Malakh - technická dokumentácia

Generované programom Doxygen 1.10.0

Kapitola 1

Malakh

Tento projekt je šachový engine, ktorý je schopný hrať s neštandardnými šachovými pravidlami. Tento engine sa dá ovládať pomocou UCI príkazov.

1.1 AI režimy

Tento engine má implementované nasledujúce AI režimy:

- **basic** - využíva staticku evaluačnú funkciu na evaluáciu šachovej pozície
- **ensemble** - využíva súborové učenie na evaluáciu šachovej pozície. Modely súborového učenia boli vytrénované v skripte `Malakh Trainer`.

1.2 UCI príkazy

1.2.1 Engine -> GUI

Zoznam štandardných UCI príkazov:

- **id** - pomocou tohto príkazu sa engine identifikuje po prijatí uci príkazu
  - **id name [name]** - engine odošle enginu svoje meno (id name Malakh)
  - **id author [author]** - engine odošle enginu svojho autora (id author Martin Svab)
- **option name [name] type [type] ...** - pomocou tohto príkazu engine odošle nastaviteľnú konfiguráciu po prijatí uci príkazu (napr. option name WhitePawn type combo default Classic var Classic var Red var Blue)
- **uciok** - po odoslaní identifikácie, konfigurovateľných nastavení a mobilít sa odošle tento príkaz
- **readyok** - tento príkaz je odoslaný ako odpoveď na 'isready' správu
- **bestmove [bestmove]** - tento príkaz slúži ako odpoveď na 'go' správu. Po ukončení prehľadávania sa odošle najlepší možný ťah hráča. (napr. bestmove d2d4_Hd3)

Zoznam dodatočných UCI príkazov:

- **mobility [piece] [essence] [type] [start_x] [start_y] [direction_x] [direction_y] [limit]** - týmito príkazmi sa odošle zoznam mobilít konkrétnej figúrky s určitou esenciou (napr. mobility Pawn Classic Move 0 1 0 0 1).
- **legalmoves [legalmove1] [legalmove2] ...** - týmto príkazom sa odošle zoznam legálnych pohybov hráča, ktorý je aktuálne na ťahu (napr. legalmoves a7a6 a7a5_Ha6 b7b6 b7b5_Hb6 ...)
- **check** - po príkaze 'position' ak v novej pozícii je kráľ napadnutý, tak engine na to upozorní GUI pomocou tohto príkazu
- **result [result]** - po príkaze 'legalmoves' a po príkaze 'go' ak legálne pohyby neexistujú tak engine odošle GUI výsledok hry týmto príkazom
  - **result White** - hra bola ukončená výhrou bieleho hráča
  - **result Black** - hra bola ukončená výhrou čierneho hráča
  - **result Stalemate** - hra bola ukončená remízou

1.2.2 GUI -> Engine

Zoznam štandardných UCI príkazov:

- **uci** - engine sa prepne do UCI režimu a na túto správu odpovie svojou identifikáciou, odoslaním konfigurovateľných nastavení a mobilít figúrok. Po odoslaní všetkých potrebných informácii enginu odošle 'uciok' správu.
- **isready** - pred prvým prehľadávaním pozície je nutné poslať tento príkaz aby si Engine inicializoval hodnoty ako zobrist kľúče. Engine po dokončení inicializácie odpovie správou 'readyok'.
- **setoption name [name] value [value]** - pomocou tohto príkazu sa zmení konfigurácia enginu. Aktuálne sa tento príkaz využíva pred 'ucinewgame' príkazom na nastavenie esencií figúrok (napr. setoption name WhitePawn value Red)
- **ucinewgame** - pre spustenie novej hry sa použije tento príkaz. Engine spustí novú hru s esenciami, ktoré boli predtým nastavené pomocou 'setoption' príkazov.
- **position [fen | startpos | curpos] moves ...** - týmto príkazom môžeme nastaviť aktuálnu pozíciu v hre (napr. position curpos moves e2e4). Štandardný protokol UCI nepodporuje pozíciu 'curpos'. Tento príkaz bol pre účely tejto implementácie modifikovaný. V aktuálnej verzii implementácia nepodporuje fen stringy.
- **go** - týmto príkazom povieme enginu aby vyhľadal najlepší ťah v aktuálnej pozícii, ktorá bola nastavená príkazom 'position'. V aktuálnej verzii je podporovaný tento príkaz len s parametrom 'depth' (napr. go depth 2).
- **quit** - engine sa po prijatí tejto správy vypne

