DLA POPRAWNEGO FORMATOWANIA - STRONA DO USUNIĘCIA

Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kierunek: **Informatyka algorytmiczna (INA)**

PRACA DYPLOMOWA INŻYNIERSKA

Bot dla gry w Szachy

Bot for Chess game

Krzysztof Wiśniewski

Opiekun pracy
dr Maciej Gębala, prof. uczelni

Słowa kluczowe: szachy, bot, teoria gier

WROCŁAW, 2025

Streszczenie

Celem pracy było opracowanie silnika szachowego w języku Java. Program ten zaprojektowano tak, aby analizować i oceniać pozycję na szachownicy, a następnie sugerować graczowi najlepszy ruch, uwzględniając jego strategiczne i taktyczne aspekty. Interakcja z programem odbywa się za pośrednictwem wiersza poleceń, z wykorzystaniem Uniwersalnego Interfejsu Szachowego. Umożliwiło to łatwą integrację z innymi aplikacjami szachowymi oraz pozwoliło na przeprowadzanie symulacji i analiz działania silnika bez potrzeby aplikowania interfejsu graficznego.

Niniejsza praca inżynierska składa się z trzech części, w których omówiono kolejne etapy pracy nad opracowaniem silnika szachowego. W pierwszej części przedstawiono podstawową wersję programu, która obejmuje generowanie możliwych ruchów zgodnie z zasadami gry w szachy, algorytm wyszukiwania optymalnego ruchu oraz implementację naiwnej heurystyki. W części drugiej opisano ulepszenia algorytmów wyszukiwania i oceny pozycji, mające na celu zwiększenie efektywności i precyzji silnika. Ostatnią część pracy poświęcono zagadnieniom związanym z testowaniem siły programu. Przeprowadzono analizę wydajności w odniesieniu do różnych jego wersji, uwzględniając testy porównawcze oraz metodologię oceny skuteczności.

Efektem prac jest silnik, którego ranking szachowy można szacować na zakres pomiędzy 1500 a 1600 ELO, co plasuje go na poziomie porównywalnym z graczem posiadającym III kategorię szachową.

Słowa kluczowe: szachy, bot, teoria gier

Abstract

The aim of this thesis is to develop a chess engine in Java. This program is designed to analyze and evaluate position on the chessboard and subsequently suggest the best move for the player, considering its both strategic and tactical aspects. Interaction with the program is conducted via the command line, using the Universal Chess Interface. This allows for easy integration with other chess applications and facilitates the simulation and analysis of the engine's performance without the need to create graphical interface.

This engineering thesis consists of three parts, which discuss the successive stages of developing the chess engine. The first part presents the basic version of the program, which includes generating possible moves according to the rules of chess, the minimax algorithm for searching for the optimal move, and the implementation of a naive heuristic. The second part implements improvements to the search and position evaluation algorithms, aiming to increase the efficiency and precision of the engine. The final part of the thesis is dedicated to issues related to testing the engine's strength. An analysis of performance was conducted with respect to various versions of the engine, including comparative tests and methodology for assessing effectiveness.

Keywords: chess, bot, game theory

Spis treści

1. Wstęp 10
1.1. Wprowadzenie 10
1.2. Cel i zakres 10
1.3. Układ pracy 11
2. Implementacja silnika szachowego 12
2.1. Komunikacja z systemem 12
2.1.1. Notacja Forsytha-Edwardsa 12
2.1.2. Szachowa Notacja Algebraiczna 13
2.1.3. Uniwersalny Interfejs Szachowy 13
2.2. Reprezentacja pozycji 14
2.2.1. Reprezentacja szachownicy In progress 14
2.2.2. Reprezentacja stanu Implemented 15
2.2.3. Reprezentacja ruchu Implemented 15
2.3. Generowanie ruchów 15
2.3.1. Generowanie ruchów pseudolegalnych Implemented 15
2.3.2. Generowanie ruchów legalnych Implemented 15
2.4. Algorytm wyszukiwania 15
2.4.1. Algorytm min-max Implemented 15
2.4.2. Iteracyjne pogłębianie wyszukiwania Implemented 15
2.4.3. Zarządzanie czasem Implemented 15
2.5. Ocena heurystyczna 15
2.5.1. Heurystyka stanu gry Implemented 15
2.5.2. Heurystyka liczebności bierek Implemented 15
3. Ulepszenia dla silnika szachowego 16
3.1. Ulepszenia dla wyszukiwania 16
3.1.1. Biblioteka otwarć TODO 16
3.1.2. Alfa-Beta cięcie Implemented 16
3.1.3. Ewaluacja cichych stanów TODO 16
3.1.4. Sortowanie ruchów TODO 16
3.1.5. Tabela transpozycji TODO 16
3.1.6. Okno estymacji TODO 16
3.1.7. Rozszerzanie wyszukiwania TODO 17

