Dokumentace

Šachová umělá inteligence

David Tesař, C3a

Obor: Informační technologie

# Anotace

Tato práce se zabývá tvorbou šachového enginu s použitím technik umělé inteligence v jazyce C#. Projekt obsahuje také grafické uživatelské rozhraní vytvořené v prostředí Unity, které umožňuje přímou interakci s AI, analýzu pozic i hru bez umělé inteligence. Cílem bylo vytvořit samostatně fungujícího šachového bota s optimalizacemi pro výkon a pohodlné ovládání.

# Úvod

V této práci jsem se rozhodl vytvořit vlastní šachovou umělou inteligenci od základů. Inspirací mi byly existující open-source enginy jako Stockfish, ale mým cílem bylo porozumět tomu, jak šachové AI skutečně fungují na nízké úrovni, a následně vytvořit vlastní implementaci v jazyce C#.   
  
Jádro šachového enginu využívá tzv. bitboardy pro efektivní reprezentaci šachovnice. Ty prospívají zejména generování legálních tahů. Hledání nejlepšího tahu zajišťuje algoritmus minimax s alpha-beta pruningem, který byl dále optimalizován pomocí move orderingu a iterative deepening (postupné prohlubování).   
  
Aby byla AI přístupná i běžnému uživateli, vytvořil jsem grafické rozhraní v Unity. Aplikace nabízí několik režimů: klasickou hru proti AI, možnost přepínat strany, volnou hru bez AI a analyzátor pozic.

# Ekonomická rozvaha

**Konkurence**

Na poli šachových enginů existuje několik velmi výkonných a zavedených projektů, mezi nejznámější patří například **Stockfish**, **Leela Chess Zero** nebo **Komodo**. Tyto enginy používají pokročilé algoritmy a strojové učení, ale jejich kód je často složitý, nepřehledný a náročný na pochopení pro začínající vývojáře nebo studenty.

**Výhody mého řešení**

Hlavním cílem tohoto projektu bylo vytvořit šachovou AI, která bude:

**Způsob propagace**

Projekt byl nahrán na veřejný repozitář **GitHub**, kde je k dispozici zdarma.Byl ukázán spolužákům, kteří poskytli zpětnou vazbu

**Návratnost investic**

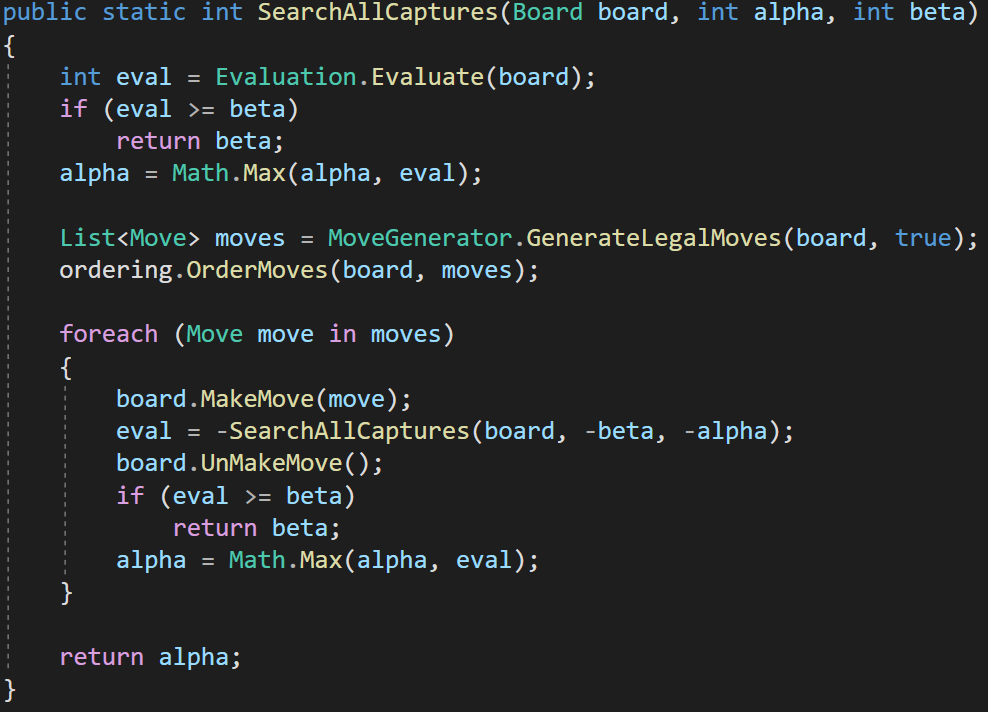
Přestože projekt nemá přímý komerční záměr, přináší následující přínosy:

* **Zkušenosti s tvorbou vlastního AI enginu** a optimalizacemi výkonu¨
* **Lze porovnávat sílu v šachách oproti lidem**
* **Případný další vývoj nebo použití jako výuková pomůcka**

# Vývoj

Projekt byl vyvíjen zpočátku jako konzolová aplikace. Jelikož je pro AI nejdůležitější rychlost prohledávání, tak bylo důležité, jaké technologie použít. Pro reprezentaci šachovnice jsou použity bitboardy. Ty využívají toho, že ulong v C# má 64 bitů a šachovnice má 64 polí. To se dá použít jednodušše tak, že tam kde je bit nastaven na 1, tak je na tomto poli figura. Přístup s bitboardy se může zdát na první pohled zvláštní, ale je doopravdy efektivní, jelikož bitové operace jsou pro počítač nenáročné. Ovšem při převedení do Unity je problém, že nepodporuje třídu BitOperations (System.Numerics), musel jsem pro to tedy napsat vlastní logiku.

Doporučuji se více zaměřit na optimalizaci generování tahů, jelikož to AI musí provádět neustále. Dají se použít technologie jako **Magic Bitboards.** Prohledávací algoritmus(mini-max) je take velmi důležité optimalizovat. Nejvíce ho zrychlí **alpha-beta pruning,** který ubere počet uzlů, které musí AI zpracovat. S tím se dá velmi dobře použít **move-ordering.** Paradoxně AI zrychlilo také **iterative-deepening**, které postupně zvyšuje hloubku prohledávání: Search(1), Search(2) … To sice program zpomaluje, jelikož musí dělat znovu výpočty, které už předtím dělal, ale pokud si budeme uchovávat nejlepší tah, který vyšel v předchozí hloubce, tak spolu s **move-orderingem**, se můžeme dívat na ty “zajímavější” tahy jako první. Také jsem se snažil implementovat **Transposition table**, která by také měla výrazně zrychlit Search, ale to se mi nezdařilo. **QuiescenceSearch** byl použit, pokud AI dorazí na nedostatečnou hloubku a je tu příliš velké riziko změny hned po následujícím tahu.

Zde prohledává všechny brací tahy, dokud nenarazí na “tichý tah”

# Testování

Během vývoje bylo provedeno několik testovacích scénářů, které ověřují správnost funkcionalit, chování AI a samotné spuštění aplikace. Níže jsou uvedeny konkrétní scénáře a jejich výsledky.

**1. Spuštění aplikace (nasazení)**

* **Scénář:** Stažení buildu z GitHubu, spuštění hry.
* **Výsledek:** Aplikace se úspěšně spustí, hlavní menu funguje, všechny tlačítka vedou na správné scény. Hra proti AI i analýza funguje bez chyb.

**2. Test generování tahů (MoveGeneration)**

* **Scénář:** Test pozice s více než 30 možnými tahy, např. zahájení hry (počáteční postavení).
* **Výsledek:** Všechny legální tahy byly správně vygenerovány, žádný tah nebyl chybně označen jako legální nebo nelegální.

**3. Test jednoduchého nejlepšího tahu**

* **Scénář:** V pozici, kde je možné vzít soupeřovu dámu nechráněnou věží, by měl engine zvolit tento tah.
* **Výsledek:** AI správně detekovala nejlepší tah a provedla ho, evaluace odpovídala očekávané materiální výhodě (např. +9).

**4. Testování generování tahů do hloubky „Shannon number“**

* **Scénář:** Do algoritmu se zadá hloubka a vrátí se počet možných pozic v hloubce.
* **Výsledek:**  Algoritmus vypsal správné číslo pro každou pozici a hloubku.

**5. Test analýzy pozice**

* **Scénář:** V režimu „Analýza“ se provede několik tahů a poté se stiskne tlačítko „Nejlepší tah“.
* **Výsledek:** AI vždy zobrazila správný tah s odpovídající evaluací. Testovány i vyrovnané pozice i pozice s jasnou převahou jedné strany.

# Závěr

Celkově hodnotím projekt velmi úspěšně. AI hraje dobře i proti pokročilým hráčům. Samozřejmě má jasné slabiny, například pokud se jedná o techniku v koncovce, nebo bezpečnost krále. Nejlépe hraje v komplikovaných pozicích, kde lidského hráče dokáže přepočítat, a zároveň nikdy neudělá hrubou chybu. Rychlost je také velice slibná, i když se dá hodně zlepšit.

Výsledkem je funkční UI a rozšířitelný šachový engine.