

DLA POPRAWNEGO FORMATOWANIA - STRONA DO USUNIĘCIA

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

# 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.

Efektom 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

<b>1. Wstęp</b>	<b>10</b>
1.1. Wprowadzenie	10
1.2. Cel i zakres	10
1.3. Układ pracy	11
<b>2. Implementacja silnika szachowego</b>	<b>12</b>
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	14
2.2.2. Reprezentacja stanu	14
2.2.3. Reprezentacja ruchu	14
2.3. Generowanie ruchów	14
2.3.1. Generowanie ruchów pseudolegalnych	14
2.3.2. Generowanie ruchów legalnych	14
2.4. Algorytm wyszukiwania	15
2.4.1. Algorytm min-max	15
2.4.2. Iteracyjne pogłębianie wyszukiwania	15
2.4.3. Zarządzanie czasem	15
2.5. Ocena heurystyczna	15
2.5.1. Heurystyka stanu gry	15
2.5.2. Heurystyka liczebności bierek	15
<b>3. Ulepszenia dla silnika szachowego</b>	<b>16</b>
3.1. Ulepszenia dla wyszukiwania	16
3.1.1. Biblioteka otwarć	16
3.1.2. Alfa-Beta cięcie	16
3.1.3. Ewaluacja cichych stanów	16
3.1.4. Sortowanie ruchów	16
3.1.5. Tabela transpozycji	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
<b>4. Ocena siły silnika</b>		<b>18</b>
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
<b>5. Zakończenie</b>		<b>19</b>
5.1. Podsumowanie pracy	TODO	19
5.2. Możliwości dalszego rozwoju aplikacji	TODO	19
<b>Bibliografia</b>		<b>20</b>
<b>A. Instrukcja wdrożenia</b>	TODO	<b>21</b>
<b>B. Przykład użycia UCI</b>	TODO	<b>22</b>

# Spis rysunków

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

# Spis tabel

2.1. UCI - komunikacja GUI do silnika . . . . .	14
2.2. UCI - komunikacja silnika do GUI . . . . .	14

# 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, powszechnie nazywane grą królewską, to najpopularniejsza forma intelektualnej rozrywki w dziejach ludzkości. Niekwestionowaną popularność tak wśród profesjonalistów, jak i amatorów zawdzięczają połączeniu prostoty zasad i złożoności strategicznych wyzwań. Ich historia, sięgająca VI wieku n.e., obejmuje stale ponawiane próby udoskonalania reguł i odkrywania nowych, coraz bardziej zaawansowanych, taktyk mających zagwarantować zwycięstwo w rozgrywce. 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 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 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 sieci neuronowe, zrewolucjonizowało sposób, w jaki rozumiemy tę grę 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 interfejsem graficznym.
- 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. Choć do zrozumienia pracy konieczna jest znajomość zasad gry w szachy, to w tym rozdziale mówiono także niektóre, bardziej zawile bądź mniej znane, jej aspekty.

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 **Written**

W 1950 roku amerykański matematyk Claude Shannon na łamach "Philosophical Magazine" opublikował pracę naukową zatytułowaną "Programowanie komputera do gry w szachy". [2] 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 określona jest skrótem pochodzącym od ich angielskich nazw:

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

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

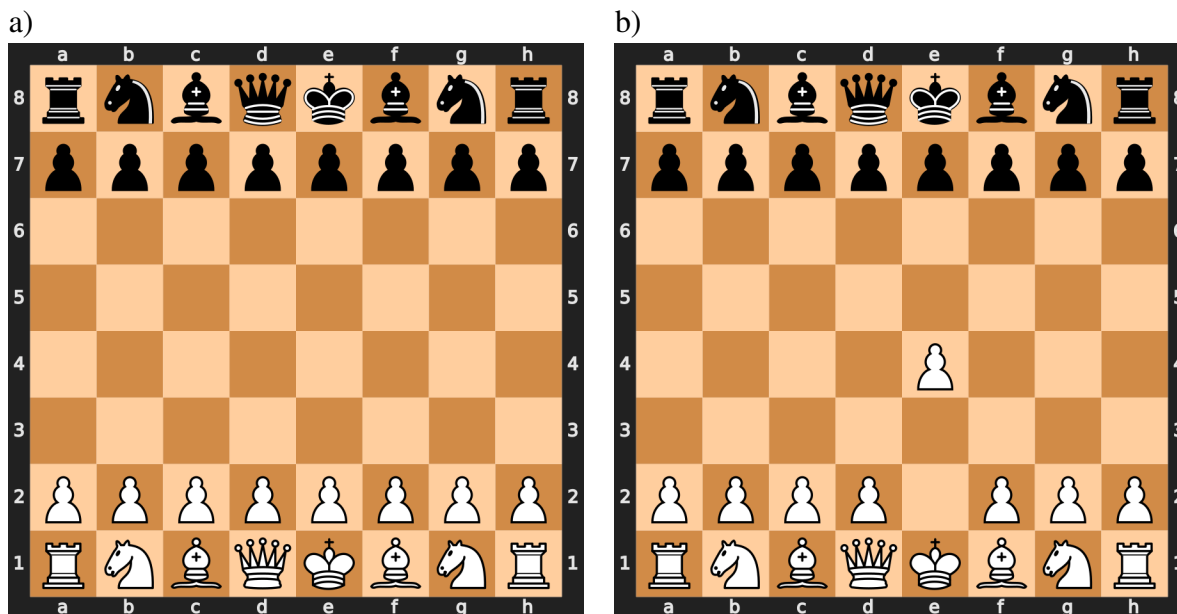
1. Reprezentacja 64 pól szachownicy z perspektywy białego gracza. Każdy z wierszy planszy oddzielony jest "/", a jego zawartość opisywana zostaje od kolumny a, do kolumny h. Liczbę nieprzerwanie pustych pól w danym wierszu określa cyfra z zakresu od 1 do 8.
2. Sprecyzowanie, do którego gracza należy 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 bierok.

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

Poniżej przedstawiono przykładowe FEN, wraz z korespondującymi im ułożeniami szachownicy:

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 **Written**

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.

Roszczy oznaczano przez pola ruchu króla, natomiast do ruchów z promocją dopisano literę określającą rodzaj podmienionej figury.

### 2.1.3. Uniwersalny Interfejs Szachowy **In progress**

Uniwersalny Interfejs Szachowy (ang. *Universal Chess Interface*, w skrócie UCI) 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 wystarczyły do rozegrania podstawowej partii mierzonej czasowo.

Tab. 2.1: UCI - komunikacja GUI do silnika

Komenda	Opis działania
uci	Wy tłumaczenie działania komendy
debug	Wy tłumaczenie działania komendy
isready	Wy tłumaczenie działania komendy

Tab. 2.2: UCI - komunikacja silnika do GUI

Komenda	Opis działania
id	Wy tłumaczenie działania komendy
uciok	Wy tłumaczenie działania komendy
readyok	Wy tłumaczenie działania komendy

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

## 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.

## **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 są 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 rzędzie, 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](https://www.thechessdrum.net/PGN_Reference.txt) (term. wiz. 04.09.2024).
- [2] 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