Dokumentacja wstępna

Temat

Szachy z wykorzystaniem algorytmu MiniMax

Rozwinięcie tematu i opis

Tematem projektu jest implementacja szachów a dokładniej aplikacji komputerowej umożliwiającej grę w szachy w kilku trybach: użytkownik kontra użytkowniki, użytkownik kontra komputer. Aplikacja powinna posiadać interfejs graficzny prezentujący plansze oraz figury, a także podstawową komunikację z użytkownikiem prezentującą wynik rozgrywki, czy też niepoprawnie wykonany ruch. Silnik szachowy, który będzie sterował komputerem w trybie użytkownik kontra komputer będzie oparty na algorytmie MiniMax, funkcji ewaluacyjnej oraz tabeli transpozycji.

Lista funkcjonalności:

- 1 Etap - implementacja zasad gry w szachy (użytkownik na przemian wykonuje ruchy białymi i czarnymi figurami)
    - Szachownica, na której można wykonywać tylko poprawne ruchy (narazie gracz rusza się białymi i czarnymi)
    - Generator ruchów
    - Graficzna reprezentacja rozgrywki
- 2 Etap - implementacja algorytmu MiniMax do gry w szachy
    - Funkcja ewaluacyjna
    - MiniMax
    - Optymalizacja wyszukiwania najlepszego ruchu

Zadania:

Zadanie	Czas
Projekt aplikacji (utworzenie dokumentacji oraz szkieletu - powiązań między klasami)	20h
Przygotowanie środowiska - budowanie, testy, formater, repozytorium, CI/CD.	15h
GUI konsolowe	3h
GUI przy pomocy biblioteki SFML	20h
Reprezentacja szachownicy	5h
Generator wszystkich możliwych ruchów	5h
Generator pseudo legalnych ruchów	6h
Mechanizm roszady, en passant	3h
Zasada 50 ruchów, Zasada 3 powtórzeń	3h
Reprezentacja rozgrywki	3h
Funkcja ewaluacyjna	8h
Klasa wyszukująca właściwy ruch w tym:	15h
● Algorytm MiniMax	5h
● Sortowanie ruchów	5h
● Mechanizm Quiescence search	5h
Suma	106h

- **Projekt aplikacji:**
  - Dla całej aplikacji podzielić projekt na moduły.
  - Zaprojektować hierarchię klas, zapisać wszystkie potrzebne pola i metody.
  - Narysować diagram klas prezentujący powiązania między klasami.
- **GUI:**
  - Do testowania w początkowej fazie projektu będziemy korzystać z prostego GUI w konsoli.

    - Następnie stworzymy GUI za pomocą prostego silnika do gier 2d SFML.

- **Reprezentacja szachownicy, figur, ruchów oraz rozgrywki:**
    - figury
    - pole szachownicy
    - szachownica
    - czyj ruch
    - ruch
    - en passant
    - możliwe roszady
    - Zasada 50 ruchów
    - Zasada 3 powtórzeń
    - pozycja
    - **rozgrywka (zawiera wszystkie pozostałe)**

- **Generator ruchów:**
    - “Promienie” (od ang ray) - Funkcja obliczająca na początku programu wszystkie możliwe ruchy dla każdej figury na każdym polu (zakładając że tylko ta jedna figura znajduje się szachownicy). Umożliwia efektywne generowanie pseudo legalnych ruchów (ruchy które nie uwzględniają ruchów które zostawiają króla w szachu). Ruchy które zostawiają króla w szachu są odrzucane po wykryciu ruchu, który zbija króla w następnym ruchu (generuje się listę pseudo legalnych ruchów dla białych, a następnie dla czarnych, jesli czarny ma możliwość zbicia króla, to odpowiadający ruch białych jest usuwany z listy).

    - “Pseudo-legalne ruchy” - Funkcja generująca wszystkie pseudo legalne ruchy dla danej pozycji. Trzeba przeiterować się przez wszystkie figury na planszy i dla każdej figury przeiterować się po promieniach (patrz poprzedni podpunkt).

    - Do weryfikacji legalności roszad trzeba przetrzymywać mapę atakowanych pól, która będzie obliczana przy generowaniu ruchów.

- **Funkcja ewaluacyjna:**

    - Na początek wystarczy funkcja licząca “material balance” (król 10000, królowa 900, wieża 500, laufer 325, koń 300, pion 100).
    - Potem można rozdzielić funkcję ewaluacyjną na 3 etapy - opening (kontrola środka, baza danych otwarć szachowych), middle game (Wyprowadzenie gońców i koni oraz roszada), endgame (aktywizacja króla i pchanie pionów, zwłaszcza “passed pawns”, czyli piony które nie mogą zostać zbite przez piony przeciwnika).
    - Mapy pól, na których figury są lepiej punktowane (piony i konie środek, laufry na przekątnych itd itp).