Zoznam dodatočných UCI príkazov:

- **legalmoves** - týmto príkazom si GUI vypýta od enginu zoznam legálnych pohybov hráča, ktorý je aktuálne na ťahu. Engine na to odpovie vlastnou 'legalmoves' správou.

Kapitola 2

Register tried

2.1 Zoznam tried

Nasledujúci zoznam obsahuje predovšetkým identifikáciu tried, ale nachádzajú sa tu i ďalšie netriviálne prvky, ako sú štruktúry (struct), uniony (union) a rozhrania (interface). V zozname sú uvedené ich stručné popisy:

Kapitola 3

Register súborov

3.1 Zoznam súborov

Tu nájdete zoznam všetkých dokumentovaných súborov so stručnými popismi:

Kapitola 4

Dokumentácia tried

4.1 Dokumentácia štruktúry BitBoard

`#include <bitboard.h>`

Verejné metódy

- unsigned long long getBit (char x, char y)
- void setBit (char x, char y)
- void clearBit (char x, char y)
- void printBits ()
- bool getKingAttack (char x, char y)
- char getTropism (char x, char y)

Verejné atribúty

- unsigned long long value = 0

Privátne metódy

- unsigned long long getMask (char x, char y)
- unsigned long long getKingPattern (unsigned long long value)
- unsigned long long getTropismPattern (unsigned long long value)

4.1.1 Detailný popis

Táto štruktúra reprezentuje určitú vlastnosť šachovnice pomocou 64-bitového čísla. Každý bit reprezentuje určitú pozíciu na šachovnici.

4.1.2 Dokumentácia k metódam

4.1.2.1 clearBit()

```
void BitBoard::clearBit (
            char x,
            char y )
```

Pre požadovanú pozíciu na šachovnici táto funkcia nastaví jej bit na hodnotu 0.

Parametre

x	Koordinát X požadovanej pozície.
y	Koordinát y požadovanej pozície.

4.1.2.2 getBit()

```
unsigned long long BitBoard::getBit (
            char x,
            char y )
```

Pre požadovanú pozíciu na šachovnici táto funkcia vráti jej bit.

Parametre

x	Koordinát X požadovanej pozície.
y	Koordinát y požadovanej pozície.

4.1.2.3 getKingAttack()

```
bool BitBoard::getKingAttack (
            char x,
            char y )
```

Táto funkcia zistí či požadovaná pozícia je v blízkosti bitu 1.

Parametre

x	Koordinát X požadovanej pozície.
y	Koordinát y požadovanej pozície.

4.1.2.4 getKingPattern()

```
unsigned long long BitBoard::getKingPattern (
            unsigned long long value )  [private]
```

Táto funkcia vytvorí masku s paternou kráľa - všetky bity 1 sú rozšírené o ich štvorcové okolie.

Parametre

value	Hodnota bitboardu z ktorého generujeme paternu.

4.1.2.5 getMask()

```
unsigned long long BitBoard::getMask (
```

```
            char x,
            char y )  [private]
```

Táto funkcia vytvorí masku potrebnú na získanie bitu pre požadovanú pozíciu na šachovnici.

Parametre

x	Koordinát X požadovanej pozície.
y	Koordinát y požadovanej pozície.