3.2. Ulepszenia dla oceny heurystycznej 17
3.2.1. Tablice figur Implemented 17
3.2.2. Ochrona króla TODO 17
3.2.3. Struktura pionów TODO 17
3.2.4. Moment gry TODO 17
3.2.5. Mobilność TODO 17
4. Ocena siły silnika 18
4.1. Metodologia badawcza Implemented 18
4.2. Porównanie wersji silnika TODO 18
4.3. Porównanie z innymi wersjami silnika TODO 18
4.4. Porównanie z graczami TODO 18
5. Zakończenie 19
5.1. Podsumowanie pracy TODO 19
5.2. Możliwości dalszego rozwoju aplikacji TODO 19
Bibliografia 20
A. Instrukcja wdrożenia TODO 21
B. Przykład użycia UCI TODO 22

Spis rysunków

2.1. Przykładowe pozycje szachowe: a) startowa, b) po ruchu e2e4 13

Spis tabel

B.1. UCI - komunikacja GUI do silnika 22
B.2. UCI - komunikacja silnika do GUI 22

Spis listingów

2.1. Implementacja reprezentacji szachownicy tablicą pól 14
2.2. Implementacja reprezentacji szachownicy tablicami bitowymi bierek 14

Skróty

UCI (ang. *Universal Chess Interface*) Uniwersalny Interfejs Szachowy
FEN (ang. *Forsyth–Edwards Notation*) Notacja Forsytha-Edwardsa
LAN (ang. *Long Algebraic Notation*) Pełna Algebraiczna Notacja Szachowa
Perft (ang. *Performance Test*) Test Wydajności

Rozdział 1

Wstęp

1.1. Wprowadzenie

Szachy, powszechnie nazywane grą królewską, to jedna z najstarszych, a zarazem najpopularniejszych form intelektualnej rozrywki w dziejach ludzkości. Niekwestionowaną popularność tak wśród profesjonalistów, jak i amatorów zawdzięczają połączeniu prostoty zasad ze złożoności strategicznych wyzwań. Ich historia, sięgająca VI wieku p.n.e., obejmuje stale ponawiane próby udoskonalania reguł i odkrywania nowych, coraz bardziej zaawansowanych, taktyk mających zagwarantować zwycięstwo nad oponentem. Wprowadzenie roszady, ruchu en-passant to najbardziej spektakularne przykłady zmian, świadczących o nieograniczonej kreatywności kolejnych pokoleń graczy.

W ponad tysiąc pięćsetletniej historii szachów, szczególne znaczenie, miało upowszechnienie w połowie XX wieku zaawansowanych maszyn liczących. Otworzyło ono możliwość zautomatyzowania procesu analizy partii szachowych.

Za pioniera w tej dziedzinie uważa się amerykańskiego matematyka Claude Shannon, który w roku 1950 opublikował pracę o teoretycznych aspektach programowania silników szachowych, opartych o ocenę heurystyczną oraz algorytm min-max. Istotny wkład w rozwój szachowej sztucznej inteligencji miał także ojciec informatyki — Alan Turing, który rok po publikacji pracy Shannona zaprojektował pierwszy program komputerowy, w pełni zdolny do gry w szachy. Ograniczenia techniczne tamtych czasów nie pozwoliły jednak na przetestowanie programu na maszynie. Rozegrano niewielką liczbę partii szachowych, w których każdy ruch był obliczany analogowo.

