordinates}
{}

Chess::PieceColor ChessPiece::getColor() const
{
    return mColor;
}

Chess::PieceType ChessPiece::getType() const
{
    return mType;
}

Chess::Coordinates ChessPiece::getCoordinates() const
{
    return mCoordinates;
}

void ChessPiece::setCoordinates(const Chess::Coordinates &coordinates)
{
    mCoordinates = coordinates;
}

ChessBoard::ChessBoard(ChessBoard::Pieces Pieces)
    : mPieces(Pieces), mEnPassantCoordinate(INVALID_COORDINATES)
{
    updateBoard();
}

size_t ChessBoard::getPiecesCount() const
{
    return mPieces.size();
}

const ChessPiece& ChessBoard::getPieceByIndex(size_t pieceIndex) const
{
    return mPieces.at(pieceIndex);
}

bool ChessBoard::isUnderAttack(ChessPiece pieceToCheck) const
{
    ChessBoard::Pieces pieces(mPieces);
    pieces.push_back(pieceToCheck);
    ChessBoard testBoard(pieces);
    for (const auto& piece : pieces)
    {
        if (piece.getColor() != pieceToCheck.getColor() &&
            testBoard.isLegalMove(piece, pieceToCheck.getCoordinates()))
        {
            return true;
```

```
        }
    }

    return false;
}

bool ChessBoard::isUnderAttack(Chess::Coordinates coordinatesToCheck, bool checkForCheck) const
{
//    ChessBoard::Pieces pieces(mPieces);
    assert(!isEmptyCell(coordinatesToCheck));
    ChessPiece &pieceToCheck = *mBoard.at(coordinatesToCheck);
//    ChessBoard testBoard(pieces);
    bool res = false;
    for (const auto& piece : mPieces)
    {
        if (piece.getColor() != pieceToCheck.getColor() &&
            isLegalMove(piece, pieceToCheck.getCoordinates(), checkForCheck))
        {
            res = true;
            break;
        }
    }

//    std::cerr << *this << std::endl;
//    std::cerr << "piece " << pieceToCheck <<  " is " << (res ? "under attack" : "safe") << std::endl;
    return res;
}

Chess::Coordinates ChessBoard::getKingCoordinates(Chess::PieceColor color) const
{
    auto kingIterator = std::find_if(mPieces.begin(), mPieces.end(),
                                     [color] (const auto &piece)
    {
        return piece.getType() == Chess::King && piece.getColor() == color;
    });
    assert(kingIterator != mPieces.end());
    return kingIterator->getCoordinates();
}

bool ChessBoard::bothKingsAreOnBoard() const
{
    auto whiteKing = std::find_if(mPieces.begin(), mPieces.end(),
                                  [] (const auto &piece)
    {
        return piece.getColor() == Chess::White && piece.getType() == Chess::King;
    });
    auto blackKing = std::find_if(mPieces.begin(), mPieces.end(),
                                  [] (const auto &piece)
    {
        return piece.getColor() == Chess::Black && piece.getType() == Chess::King;
    });
    return whiteKing != mPieces.end() && blackKing != mPieces.end();
}

bool ChessBoard::makeMove(const std::string &moveStr)
{
    ChessMove move(moveStr);
    Chess::Coordinates from = move.getFrom();
    Chess::Coordinates to = move.getTo();

    if (isEmptyCell(from))
    {
        return  false;
```

```
    }

    return makeMove(*mBoard[from], to);
}

bool ChessBoard::makeMove(size_t pieceIndex, Chess::Coordinates destination)
{
    return makeMove(pieceIndex, destination, true);
}

bool ChessBoard::makeMove(const ChessPiece &piece, Chess::Coordinates destination, bool checkMove)
{
    auto pieceIter = std::find(mPieces.begin(), mPieces.end(), piece);

    return pieceIter != mPieces.end()
            && makeMove(std::distance(mPieces.begin(), pieceIter), destination, checkMove);
}

bool ChessBoard::makeMove(size_t pieceIndex, Chess::Coordinates destination, bool checkMove)
{

    if (pieceIndex >= mPieces.size())
    {
        return false;
    }

    ChessPiece &piece = mPieces[pieceIndex];