4.1.2.6 getTropism()

```
char BitBoard::getTropism (
            char x,
            char y )
```

Táto funkcia vypočíta vzdialenosť medzi požadovanou pozíciou na šachovnici a najbližším bitom 1 na bitboarde.

Parametre

x	Koordinát X požadovanej pozície.
y	Koordinát y požadovanej pozície.

4.1.2.7 getTropismPattern()

```
unsigned long long BitBoard::getTropismPattern (
            unsigned long long value )  [private]
```

Táto funkcia vytvorí masku s paternou kráľa - všetky bity 1 sú rozšírené o ich krížové okolie.

Parametre

value	Hodnota bitboardu z ktorého generujeme paternu.

4.1.2.8 printBits()

```
void BitBoard::printBits ( )
```

Táto funkcia vypíše jednotlivé bity bitboardu.

Parametre

x	Koordinát X požadovanej pozície.
y	Koordinát y požadovanej pozície.

4.1.2.9 setBit()

```
void BitBoard::setBit (
            char x,
            char y )
```

Pre požadovanú pozíciu na šachovnici táto funkcia nastaví jej bit na hodnotu 1.

Parametre

x	Koordinát X požadovanej pozície.
y	Koordinát y požadovanej pozície.

4.1.3 Dokumentácia k dátovým členom

4.1.3.1 value

```
unsigned long long BitBoard::value = 0
```

64-bitové číslo reprezentujúce určitú vlastnosť na šachovnici.

Dokumentácia pre túto štruktúru (struct) bola generovaná z nasledujúcich súborov:

- bitboard.h
- bitboard.cpp

4.2 Dokumentácia triedy Board

```
#include <board.h>
```

Verejné metódy

- void initBoard (EssenceArgs essenceArgs)
- void initBoard (EssenceArgs essenceArgs, BitBoard pieces[12], PieceColor curTurn)
- void printBoard ()
- void printMoves ()
- int evalBoard (PieceColor color)
- bool isQuiet ()
- bool makeMove (char x1, char y1, char x2, char y2)
- bool makeMove (char x1, char y1, char x2, char y2, PieceType promotion)
- bool makeMove (LegalMove move)
- std::pair< bool, Piece > getPiece (char x, char y)
- std::vector< LegalMove > getLegalMoves ()
- std::string toString ()
- GameResult getResult ()

Verejné atribúty

- PieceEssence essenceConfig [2 ∗6] = {}
- int essenceCounts [3]
- BitBoard pieces [2 ∗6]
- BitBoard colors [2]
- BitBoard attacks [2]
- BitBoard allPieces
- BitBoard notMoved
- Ghost ghost
- int pieceCounts [2 ∗6] = {}
- int inputArray [2 ∗8 ∗8]
- std::vector< LegalMove > moves [2]
- bool movesValidated = false
- PieceColor curTurn = White
- EvalArgs eval {}
- BoardHash hash {}
- int halfmoveClock = 0
- int fullmoveClock = 1

Privátne metódy

- void setEssenceConfig (EssenceArgs essenceArgs)
- void clearBoard ()
- void addPiece (PieceColor color, PieceType type, char x, char y, bool isNew)
- std::pair< bool, Piece > removePiece (char x, char y)
- void refreshAggregations ()
- void setInputArrayPiece (char pieceIndex, char x, char y)
- void setInputArrayColor (PieceColor color)

4.2.1 Detailný popis

Táto štruktúra reprezentuje šachovnicu so všetkými relevantnými informáciami.

4.2.2 Dokumentácia k metódam

4.2.2.1 addPiece()

```
void Board::addPiece (
            PieceColor color,
            PieceType type,
            char x,
            char y,
            bool isNew )  [private]
```

Pridá figúrku na šachovnicu a aktualizuje relevantné evaluačné argumenty.