Najstarszy program uruchomiony na komputerze, który pozwalał na przeprowadzenie pełnej rozgrywki, powstał w 1958 roku. Od tamtego momentu wiele silników szachowych biło rekordy swoich poprzedników. Do przełomu doszło zimą 1997 roku, kiedy to silnik szachowy DeepBlue wygrał pojedynek $3\frac{1}{2} - 2\frac{1}{2}$ z ówczesnym mistrzem świata, Garrym Kasparovem.

Po tym wydarzeniu świat wkroczył w erę super silników. Szachy stały się nie tylko areną dla ludzkiego intelektu, ale także polem testowym zaawansowanych technologii. Współcześnie, wykorzystanie komputerów stanowi nieodłączny element analizy partii szachowych. Zastosowanie najnowocześniejszych rozwiązań, takich jak uczenie maszynowe i sieci neuronowe, zrewolucjonizowało sposób, w jaki rozumiemy tą gre, oraz pokazało, jak wiele jeszcze można w tej dziedzinie osiągnąć.

1.2. Cel i zakres

Zasadniczym celem pracy było stworzenie silnika szachowego, zdolnego do oceny heurystycznej pozycji, oraz proponowania graczowi ruchów z uwzględnieniem ich strategicznych aspektów.

Zakres pracy obejmuje następujące zagadnienia:

- Przegląd literatury na temat technik oraz algorytmów wykorzystywanych przy tworzeniu nowoczesnych silników szachowych.
- Zapoznanie się z powszechnie obowiązującymi zasadami turniejowej gry w szachy, opublikowanymi przez Międzynarodową Federację Szachową.
- Stworzenie silnika szachowego w języku programowania Java 22, z celowym pominięciem dodatkowych rozwiązań open-source.
- Wykorzystanie Uniwersalnego Interfejsu Szachowego do komunikacji z systemem.
- Zintegrowanie silnika z wybranym interfejs graficzny.
- Testowanie poprawności stworzonego oprogramowania.
- Implementacja rozwiązań programistycznych przyspieszających przeszukiwanie drzewa decyzyjnego oraz ulepszających dokładność oceny heurystycznej.
- Przeprowadzenie analizy porównawczej pomiędzy wersjami systemu w celu oceny efektywności zastosowanych rozwiązań.
- Porównanie najlepszej wersji silnika z już istniejącymi rozwiązaniami w celu określenia poziomu gry.

Techniki wykorzystujące uczenie maszynowe, choć zostały incydentalnie wspomniane w pracy, wykraczają poza jej zakres.

1.3. Układ pracy

Niniejsza praca inżynierska składa się z trzech głównych części.

W pierwszej z nich opisano elementy oprogramowania konieczne do stworzenia podstawowej wersji silnika szachowego. Przedstawiono metody komunikacji z interfejsem, techniki reprezentacji pozycji, algorytm min-max wraz z metodologią zarządzania czasem gry. Przetestowano aplikację w celu sprawdzenia poprawności działania. Choć do zrozumienia pracy konieczna jest znajomość zasad gry w szachy, to w tym rozdziale mówiono także niektóre, bardziej zawiłe bądź mniej znane, jej aspekty.

Następny rozdział poświęcono pracy nad ulepszeniem systemu. Przedstawiono zaimplementowane rozwiązania, mające na celu poprawę poziomu gry systemu. Skupiono się na dwóch kierunkach: poprawy prędkości przeszukiwania oraz na poprawie precyzji oceny heurystycznej. Jako że dla słabych silników o wiele większe znaczenie w poprawie jego gry ma pierwszy z tych aspektów, omówiono go na początku.

Ostatnią część pracy poświęcono testom wydajnościowym oraz jakościowym. Przetestowano w pierwszej kolejności program pomiędzy różnymi jego wersjami. W dalszej kolejności przetestowano system przeciwko innym, publicznie dostępnym silnikom. Na końcu przedstawiono podsumowanie rozgrywek przeprowadzonych z graczami.