//    std::cerr << "Current piece " << piece << " move: " <<
CoordinatesToStr(mPieces[pieceIndex].getCoordinates()).c_str()
//              << "-" << CoordinatesToStr(destination).c_str() << std::endl;

    if (checkMove && !isLegalMove(piece, destination))
    {
        return false;
    }

    if (isCastling(piece, destination))
    {
        ChessPiece &rock = getCastlingRock(piece, destination);
        piece.setCoordinates(destination);
        rock.setCoordinates(getRockCoordinatesAfterCastle(rock));
    }
    else
    {
        auto captureCoordinates = getCaptureCoordinates(piece, destination);

        updateEnPassantCoordinates(piece, destination);
        piece.setCoordinates(destination);

        if (!isEmptyCell(captureCoordinates))
        {
            auto pieceToCapture = std::find(mPieces.begin(), mPieces.end(), *mBoard.at(captureCoordinates));
            if (pieceToCapture != mPieces.end())
            {
                mPieces.erase(pieceToCapture);
            }
        }
    }

    updateBoard();

    if (checkMove)
```

```
    {
        updateGameState(piece.getColor());
    }

    return true;
}

void ChessBoard::updateBoard()
{
    mBoard.fill(nullptr);
    for (auto &piece : mPieces)
    {
        mBoard[piece.getCoordinates()] = &piece;
    }
}

void ChessBoard::updateGameState(Chess::PieceColor currentPlayer)
{
    if (bothKingsAreOnBoard())
    {
        mIsFinished = getPossibleMoves(getOponentColor(currentPlayer)).empty() == true;

        if (mIsFinished == false)
            return;

        // Move out as isPlayerChecked()
        Chess::Coordinates oponentKingCoordinates = getKingCoordinates(getOponentColor(currentPlayer));
        mIsCheckMate = isUnderAttack(oponentKingCoordinates, true);

        if (mIsCheckMate == true)
        {
            mWinner = currentPlayer;
        }
        else
        {
            mWinner = Chess::PieceColor::None;
        }
    }
}

Chess::PieceColor ChessBoard::getOponentColor(Chess::PieceColor currentPlayerColor)
{
    return currentPlayerColor == Chess::White ? Chess::Black : Chess::White;
}

ChessBoard::Moves ChessBoard::getPossibleMoves(Chess::PieceColor sideColor) const
{
    std::vector<std::string> moves;
//    moves.reserve(100);

    for (const auto &piece : mPieces)
    {
        if (piece.getColor() != sideColor)
        {
            continue;
        }

        for (unsigned coordinates = 0; coordinates < BOARD_SIZE; coordinates++)
        {
            if (isLegalMove(piece, coordinates))
            {
                std::string move = CoordinatesToStr(piece.getCoordinates())
                                   + CoordinatesToStr(coordinates);
```

```
                moves.push_back(move);
            }
        }
    }

//    moves.shrink_to_fit();
    return moves;
}

void ChessBoard::updateEnPassantCoordinates(const ChessPiece &piece, Chess::Coordinates destination)
{
    mEnPassantCoordinate = INVALID_COORDINATES;

    if (isLegalPawnBigMove(piece, destination))
        mEnPassantCoordinate = destination;
}

bool ChessBoard::isKingSafeAfterMove(const ChessPiece &piece, Chess::Coordinates destination) const
{
    ChessBoard testBoard(mPieces);
    testBoard.makeMove(piece, destination, false);
    Chess::Coordinates testKingCoordinates = testBoard.getKingCoordinates(piece.getColor());
//    qDebug() << "original king coordinates:" << piece.getCoordinates()
//             << "test king coordinates:" << testKingCoordinates;

    return !testBoard.isUnderAttack(testKingCoordinates, false);
}

bool ChessBoard::isLegalMove(const ChessPiece &piece, Chess::Coordinates destination, bool
needCheckForCheck) const
{
    bool isLegalMove = true;

    if (destination > BOARD_LAST_INDEX)
    {
        qCritical() << "Imposible move! destination it out of board -" << destination;
        return false;
    }

    if (piece.getCoordinates() == destination)
    {
//        qWarning() << "Staying on the same possition is not a move!";
        return false;
    }

    switch (piece.getType()) {
    case Chess::King:
        isLegalMove = isLegalKingMove(piece, destination)
                    || isLegalCastling(piece, destination);
        break;
    case Chess::Queen:
        isLegalMove = isLegalQueenMove(piece, destination);
        break;
    case Chess::Bishop:
        isLegalMove = isLegalBishopMove(piece, destination);
        break;
    case Chess::Knight:
        isLegalMove = isLegalKnightMove(piece, destination);
        break;
    case Chess::Rock:
        isLegalMove = isLegalRockMove(piece, destination);
        break;
    case Chess::Pawn:
```