Parametre

color	Vlastník novej figúrky.
type	Typ novej figúrky.
x	Koordinát X novej figúrky.
y	Koordinát Y novej figúrky.
isNew	Ak je figúrka nová, tak sa nastaví aj notMoved bitboard.

4.2.2.2 clearBoard()

```
void Board::clearBoard ( )  [private]
```

Odstráni všetky figúrky zo šachovnice a resetuje evaluačné argumenty.

4.2.2.3 evalBoard()

```
int Board::evalBoard (
            PieceColor color )
```

Táto funkcia vypočíta číselnú hodnotu aktuálnej pozície podľa daného hráča.

Parametre

color	Hráč pre ktorého je vypočítaná hodnota pozície.

Návratová hodnota

Hodnota pozície.

4.2.2.4 getLegalMoves()

```
std::vector< LegalMove > Board::getLegalMoves ( )
```

Vráti legálne pohyby aktuálneho hráča. Vykoná validáciu pohybov ak ešte nebola vykonaná.

Návratová hodnota

Zoznam legálnych pohybov hráča, ktorý je na rade.

4.2.2.5 getPiece()

```
std::pair< bool, Piece > Board::getPiece (
            char x,
            char y )
```

Nájde sa figúrka na konkrétnej pozícii na šachovnici.

Parametre

x	Koordinát X danej pozície.
y	Koordinát Y danej pozície.

Návratová hodnota

Vráti pár bitu reprezentujúci úšpešné/neúspešné nájdenie figúrky a nájdenú figúrku.

4.2.2.6 getResult()

```
GameResult Board::getResult ( )
```

Vráti aktuálny výsledok hry.

Návratová hodnota

Enumerátor reprezentujúci aktuálny výsledok hry.

4.2.2.7 initBoard() [1/2]

```
void Board::initBoard (
            EssenceArgs essenceArgs )
```

Inicializácia počiatočného stavu šachovnice.

Parametre

essenceArgs	Herná konfigurácia esencií.

4.2.2.8 initBoard() [2/2]

```
void Board::initBoard (
            EssenceArgs essenceArgs,
            BitBoard pieces[12],
            PieceColor curTurn )
```

Inicializáccia šachovnice podľa argumentov.

Parametre

essenceArgs	Herná konfigurácia esencií.
pieces	Bitboardy reprezentujúce konkrétny typ figúrok.
curTurn	Aktuálny hráč na rade.

4.2.2.9 isQuiet()

```
bool Board::isQuiet ( )
```

Overenie či je daná pozícia považovaná za tichú.

Návratová hodnota

Vráti 1 ak je pozícia tichá, 0 ak pozícia nie je tichá.

4.2.2.10 makeMove() [1/3]

```
bool Board::makeMove (
            char x1,
            char y1,
            char x2,
            char y2 )
```

Vykoná sa pohyb figúrky bez nastavenej promócie pešiaka (len ak je pohyb legálny).

Parametre

x1	Koordinát X zdrojovej pozície.
y1	Koordinát Y zdrojovej pozície.
x2	Koordinát X cieľovej pozície.
y2	Koordinát Y cieľovej pozície.

Návratová hodnota

Bit reprezentujúci či bol pohyb úspešne vykonaný.

4.2.2.11 makeMove() [2/3]

```
bool Board::makeMove (
            char x1,
            char y1,
            char x2,
            char y2,
            PieceType promotion )
```

Vykoná sa pohyb figúrky s nastevenou promóciou pešiaka (len ak je pohyb legálny).

Parametre

x1	Koordinát X zdrojovej pozície.
y1	Koordinát Y zdrojovej pozície.
x2	Koordinát X cieľovej pozície.
y2	Koordinát Y cieľovej pozície.
promotion	Typ figúrky na ktorý sa presunutý pešiak zmení.

Návratová hodnota

Bit reprezentujúci či bol pohyb úspešne vykonaný.

4.2.2.12 makeMove() [3/3]

```
bool Board::makeMove (
            LegalMove move )
```

Vykoná sa konkrétny pseudolegálny pohyb figúrky.