W dodatku do pracy zamieszczono instrukcję wdrożenia aplikacji w celu odpalenia silnika we własnym środowisku.

Rozdział 2

Implementacja silnika szachowego

2.1. Komunikacja z systemem

2.1.1. Notacja Forsytha-Edwardsa

W 1950 roku amerykański matematyk Claude Shanon na łamach "Philosophical Magazine" opublikował pracę naukową zatytułowaną "Programowanie komputera do gry w szachy". [3] Stała się ona teoretyczną podstawą dla dalszego rozwoju silników szachowych. Zawarte w niej zostało między innymi oszacowanie, co do ilości możliwych pozycji szachowych, wynoszące 10^{43}. Oznacza to, że liczba legalnych ułożeń planszy przewyższa o rzędy wielkości liczbę gwiazd w widzialnym wszechświecie.

Aby umożliwić użytkownikowi wprowadzenie danych oraz komunikację z programem, należało w pierwszej kolejności sprecyzować format, w jakim zostaną dostarczone informacje dotyczące aktualnej pozycji. Standardem, wykorzystywanym nie tylko w większości silników, ale także w pojedynkach rozgrywanych online, jest Notacja Forsytha-Edwardsa (ang. *Forsyth–Edwards Notation*, w skrócie FEN). Stworzona pierwotnie przez dziennikarza Davida Forsytha, a następnie dostosowana do potrzeb komputerów przez Stevena Edwardsa.

Notacja FEN jest linią znaków ASCII, która pozwala na jednoznaczne określenie aktualnego stanu gry. Wielkimi literami kodowane są bierki białe, małymi natomiast bierki czarne. Każda z nich opisana jest skrótem pochodzącym od ich angielskich nazw:

- P/p — pion
- N/n — skoczek
- B/b — goniec
- R/r — wieża
- Q/q — hetman
- K/k — król

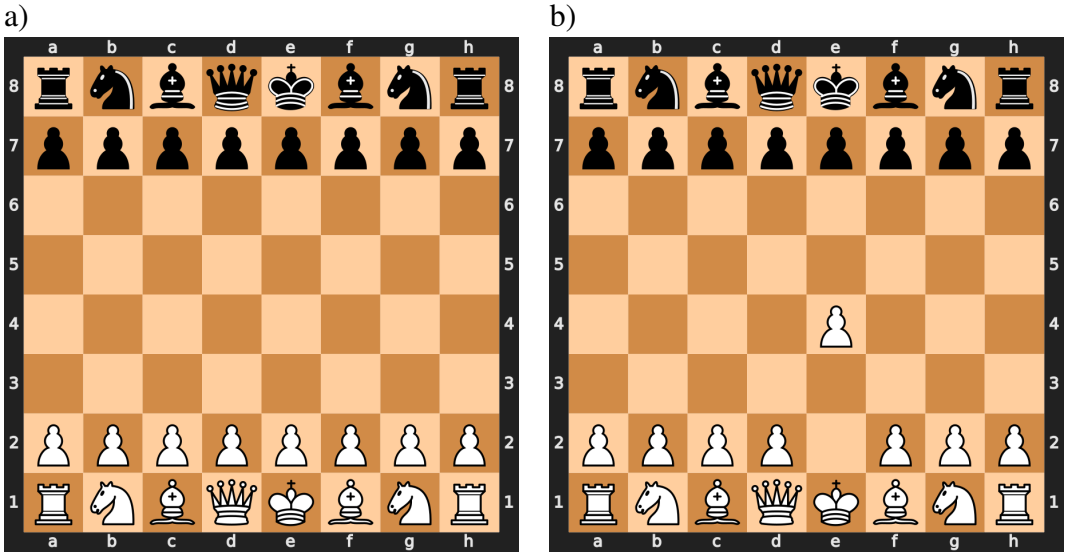
Sześć następujących po sobie pól, oddzielonych spacjami, określa następujące aspekty gry:

1. Reprezentacja 64 pól szachownicy z perspektywy białego gracza. Każdy z wierszy planszy oddzielony jest "/", a jego zawartość opisywana zostaje od kolumny a, do kolumny h. Liczbę nieprzerwanie pustych pól w danym wierszu określa cyfra z zakresu od 1 do 8.
2. Sprecyzowanie, do którego gracza należy następny ruch (w - biały, b - czarny).
3. Przedstawienie możliwości roszady obu stron (K/k - krótka roszada, Q/q - długa roszada).
4. Sprecyzowanie pola będącego celem bicia w przelocie, szerzej znanego jako ruch en passant. Brak możliwości bicia określany jest jako"-"
5. Liczba posunięć od ostatniego bicia bądź ruchu pionem. Wartość ta jest istotna z punktu widzenia reguły 50 posunięć.
6. Liczba pełnych ruchów, która zostaje każdorazowo zwiększana po ruchu czarnych bierek.

Pełna specyfikacja FEN dostępna jest w dokumentacji "Portable Game Notation". [1]

2.1 a) `rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1`

2.1 b) `rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR b KQkq e3 0 1`



Rys. 2.1: Przykładowe pozycje szachowe: a) startowa, b) po ruchu e2e4

2.1.2. Szachowa Notacja Algebraiczna

Kluczowym aspektem, z perspektywy komunikacji z systemem, jest także określenie formatu zapisu ruchów. W aplikacji wykorzystano Szachową Notację Algebraiczną.

Notacja ta, w swojej krótkiej formie, jest powszechnie stosowana w literaturze oraz podczas oficjalnych zawodów. Stanowi jedyną formę zapisu posunięć uznawaną przez Międzynarodową Federację Szachową. Jej zapis zawiera informacje o rodzaju ruszanej bierki oraz o jej polu docelowym. Zapis ten z punktu widzenia komputerów zawiera jednak wadę. W sytuacji, w której dwie bierki tego samego rodzaju mogą poruszyć się na jedno pole, występuje dwuznaczność zapisu. Choć w takiej sytuacji dodaje się do ruchu kolumnę bądź wiersz startowy różniący obie bierki, jest to rozwiązanie wymagające implementacji dodatkowej logiki oraz wiedzy o stanie całej planszy.

Znacznie bardziej intuicyjne dla komputerów jest zastosowanie długiej wersji szachowej notacji algebraicznej. Zawarte są w niej informacje o polu startowym oraz polu docelowym ruchu, usuwając tym samym ryzyko dwuznaczności. Roszady oznaczano przez pola ruchu króla, natomiast do ruchów z promocją dopisano literę określającą rodzaj podmienionej figury.

2.1.3. Uniwersalny Interfejs Szachowy

Uniwersalny Interfejs Szachowy (ang. *Universal Chess Interface*, w skrócie UCI) [2] jest ustandaryzowanym protokołem tekstowym, służącym do wymiany informacji pomiędzy różnymi programami szachowymi. Jego implementacja pozwoliła na komunikację silnika szachowego z wybranymi interfejsami graficznymi oraz środowiskami testowymi.

UCI jest protokołem rozbudowanym, pozwalającym między innymi na rozgrywki innych wersji szachów niż europejskie, dla przykładu Chess960. W silniku zaimplementowano jedynie te z komend, które konieczne były do rozegrania podstawowej partii mierzonej czasowo.

Metodę połączenia z dowolnym programem obsługującym UCI przedstawiono w dodatku A. Opisy komend oraz przykład wymiany informacji pomiędzy aplikacją a GUI zaprezentowano w dodatku B.

2.2. Reprezentacja pozycji

2.2.1. Reprezentacja szachownicy In progress

Istotnym aspektem, z punktu widzenia prędkości działania silnika, było zastosowanie odpowiedniego typu reprezentacji położenia bierek na 64-polowej planszy szachowej. Po zapoznaniu się z proponowanymi w literaturze rozwiązaniami, w programie zdecydowano się zaimplementować dwie redundantne techniki.