```
        isLegalMove = isLegalPawnMove(piece, destination)
                    || isLegalPawnCapture(piece, destination);
        break;
    default:
        qCritical() << "Unkown type of piece ->" << piece.getType();
    }

    if (isLegalMove == false)
        return false;

    isLegalMove = noObsactlesOnPieceWay(piece, destination);

    if (isLegalMove && !isEmptyCell(destination))
        isLegalMove = canCapturePiece(piece, destination);

    if (needCheckForCheck && isLegalMove)
        if (bothKingsAreOnBoard())
            isLegalMove = isKingSafeAfterMove(piece, destination);
        else
            qWarning() << "Could not find two kings on the board. Won't check if king is safe after this move";

    return isLegalMove;
}

unsigned getClearRow(unsigned orientedRow, Chess::PieceColor pieceColor)
{
    return pieceColor == Chess::White ? BOARD_SIDE_LAST_INDEX - orientedRow
                                      : orientedRow;
}

bool isPawnOnStartPosition(Chess::Coordinates coordinates, Chess::PieceColor pieceColor)
{
    unsigned currentRow = coordinates / BOARD_SIDE_SIZE;
    unsigned startingRow = getClearRow(1, pieceColor);
    return currentRow == startingRow;
}

bool isKingOnStartPosition(Chess::Coordinates coordinates, Chess::PieceColor pieceColor)
{
    const unsigned currentColumn = coordinates % BOARD_SIDE_SIZE;
    const unsigned startingColumn = 4;
    const unsigned currentRow = coordinates / BOARD_SIDE_SIZE;
    const unsigned startingRow = getClearRow(0, pieceColor);
    return currentColumn == startingColumn
           && currentRow == startingRow;
}

bool ChessBoard::isLegalPawnMove(const ChessPiece &piece, Chess::Coordinates destination)
{
    PieceMove move(piece.getCoordinates(), destination, piece.getColor());

    return move.getVector() == PieceMove::Vector{0, 1}
        || isLegalPawnBigMove(piece, destination);
}

bool ChessBoard::isLegalPawnCapture(const ChessPiece &piece, Chess::Coordinates destination) const
{
    if (piece.getType() != Chess::Pawn)
        return false;

    PieceMove move(piece.getCoordinates(), destination, piece.getColor());
    if (abs(move.getVector().getX()) == 1 && move.getVector().getY() == 1
            && !isEmptyCell(destination)
```

```
            && mBoard[destination]->getColor() != piece.getColor())
        return true;
    else
        return isEnPassantPossible(piece, destination);
}

bool ChessBoard::isLegalPawnBigMove(const ChessPiece &piece, Chess::Coordinates destination)
{
    PieceMove pieceMove(piece.getCoordinates(), destination, piece.getColor());

    return isPawnOnStartPosition(piece.getCoordinates(), piece.getColor())
        && (pieceMove.getVector() == PieceMove::Vector{0, 2});
}

bool ChessBoard::isLegalRockMove(const ChessPiece &piece, Chess::Coordinates destination)
{
    PieceMove move(piece.getCoordinates(), destination, piece.getColor());

    return (move.getVector().getX() == 0 && move.getVector().getY() != 0)
        || (move.getVector().getX() != 0 && move.getVector().getY() == 0);
}

bool ChessBoard::isLegalBishopMove(const ChessPiece &piece, Chess::Coordinates destination)
{
    PieceMove move(piece.getCoordinates(), destination, piece.getColor());

    return (abs(move.getVector().getX()) == abs(move.getVector().getY()));
}

bool ChessBoard::isLegalQueenMove(const ChessPiece &piece, Chess::Coordinates destination)
{
    PieceMove move(piece.getCoordinates(), destination, piece.getColor());

    return ((move.getVector().getX() == 0 && move.getVector().getY() != 0)
        || (move.getVector().getX() != 0 && move.getVector().getY() == 0)
        || (abs(move.getVector().getX()) == abs(move.getVector().getY())));
}

bool ChessBoard::isLegalKingMove(const ChessPiece &piece, Chess::Coordinates destination)
{
    PieceMove move(piece.getCoordinates(), destination, piece.getColor());