Parametre

move	Vykonávaný pseudolegálny pohyb.

Návratová hodnota

Vráti 1 ak je vykonaný pohyb legálny, vráti 0 ak je vykonaný pohyb pseudolegálny.

4.2.2.13 printBoard()

```
void Board::printBoard ( )
```

Táto funkcia vypíše umiestnenie figúrok na šachovnicí.

4.2.2.14 printMoves()

```
void Board::printMoves ( )
```

Táto funkcia vypíše dostupné pohyby figúrok.

4.2.2.15 refreshAggregations()

```
void Board::refreshAggregations ( )  [private]
```

Aktualizuje agregácie bitboardov - colors, allPieces.

4.2.2.16 removePiece()

```
std::pair< bool, Piece > Board::removePiece (
            char x,
            char y )  [private]
```

Odstráni zo šachovnice figúrku a aktualizuje relevantné evaluačné argumenty.

Parametre

x	Koordinát X odstránenej figúrky.
y	Koordinát Y odstránenej figúrky.

Návratová hodnota

Vráti pár bitu, ktorý reprezentuje úspešné/neúspešné odstránenie figúrky a odstránenú figúrku.

4.2.2.17 setEssenceConfig()

```
void Board::setEssenceConfig (
```

```
            EssenceArgs essenceArgs )  [private]
```

Nastaví konfigurácia esencií na šachovnici.

Parametre

essenceArgs	Nová konfigurácia esencií.

4.2.2.18 setInputArrayColor()

```
void Board::setInputArrayColor (
            PieceColor color )  [private]
```

Nastaví farbu hráča na druhom kanáli vstupnej matice pre CNN.

Parametre

color	Farba hráča na rade

4.2.2.19 setInputArrayPiece()

```
void Board::setInputArrayPiece (
            char pieceIndex,
            char x,
            char y )  [private]
```

Nastaví figúrku na prvom kanáli vstupnej matice pre CNN.

Parametre

pieceIndex	Identifikátor figúrky na danej pozícii.
x	Koordinát X danej pozície.
y	Koordinát Y danej pozície.

4.2.2.20 toString()

```
std::string Board::toString ( )
```

Zapíše našu šachovnicu do FEN stringu.

Návratová hodnota

FEN string reprezentujúci našu šachovnicu.

4.2.3 Dokumentácia k dátovým členom

4.2.3.1 allPieces

```
BitBoard Board::allPieces
```

Bitboard reprezentujúci všetky figúrky na šacchovnici.

4.2.3.2 attacks

```
BitBoard Board::attacks[2]
```

Bitboardy reprezentujúce napadnuté políčka konkrétneho hráča.

4.2.3.3 colors

```
BitBoard Board::colors[2]
```

Bitboardy reprezentujúce všetky figúrky konkrétneho hráča.

4.2.3.4 curTurn

```
PieceColor Board::curTurn = White
```

Aktuálny hráč na rade.

4.2.3.5 essenceConfig

```
PieceEssence Board::essenceConfig[2 *6] = {}
```

Konfigurácia esencií fúgurok.

4.2.3.6 essenceCounts

```
int Board::essenceCounts[3]
```

Počítadlo herných esencií.

4.2.3.7 eval

```
EvalArgs Board::eval {}
```

Priebežne aktualizované argumenty evaluačnej funkcie.

4.2.3.8 fullmoveClock

```
int Board::fullmoveClock = 1
```

Počítadlo všetkých plných pohybov. Inkrementované po každom pohybe čierneho hráča.

4.2.3.9 ghost

```
Ghost Board::ghost
```

Figúrka zanechaná po vykonaní špeciálneho pohybu pešiaka na začiatku hry.

4.2.3.10 halfmoveClock

```
int Board::halfmoveClock = 0
```

Počítadlo polovičných pohybov. Toto počítadlo sa resetuje po pohybu pešiaka alebo obsadení figúrky.

