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 zaimplementowano ulepszenia algorytmów wyszukiwania i oceny pozycji, mające na celu zwiększenie efektywności i precyzji silnika. Ostatnią część pracy poświęcono zagadnieniom związanym z testowaniem siły silnika. Przeprowadzono analizę wydajności w odniesieniu do różnych jego wersji, uwzględniając testy porównawcze oraz metodologię oceny skuteczności.

Efektem prac jest program, którego efektywny ranking szachowy można szacować na zakres pomiędzy 1500 a 1600 ELO.

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

Spis rysunków

Spis tabel

Spis listingów

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 Written

Szachy są najpopularniejszą grą planszową w historii ludzkości. Dzięki prostocie zasad, a zarazem złożoności strategii gra królewska zyskała uznanie wielu, zarówno profesjonalnych graczy biorących udział w turniejach szachowych, jak i amatorów poszukujących intelektualnych wyzwań.

Jej początki, wywodzące się z Indii, datuje się na VI wiek. Od tamtych czasów szachy znacząco ewoluowały. Przemierzając Persję i świat arabski, w końcu dotarły do Europy, gdzie stała się cenioną rozgrywką dworską. Jej europejską odmianę nazywamy szachami klasycznymi.

W swojej 1500-letniej historii szachy zmieniały się nie tylko oficjalne zasady gry, jak na przykład wprowadzenie roszady, czy ruchu en-passant, ale także stosowane techniki i strategie, mające zagwarantować zwycięstwo. Szczególne znaczenie, dla rozwoju tych strategii, miał rozwój maszyn liczących w XX wieku, co otworzyło możliwość zautomatyzowania procesu analizy partii szachowych.

Za pioniera w tej dziedzinie uważany jest amerykański matematyk 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 później zaprojektował pierwszy program komputerowy, w pełni zdolny do gry w szachy. Ograniczenia techniczne tamtych czasów nie pozwoliły jednak na przetestowanie programu na maszynie. Rozegrano parę 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. Aż do pamiętnej zimy 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. Wykorzystanie komputerów stanowi nieodłączny element analizy partii szachowych. Zastosowanie najnowocześniejszych rozwiązań, takich jak sieci neuronowe, zrewolucjonizowało sposób, w jaki rozumiemy tą gre oraz pokazało, jak wiele jeszcze można w tej dziedzinie osiągnąć.

1.2. Cel i zakres Written

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 Written

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.

Następny rozdział poświęcono pracy nad ulepszeniem systemu. Przedstawiono zaimplementowane rozwiązania, mające na celu poprawę poziomu gry systemu.