    return abs(move.getVector().getX()) <= 1 && abs(move.getVector().getY()) <= 1;
}

Chess::Coordinates ChessBoard::getRockCoordinatesAfterCastle(const ChessPiece &piece)
{
    if (piece.getCoordinates() == 0)
        return 0 + Right + Right + Right;
    if (piece.getCoordinates() == Coordinates("a1"))
        return Coordinates("a1") + Right + Right + Right;
    if (piece.getCoordinates() == Coordinates("h8"))
        return Coordinates("h8") + Left + Left;
    if (piece.getCoordinates() == Coordinates("h1"))
        return Coordinates("h1") + Left + Left;
    return INVALID_COORDINATES;
}

ChessPiece &ChessBoard::getCastlingRock(const ChessPiece &piece, Chess::Coordinates destination)
{
    PieceMove move(piece.getCoordinates(), destination, piece.getColor());
    Chess::Coordinates rockCoordinates = INVALID_COORDINATES;
```

```
    if (move.getClearDirection() == PieceMove::Direction::Left)
    {
        rockCoordinates = destination + Left + Left;
    }
    else if (move.getClearDirection() == PieceMove::Direction::Right)
    {
        rockCoordinates = destination + Right;
    }

    if (isEmptyCell(rockCoordinates))
    {
        std::cerr << "current board: " << *this << std::endl;
        std::cerr << "King: " << piece << " destination: " << CoordinatesToStr(destination) << std::endl;
        std::cerr << "rockCoordinates: " << rockCoordinates << std::endl;
    }
    assert(!isEmptyCell(rockCoordinates));
    return *mBoard.at(rockCoordinates);
}

bool ChessBoard::isCastling(const ChessPiece &piece, Chess::Coordinates destination) const
{
    PieceMove move(piece.getCoordinates(), destination, piece.getColor());

    if (piece.getType() != Chess::King
        || !isKingOnStartPosition(piece.getCoordinates(), piece.getColor()))
    {
        return false;
    }

    // XXX: Why don't check for rock existance on edge squares?
    Chess::Coordinates rockCoordinates = INVALID_COORDINATES;
    if (move.getClearVector() == PieceMove::Vector{-2, 0})
    {
        rockCoordinates = Coordinates(BOARD_SIDE_FIRST_INDEX,
                                      getClearRow(BOARD_SIDE_FIRST_INDEX, piece.getColor()));

    }
    else if (move.getClearVector() == PieceMove::Vector{2, 0})
    {
        rockCoordinates = Coordinates(BOARD_SIDE_LAST_INDEX,
                                      getClearRow(BOARD_SIDE_FIRST_INDEX, piece.getColor()));
    }

    if (rockCoordinates == INVALID_COORDINATES)
    {
        return false;
    }

    ChessPiece *rockPtr = mBoard.at(rockCoordinates);
    return !isEmptyCell(rockCoordinates)
            && rockPtr->getType() == Chess::Rock
            && rockPtr->getColor() == piece.getColor();

    return false;
}

bool ChessBoard::isLegalCastling(const ChessPiece &piece, Chess::Coordinates destination) const
{
    PieceMove move(piece.getCoordinates(), destination, piece.getColor());

    if (isCastling(piece, destination))
    {
        return isKingSafeAfterMove(piece, piece.getCoordinates() + static_cast<int>(move.getClearDirection()));
```

```
    }

    return false;
}

bool ChessBoard::isLegalKnightMove(const ChessPiece &piece, Chess::Coordinates destination)
{
    PieceMove move(piece.getCoordinates(), destination, piece.getColor());

    return (abs(move.getVector().getX()) == 1 && abs(move.getVector().getY()) == 2)
        || (abs(move.getVector().getX()) == 2 && abs(move.getVector().getY()) == 1);
}

bool ChessBoard::noObsactlesOnPieceWay(const ChessPiece &piece, Chess::Coordinates destination) const
{
    PieceMove move(piece.getCoordinates(), destination, piece.getColor());