4.2.3.11 hash

```
BoardHash Board::hash {}
```

Hash šachovej pozície pomocou ktorého vieme identifikovať šachovnicu.

4.2.3.12 inputArray

```
int Board::inputArray[2 *8 *8]
```

Reprezentácia šachovnice pre vstup do modelu CNN.

4.2.3.13 moves

```
std::vector<LegalMove> Board::moves[2]
```

Zoznamy dostupných pohybov pre konkrétneho hráča.

4.2.3.14 movesValidated

```
bool Board::movesValidated = false
```

Bit pomocou ktorého si pamätáme či sme zvalidovali vygenerované pseudolegálne pohyby.

4.2.3.15 notMoved

```
BitBoard Board::notMoved
```

Bitboard reprezentujúci figúrky, ktoré sa ešte nepohli (pre En Passant a rošádu).

4.2.3.16 pieceCounts

```
int Board::pieceCounts[2 *6] = {}
```

Počítadla figúrok.

4.2.3.17 pieces

```
BitBoard Board::pieces[2 *6]
```

Bitboardy reprezentujúce konkrétny typ figúrok.

Dokumentácia pre túto triedu bola generovaná z nasledujúcich súborov:

- board.h
- board.cpp

4.3 Dokumentácia štruktúry BoardHash

```
#include <hashing.h>
```

Verejné metódy

- void switchSquare (Piece piece, char x, char y)
- void switchNotMoved (char x, char y)
- void switchGhost (Ghost ghost)
- void switchTurn ()

Verejné atribúty

- unsigned long long value

4.3.1 Detailný popis

Štruktúra reprezentujúca hash šachovnice.

4.3.2 Dokumentácia k metódam

4.3.2.1 switchGhost()

```
void BoardHash::switchGhost (
            Ghost ghost )
```

Touto funkciou upravíme hash šachovnice po pridaní alebo odstránení špeciálnej figúrky (vytvorená 'hasty' pohybom).

Parametre

ghost	Špeciálna figúrka.

4.3.2.2 switchNotMoved()

```
void BoardHash::switchNotMoved (
            char x,
            char y )
```

Touto funkciou upravíme hash šachovnice po pridaní alebo odstránení nepohnutej figúrky.

Parametre

x	Koordinát X zmeneného políčka.
y	Koordinát Y zmeneného políčka.

4.3.2.3 switchSquare()

```
void BoardHash::switchSquare (
            Piece piece,
            char x,
            char y )
```

Touto funkciou upravíme hash šachovnice po premiestnení figúrky z alebo do danej pozície.

Parametre

piece	Presunutá figúrka.
x	Koordinát X zmeneného políčka.
y	Koordinát Y zmeneného políčka.

4.3.2.4 switchTurn()

```
void BoardHash::switchTurn ( )
```

Touto funkciou upravíme hash šachovnice po zmenený aktívneho hráča.

4.3.3 Dokumentácia k dátovým členom

4.3.3.1 value

```
unsigned long long BoardHash::value
```

64-bitové číslo reprezentujúce identikikátor šachovnice.

Dokumentácia pre túto štruktúru (struct) bola generovaná z nasledujúcich súborov:

- hashing.h
- hashing.cpp

4.4 Dokumentácia štruktúry ConnectionString

`#include <database.h>`

Verejné metódy

- std::string toString ()

Verejné atribúty

- std::string **dbname**
- std::string user
- std::string password
- std::string hostaddr
- std::string port

4.4.1 Detailný popis

Trieda potrebná na pripojenie do databázy.

4.4.2 Dokumentácia k metódam

4.4.2.1 toString()

`std::string ConnectionString::toString ( )`

Z atribútov je vygenerovaný konekčný string.

Návratová hodnota

Konekčný string.