Obie charakteryzują się odmiennymi właściwościami, znajdując zastosowanie dla innych algorytmów wewnątrz programu. Różnią się one pod względem gęstości zawartych informacji, szybkości dostępu do danych oraz łatwości modyfikacji. Jedno z nich skupia się na każdym z pól szachownicy (ang. *Square Centric*), drugie natomiast bierze pod uwagę konkretne rodzaje bierek (ang. *Piece Centric*).

Tablica pól szachowych

Naturalnym podejściem do reprezentacji szachownicy wydaje się zastosowanie 64-elementowej tablicy, w której każde pole odpowiada konkretnemu miejscu na planszy. W tej implementacji poszczególne bierki zostały zakodowane liczbami od 1 do 6, natomiast cyfry 0 i 8 odpowiednio reprezentują biały oraz czarny kolor. W ten sposób, za pomocą pojedynczych bajtów, można jednoznacznie określić zarówno typ figury, jak i jej kolor na danym polu.

Taka struktura danych jest szczególnie użyteczna, gdy chcemy szybko odpowiedzieć na pytanie, czy na danym polu znajduje się figura, a jeśli tak, to jaka. Dzięki prostemu indeksowaniu tablicy dostęp do informacji o stanie pojedynczego pola jest bardzo efektywny.

Problem pojawia się w momencie, gdy wymagane jest odnalezienie wszystkich pól zawierających konkretny typ figury. W takim przypadku konieczna staje się iteracja całej tablicy, w celu zidentyfikowania odpowiednich pól. Operacja ta, szczególnie przy wielokrotnym wywołaniu, może znacząco obniżyć wydajność algorytmu.

Listing 2.1: Implementacja reprezentacji szachownicy tablicą pól

```
public byte[] board = new byte[64];

@Override
public void addPieceOnSquare(byte square, byte color, byte piece) {
    square = (byte) (64 - square);
    board[square] = (byte) (color | piece);
}

@Override
public void deletePieceOnSquare(byte square, byte color, byte piece) {
    square = (byte) (64 - square);
    board[square] = 0;
}
```

Tablice bitowe bierek

Listing 2.2: Implementacja reprezentacji szachownicy tablicami bitowymi bierek

```
public long[] bitBoards = new long[15];

@Override
public void addPieceOnSquare(byte square, byte color, byte piece) {
```

```
        bitBoards[piece | color] |= Long.rotateLeft(1L, square-1);
        bitBoards[color] |= Long.rotateLeft(1L, square-1);
    }

    @Override
    public void deletePieceOnSquare(byte square, byte color, byte piece) {
        bitBoards[piece | color] &= ~Long.rotateLeft(1L, square-1);
        bitBoards[color] &= ~Long.rotateLeft(1L, square-1);
    }
```

2.2.2. Reprezentacja stanu Implemented

2.2.3. Reprezentacja ruchu Implemented

2.3. Generowanie ruchów

2.3.1. Generowanie ruchów pseudolegalnych Implemented

Generowanie ruchów króla i skoczka

Generowanie ruchów hetmana, wieży i gońca

Generowanie ruchów piona

2.3.2. Generowanie ruchów legalnych Implemented

Technika usuwania ruchów pseudo-legalnych

Perft test

Z uwagi na złożoność powyższego problemu, konieczne było wykonanie testów jednostkowych, które pozwolą sprawdzić poprawność otrzymywanych tablic ruchów.

2.4. Algorytm wyszukiwania

2.4.1. Algorytm min-max Implemented

2.4.2. Iteracyjne pogłębianie wyszukiwania Implemented

2.4.3. Zarządzanie czasem Implemented

2.5. Ocena heurystyczna

2.5.1. Heurystyka stanu gry Implemented

2.5.2. Heurystyka liczebności bierek Implemented

Rozdział 3

Ulepszenia dla silnika szachowego

3.1. Ulepszenia dla wyszukiwania

3.1.1. Biblioteka otwarć TODO

Biblioteka otwarć jest parsowana z pliku na hashmape (FEN, możliwe ruchy). Losowo wybierany ruch spośród możliwych. Pozwala na randomizację posunięć szczególnie na początku.