    PieceMove::Vector vector = move.getVector();
    if (piece.getType() != Chess::Knight)
    {
        int steps = std::max(abs(vector.getY()), abs(vector.getX())) - 1;
        Chess::Coordinates currentCoordinates = piece.getCoordinates();
        while (steps > 0)
        {
            if (isEmptyCell(currentCoordinates + move.getClearDirection()) == false)
            {
                return false;
            }
            currentCoordinates = currentCoordinates + move.getClearDirection();
            steps--;
        }
    }

    return true;
}

bool ChessBoard::canCapturePiece(const ChessPiece &piece, Chess::Coordinates destination) const
{
    if (piece.getType() != Chess::Pawn)
    {
        return mBoard[destination]->getColor() != piece.getColor();
    }
    else
    {
        return isLegalPawnCapture(piece, destination);
    }
}

int ChessBoard::orientMove(int move, Chess::PieceColor color)
{
    return move * (color == Chess::White ? 1 : -1);
}

Chess::Coordinates ChessBoard::getCaptureCoordinates(const ChessPiece &piece, Chess::Coordinates
destination)
{
    if (isEnPassantPossible(piece, destination))
        return destination + orientMove(Down, piece.getColor());
    return destination;
}

bool ChessBoard::isEnPassantPossible(const ChessPiece &piece, Chess::Coordinates destination) const
{
```

```
    if (piece.getType() != Chess::Pawn || mEnPassantCoordinate > BOARD_LAST_INDEX)
    {
        return false;
    }

    PieceMove move(piece.getCoordinates(), destination, piece.getColor());

    if (abs(move.getVector().getX()) == 1 && move.getVector().getY() == 1
        && mEnPassantCoordinate == destination + orientMove(Down, piece.getColor()))
    {
        return true;
    }

    return false;
}

bool ChessBoard::isFinished() const
{
    return mIsFinished;
}

bool ChessBoard::isCheckMate() const
{
    return mIsCheckMate;
}

Chess::PieceColor ChessBoard::getWinner() const
{
    return mWinner;
}

bool ChessBoard::isEmptyCell(Chess::Coordinates destination) const
{
    return destination > BOARD_LAST_INDEX || mBoard[destination] == nullptr;
}

std::ostream& operator<<(std::ostream &out, const Chess::PieceColor &pieceColor)
{
    out << (pieceColor == Chess::White ? "white" : "black");
    return out;
}

std::ostream& operator<<(std::ostream &out, const Chess::PieceType &pieceType)
{
    switch(pieceType)
    {
    case Chess::King:
        out << "King";
        break;
    case Chess::Queen:
        out << "Queen";
        break;
    case Chess::Bishop:
        out << "Bishop";
        break;
    case Chess::Knight:
        out << "Knight";
        break;
    case Chess::Rock:
        out << "Rock";
        break;
    case Chess::Pawn:
        out << "Pawn";
```

```
        break;
    }

    return out;
}

std::ostream& operator<<(std::ostream &out, const ChessPiece &piece)
{
    out << "color[" << piece.getColor() << "] "
        << "type[" << piece.getType() << "] "
        << "coordinates[" << CoordinatesToStr(piece.getCoordinates()) << "]";
    return out;
}

//----------------------- TESTS -----------------------//

using PossibleMoves = std::vector<bool>;

inline unsigned coordinatesToNumber(const std::string &strCoordinates)
{
    unsigned column = strCoordinates[0] - 'a';
    unsigned row =  (BOARD_SIDE_SIZE - 1) - (strCoordinates[1] - '1');
    assert(column < BOARD_SIDE_SIZE);
    assert(row < BOARD_SIDE_SIZE);
    return column + BOARD_SIDE_SIZE * row;
}

inline bool operator==(const ChessPiece &lhs, const ChessPiece &rhs)
{
    return lhs.getColor() == rhs.getColor() &&
           lhs.getType() == rhs.getType() &&
           lhs.getCoordinates() == rhs.getCoordinates();
}

template <typename T1, typename T2>
std::ostream& operator<<(std::ostream &out, const std::pair<T1, T2> &var)
{
    out << "first[" << var.first << "] "
        << "second[" << var.second << "]";
    return out;
}

std::ostream& operator<<(std::ostream &out, const std::pair<unsigned, unsigned> &var)
{
    out << "first[" << var.first << "] "
        << "second[" << var.second << "]";
    return out;
}

std::string PieceToStr(const ChessPiece &piece)
{
    char pieceSymbol = ' ';
    switch (piece.getType())
    {
    case Chess::King:
        pieceSymbol = 'k';
        break;
    case Chess::Queen:
        pieceSymbol = 'q';
        break;
    case Chess::Bishop:
        pieceSymbol = 'b';
        break;
```