4.4.3 Dokumentácia k dátovým členom

4.4.3.1 hostaddr

`std::string ConnectionString::hostaddr`

Adresa hostiteľa databázy.

4.4.3.2 password

`std::string ConnectionString::password`

Heslo používateľa.

4.4.3.3 port

```
std::string ConnectionString::port
```

Port databázy.

4.4.3.4 user

```
std::string ConnectionString::user
```

Meno používateľa.

Dokumentácia pre túto štruktúru (struct) bola generovaná z nasledujúcich súborov:

- database.h
- database.cpp

4.5 Dokumentácia triedy DatabaseConnection

```
#include <database.h>
```

Verejné metódy

- DatabaseConnection ()
- int getIdEssenceConfig (EssenceArgs essenceArgs)
- void updateScores ()
- void addBoardResult (Board board, int idEssenceConfig, GameResult gameResult)

Privátne atribúty

- pqxx::connection connection
- pqxx::work txn

4.5.1 Detailný popis

Konkrétne pripojenie na databáze.

4.5.2 Dokumentácia konštruktoru a deštruktoru

4.5.2.1 DatabaseConnection()

```
DatabaseConnection::DatabaseConnection ( )
```

Konštruktor vytvorí spojenie pomocou konekčného stringu z triedy DatabaseManager.

4.5.3 Dokumentácia k metódam

4.5.3.1 addBoardResult()

```
void DatabaseConnection::addBoardResult (
            Board board,
            int idEssenceConfig,
            GameResult gameResult )
```

Do databázy uložíme výsledok hry pre danú šachovú pozíciu. Ak táto pozícia ešte v databáze neexistuje, tak vložíme nový záznam. Ak táto pozícia sa už v databáze nachádza, tak len inkrementujeme počítadlo výsledku.

Parametre

board	Uložená šachovnica.
idEssenceConfig	ID konfigurácie esencií.
gameResult	Výsledok hry.

4.5.3.2 getIdEssenceConfig()

```
int DatabaseConnection::getIdEssenceConfig (
            EssenceArgs essenceArgs )
```

Pre danú konfiguráciu esencií zistíme jej ID.

Parametre

essenceArgs	Konfigurácia esencií.

Návratová hodnota

ID konfigurácie esencií.

4.5.3.3 updateScores()

```
void DatabaseConnection::updateScores ( )
```

Pomocou tejto metódy môžeme upraviť skóre šachových pozícii uložených v databáze pri zmene evaluačnej funkcie.

4.5.4 Dokumentácia k dátovým členom

4.5.4.1 connection

```
pqxx::connection DatabaseConnection::connection  [private]
```

Pripojenie do databázy.

4.5.4.2 txn

```
pqxx::work DatabaseConnection::txn  [private]
```

Tranzakcia v databáze.

Dokumentácia pre túto triedu bola generovaná z nasledujúcich súborov:

- database.h
- database.cpp

4.6 Dokumentácia triedy DatabaseManager

`#include <database.h>`

Statické verejné metódy

- static void initConnectionString ()

Statické verejné atribúty

- static ConnectionString connectionString

Statické privátne metódy

- static void loadEnvFromFile (std::string filePath)

4.6.1 Detailný popis

Trieda ktorá pracuje s konekčným stringom a jej načítaním.

4.6.2 Dokumentácia k metódam

4.6.2.1 initConnectionString()

```
void DatabaseManager::initConnectionString ( )  [static]
```

Inicializácia konekčného stringu z premenných prostredia.

4.6.2.2 loadEnvFromFile()

```
void DatabaseManager::loadEnvFromFile (
            std::string filePath )  [static], [private]
```

Načítanie premenných prostredia zo súboru.

Parametre

filePath	Názov súboru kde sú uložené premenné prostredia.

4.6.3 Dokumentácia k dátovým členom

4.6.3.1 connectionString

```
ConnectionString DatabaseManager::connectionString  [static]
```

Aktuálny konekčný string.