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

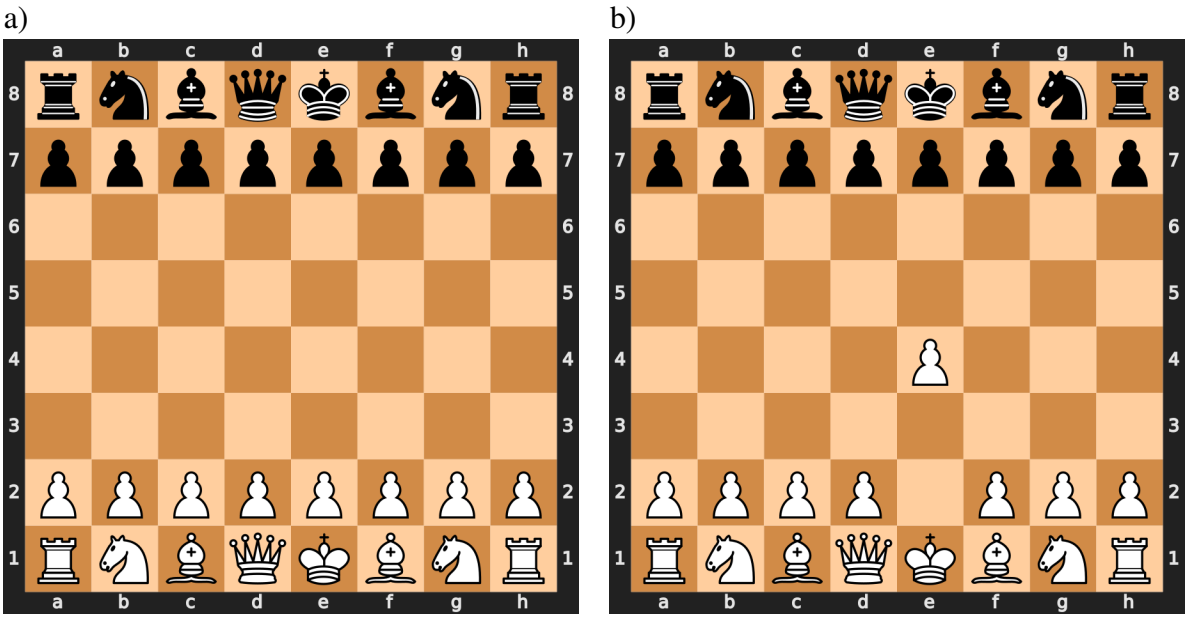
2.1. Komunikacja z systemem

2.1.1. Notacja Forsytha-Edwardsa Implemented

Aby umożliwić komunikację z programem, należy w pierwszej kolejności sprecyzować, w jakim formacie dostarczone zostaną dane wejściowe reprezentujące konkretną pozycję szachową. Standardem, wykorzystywanym nie tylko w większości silników, ale także w pojedynkach rozgrywanych online, jest Notacja Forsytha-Edwardsa (ang. *Forsyth–Edwards Notation*, w skrócie FEN).

FEN pozwala na jednoznaczne określenie pozycji na szachownicy. Przykład notacji dla pozycji startowej:

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: Przykładowce pozycje szachowe: a) startowa, b) po ruchu e2e4

Całość informacji, zawartych w jednej linii znaków kodowanych ASCII, ma 6 pól oddzielonych spacjami. Pola te informują, o różnych aspektach danej pozycji:

1. Reprezentacja 64 pól szachownicy. Każdy z wierszy szachownicy oddzielony jest „/”. Odpowiednio dla białych i czarnych figur.
2. Czyj jest ruch (w — białe, b — czarne)
3. Możliwości roszady (K/k — krótka roszada, Q/q — długa roszada)
4. Możliwości bicia w przelocie, znanego szerzej jako ruch en passant. Sprecyzowane pole jest polem atakowanym. W przypadku braku możliwości bicia w przelocie w polu widnieje „-”.
5. Liczba posunięć od ostatniego bicia bądź ruchu pionem. Istotna w regule 50 posunięć.
6. Liczba pełnych ruchów.

2.1.2. Szachowa Notacja Algebraiczna Implemented

2.1.3. Uniwersalny Interfejs Szachowy Implemented

2.2. Reprezentacja pozycji

2.2.1. Reprezentacja szachownicy Implemented

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.

biblatex

2.4. Algorytm wyszukiwania

W 1950 roku Amerykański matematyk Claude Shanon opublikował pracę naukową zatytułowaną ”Programowanie komputera do gry w szachy”[3]. Praca ta stała się teoretyczną podstawą tworzenia silników szachowych. Zawiera ona między innymi oszacowanie co do ilości możliwych pozycji szachowych, wynoszące 10^{43}. Choć oszacowanie to zmieniało się w pewnym stopniu na przestrzeni lat, to jednak liczba ta dowodzi jednoznacznie, że z uwagi na skalę problemu nie jest możliwa implementacja tablicy zawierającej wszystkie pozycje szachowe, oraz

najlepsze możliwe na nie odpowiedzi. Koniecznym było zaimplementowanie algorytmu, który dla konkretnej strategii, decydowałby jaki ruch wykonać uwzględniając daną głębokość drzewa decyzyjnego.

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

color

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 TODO

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] G. D. Greenwade. “The Comprehensive Tex Archive Network (CTAN)”. W: *TUGBoat* 14.3 (1993), s. 342–351.

[2] R. Huber i S. Meyer-Kahlen. *Description of the universal chess interface (UCI)*. 2006. URL: https://www.shredderchess.com/download/div/uci.zip (term. wiz. 18. 07. 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

To jest dodatek B