```
    case Chess::Knight:
        pieceSymbol = 'n';
        break;
    case Chess::Rock:
        pieceSymbol = 'r';
        break;
    case Chess::Pawn:
        pieceSymbol = 'p';
        break;
    }

    if (piece.getColor() == Chess::White)
        pieceSymbol = toupper(pieceSymbol);

    return std::string(1, pieceSymbol);
}

std::ostream& operator<<(std::ostream &out, const ChessBoard &board)
{
    out << "board {\n";
    out << "____________________\n";
    for (unsigned row = 0; row < BOARD_SIDE_SIZE; row++)
    {
        for (unsigned column = 0; column < BOARD_SIDE_SIZE; column++)
        {
            auto cell = board.mBoard[Coordinates(column, row)];
            if (cell == nullptr)
                out << ". ";
            else
                out << PieceToStr(*cell) << " ";
        }
        out << "|" << BOARD_SIDE_SIZE - row << " \n";
    }
    out << "-----------------|\n";
    out << "a b c d e f g h\n}\n";
    return out;
}

namespace std {
std::ostream& operator<<(std::ostream &out, const PossibleMoves &board)
{
    out << "legal moves{\n";
    for (unsigned row = 0; row < BOARD_SIDE_SIZE; row++)
    {
        for (unsigned column = 0; column < BOARD_SIDE_SIZE; column++)
        {
            out << board[row * BOARD_SIDE_SIZE + column] << " ";
        }
        out << "\n";
    }
    out << "}\n";
    return out;
}
}

static const ChessBoard::Pieces defaultPieces{
    {ChessPiece{Chess::Black, Chess::Rock, 0}},
    {ChessPiece{Chess::Black, Chess::Knight, 1}},
    {ChessPiece{Chess::Black, Chess::Bishop, 2}},
    {ChessPiece{Chess::Black, Chess::Queen, 3}},
    {ChessPiece{Chess::Black, Chess::King, 4}},
    {ChessPiece{Chess::Black, Chess::Bishop, 5}},
    {ChessPiece{Chess::Black, Chess::Knight, 6}},
```

```
    {ChessPiece{Chess::Black, Chess::Rock, 7}},

    {ChessPiece{Chess::Black, Chess::Pawn, 0 + 1*8}},
    {ChessPiece{Chess::Black, Chess::Pawn, 1 + 1*8}},
    {ChessPiece{Chess::Black, Chess::Pawn, 2 + 1*8}},
    {ChessPiece{Chess::Black, Chess::Pawn, 3 + 1*8}},
    {ChessPiece{Chess::Black, Chess::Pawn, 4 + 1*8}},
    {ChessPiece{Chess::Black, Chess::Pawn, 5 + 1*8}},
    {ChessPiece{Chess::Black, Chess::Pawn, 6 + 1*8}},
    {ChessPiece{Chess::Black, Chess::Pawn, 7 + 1*8}},

    {ChessPiece{Chess::White, Chess::Pawn, 0 + 6*8}},
    {ChessPiece{Chess::White, Chess::Pawn, 1 + 6*8}},
    {ChessPiece{Chess::White, Chess::Pawn, 2 + 6*8}},
    {ChessPiece{Chess::White, Chess::Pawn, 3 + 6*8}},
    {ChessPiece{Chess::White, Chess::Pawn, 4 + 6*8}},
    {ChessPiece{Chess::White, Chess::Pawn, 5 + 6*8}},
    {ChessPiece{Chess::White, Chess::Pawn, 6 + 6*8}},
    {ChessPiece{Chess::White, Chess::Pawn, 7 + 6*8}},

    {ChessPiece{Chess::White, Chess::Rock, 0 + 7*8}},
    {ChessPiece{Chess::White, Chess::Knight, 1 + 7*8}},
    {ChessPiece{Chess::White, Chess::Bishop, 2 + 7*8}},
    {ChessPiece{Chess::White, Chess::Queen, 3 + 7*8}},
    {ChessPiece{Chess::White, Chess::King, 4 + 7*8}},
    {ChessPiece{Chess::White, Chess::Bishop, 5 + 7*8}},
    {ChessPiece{Chess::White, Chess::Knight, 6 + 7*8}},
    {ChessPiece{Chess::White, Chess::Rock, 7 + 7*8}},
};
```