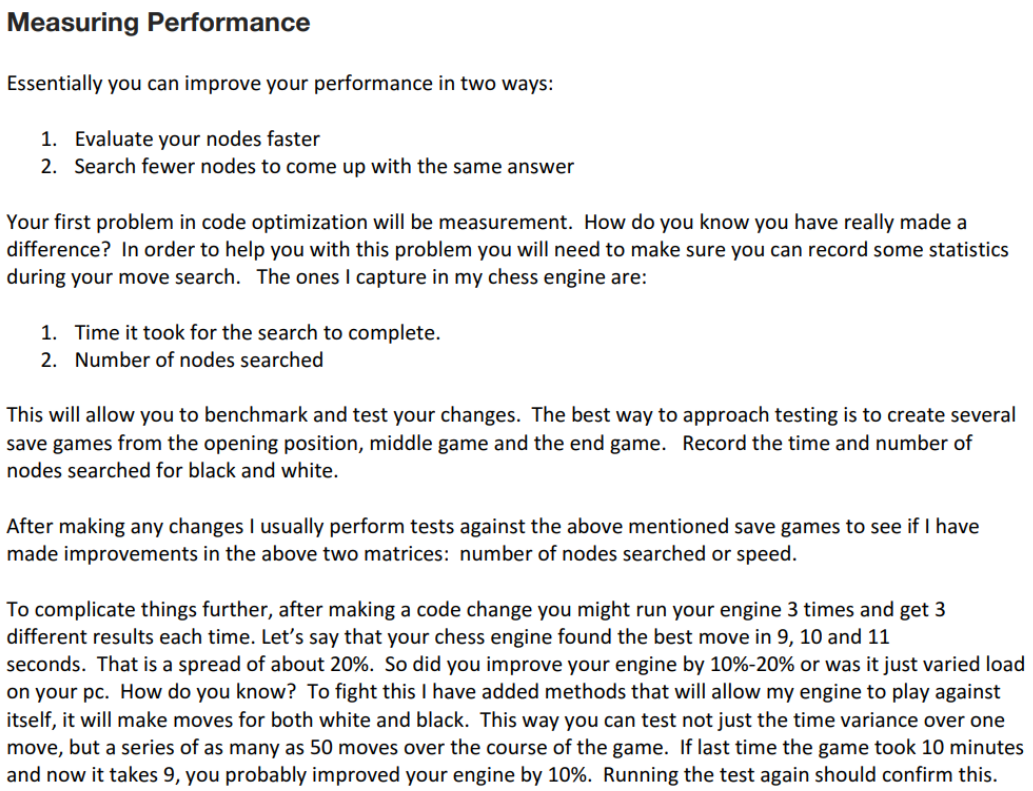
- **Algorytm MiniMax z przycinaniem Alpha Beta do znajdowania najlepszego ruchu:**
    - Sortowanie ruchów może usprawnić algorytm (Najprościej captures first https://www.chessprogramming.org/Move_Ordering).
    - Przykładowa implementacja algorytmu będzie dostępna w internecie.

- **Quiescence search:**
    - Ważne rozszerzenie algorytmu MiniMax.
    - po minimaxie do ustalonej głębokości trzeba dotrzeć do “cichej pozycji”, czyli pozycji, w której nie ma oczywistych bić prowadzących do drastycznej zmiany ewaluacji (np zbicie królowej).

- **Tabela transpozycji do usprawnienia MiniMaxa:**
    - Polega na zachowywaniu wcześniej wyszukiwanych pozycji i ich ewaluacji.
    - Należy zmodyfikować minimaxa, żeby najpierw patrzył czy dana pozycja była już oceniana.
    - Trzeba zaimplementować sposób haszowania pozycji (technika zoobrist hashing).

- **Przygotowanie środowiska projektu:**
    - Cmake:
        - Silnik gier 2d SFML https://www.sfml-dev.org/tutorials/2.5/compile-with-cmake.php
        - Cross platform
    - Testy
        - Będziemy wykorzystywać bibliotekę Catch2

        - “Perft” - testowanie czy generator ruchów wskazał prawidłową liczbę legalnych ruchów dla danej pozycji i jak szybko to zrobił
        - Wskazówki do testowania silnika szachowego:



Measuring Performance

Essentially you can improve your performance in two ways:

1. Evaluate your nodes faster
2. Search fewer nodes to come up with the same answer

Your first problem in code optimization will be measurement. How do you know you have really made a difference? In order to help you with this problem you will need to make sure you can record some statistics during your move search. The ones I capture in my chess engine are:

1. Time it took for the search to complete.
2. Number of nodes searched

This will allow you to benchmark and test your changes. The best way to approach testing is to create several save games from the opening position, middle game and the end game. Record the time and number of nodes searched for black and white.

After making any changes I usually perform tests against the above mentioned save games to see if I have made improvements in the above two matrices: number of nodes searched or speed.

To complicate things further, after making a code change you might run your engine 3 times and get 3 different results each time. Let’s say that your chess engine found the best move in 9, 10 and 11 seconds. That is a spread of about 20%. So did you improve your engine by 10%-20% or was it just varied load on your pc. How do you know? To fight this I have added methods that will allow my engine to play against itself, it will make moves for both white and black. This way you can test not just the time variance over one move, but a series of as many as 50 moves over the course of the game. If last time the game took 10 minutes and now it takes 9, you probably improved your engine by 10%. Running the test again should confirm this.

    - CI/CD
        - Automatyczne budowanie i testowanie
    - Clang-format
    - Valgrind

Ewentualne plany na dalszy rozwój projektu:

- Moduł sieci do rozgrywek online
- Timery do rozgrywki na czas
- Tryby gry do wyboru w GUI do gry przeciwko AI, sandbox z szachownicą albo z innymi graczami online
- System rankingowy
- Analiza rozgrywek
- Integracja innych silników szachowych z GUI lub naszego silnika z innymi GUI