3.1.2. Alfa-Beta cięcie Implemented

Zaimplementowane, pozostało wytłumaczyć o co cmon. Powołać się na Papadimitru

3.1.3. Ewaluacja cichych stanów TODO

Po zakończeniu zwykłego min-max należy doprowadzić do stanu ”cichego” to znaczy takiego, gdzie nie ma żadnych dostępnych bić ani roszad. Inaczej może to zaburzyć poprawną interpretację pozycji przez heurystykę.

3.1.4. Sortowanie ruchów TODO

Sortujemy ruchy zaczynając od roszad i bić w celu rozważenia ich na początku. Umożliwi to szybsze działanie alfa-beta cięcia i przez to rozważanie mniejszego drzewa decyzyjnego.

3.1.5. Tabela transpozycji TODO

Pozycje już policzone sa haszowane Zobrist hashing oraz zapisywane. Gdy ponownie (na danym poziomie!?!) natrafimy na ten sam hash, to zwracamy wartość, nie przeszukując niżej drzewa. (Czy można tego użyć przy move ordering także?!?)

3.1.6. Okno estymacji TODO

Z tego co rozumiem, zakłada to, że znajdziemy minimum ruch o danej jakości i przez to odcinamy alfa-beta szybciej. Jest jednak ryzyko, że takiego nie znajdziemy i będziemy musieli szukać od zera w większym oknie

3.1.7. Rozszerzanie wyszukiwania TODO

Wydłużenie o maksymalnie dwa przeszukiwania na danej głębokości o ile jest to ruch szczególny.

3.2. Ulepszenia dla oceny heurystycznej

Gdzieś widziałem, że do elo 1500 o wiele istotniejsze są ulepszenia co do heurystyki niż wyszukiwania. Trzeba znaleźć linka do źródła

3.2.1. Tablice figur Implemented

Tablice dla każdej ze stron i każdej z figur w celu przypisania punktacji.

3.2.2. Ochrona króla TODO

Sprawdzanie, czy król jest chroniony, na przykład przez piony

3.2.3. Struktura pionów TODO

Czy piony są w rządku, czy chronią siebie wzajemnie. Najłatwiej to chyba przez bit-boardy sprawdzać

3.2.4. Moment gry TODO

Początek - środek - koniec. Różne wartości, szczególnie dla króla w zależności od momentu gry.

3.2.5. Mobilność TODO

Możliwość ruchów konkretnych figur, szczególnie gońców i skoczków.

Rozdział 4

Ocena siły silnika

4.1. Metodologia badawcza Implemented

4.2. Porównanie wersji silnika TODO

4.3. Porównanie z innymi wersjami silnika TODO

4.4. Porównanie z graczami TODO

Rozdział 5

Zakończenie

5.1. Podsumowanie pracy TODO

5.2. Możliwości dalszego rozwoju aplikacji TODO

Bibliografia

[1] S. J. Edwards. *Standard: Portable Game Notation Specification and Implementation Guide*. 1994. URL: https://www.thechessdrum.net/PGN_Reference.txt (term. wiz. 04. 09. 2024).

[2] R. Huber i S.-M. Kahlen. *Description of the universal chess interface (UCI)*. 2006. URL: https://backscattering.de/chess/uci/ (term. wiz. 05. 09. 2024).

[3] C. E. Shannon i B. Telephone. "XXII. Programming a Computer for Playing Chess 1". W: *Philosophical Magazine Series 1* 41 (1950), s. 256–275. URL: https://api.semanticscholar.org/CorpusID:12908589.

Dodatek A

Instrukcja wdrożenia TODO

To jest dodatek A

Dodatek B

Przykład użycia UCI TODO

Tab. B.1: UCI - komunikacja GUI do silnika

Komenda	Opis działania
uci	Wytłumaczenie działania komendy
debug	Wytłumaczenie działania komendy
isready	Wytłumaczenie działania komendy
setoption	Wytłumaczenie działania komendy

Tab. B.2: UCI - komunikacja silnika do GUI

Komenda	Opis działania
id	Wytłumaczenie działania komendy
uciok	Wytłumaczenie działania komendy
readyok	Wytłumaczenie działania komendy