Dokumentácia pre túto triedu bola generovaná z nasledujúcich súborov:

- database.h
- database.cpp

4.7 Dokumentácia triedy Ensemble

```
#include <ensemble.h>
```

Verejné metódy

- Ensemble ()
- double forward (int ∗inputArray, int ∗essenceCounts)

Privátne atribúty

- torch::jit::script::Module models [3]

4.7.1 Detailný popis

Trieda zodpovedná za súborové učenie.

4.7.2 Dokumentácia konštruktoru a deštruktoru

4.7.2.1 Ensemble()

```
Ensemble::Ensemble ( )
```

Konštruktor súborového učenia načíta modely.

4.7.3 Dokumentácia k metódam

4.7.3.1 forward()

```
double Ensemble::forward (
            int * inputArray,
            int * essenceCounts )
```

Pomocou tejto metódy vykonáme predikciu skóre šachovej pozície pomocou vstupnej matice 8x8x2.

Parametre

inputArray	Vstupná matica šachovnica 8x8x2. Prvý kanál reprezentuje šachové figúrky. Druhý kanál reprezentuje hráča na rade.
essenceCounts	Počty esencií v hernej konfigurácii.

Návratová hodnota

Predikcia evaluácie šachovej pozície.

4.7.4 Dokumentácia k dátovým členom

4.7.4.1 models

`torch::jit::script::Module Ensemble::models[3] [private]`

Modely, ktoré tvoria naše súborové učenie.

Dokumentácia pre túto triedu bola generovaná z nasledujúcich súborov:

- ensemble.h
- ensemble.cpp

4.8 Dokumentácia štruktúry EssenceArgs

`#include <board.h>`

Verejné atribúty

- PieceEssence whitePawn = Classic
- PieceEssence whiteRook = Classic
- PieceEssence whiteKnight = Classic
- PieceEssence whiteBishop = Classic
- PieceEssence blackPawn = Classic
- PieceEssence blackRook = Classic
- PieceEssence blackKnight = Classic
- PieceEssence blackBishop = Classic

4.8.1 Detailný popis

Herná konfigurácia esencií.

4.8.2 Dokumentácia k dátovým členom

4.8.2.1 blackBishop

```
PieceEssence EssenceArgs::blackBishop = Classic
```

Esencia čierneho strelca.

4.8.2.2 blackKnight

```
PieceEssence EssenceArgs::blackKnight = Classic
```

Esencia čierneho rytiera.

4.8.2.3 blackPawn

```
PieceEssence EssenceArgs::blackPawn = Classic
```

Esencia čierneho pešiaka.

4.8.2.4 blackRook

```
PieceEssence EssenceArgs::blackRook = Classic
```

Esencia čiernej veže.

4.8.2.5 whiteBishop

```
PieceEssence EssenceArgs::whiteBishop = Classic
```

Esencia bieleho strelca.

4.8.2.6 whiteKnight

```
PieceEssence EssenceArgs::whiteKnight = Classic
```

Esencia bieleho rytiera.

4.8.2.7 whitePawn

```
PieceEssence EssenceArgs::whitePawn = Classic
```

Esencia bieleho pešiaka.

4.8.2.8 whiteRook

```
PieceEssence EssenceArgs::whiteRook = Classic
```

Esencia bielej veže.

Dokumentácia pre túto štruktúru (struct) bola generovaná z nasledujúceho súboru:

- board.h

4.9 Dokumentácia štruktúry EvalArgs

```
#include <evaluation.h>
```

Verejné atribúty

- int matEval [2]
- int curPhase = 0
- int mg_pcsqEval [2]
- int eg_pcsqEval [2]
- int mobCount [2 ∗6]
- int attCount [2]
- int attWeight [2]

4.9.1 Detailný popis

Evaluačné argumenty na výpočet evaluácie šachovnice.

4.9.2 Dokumentácia k dátovým členom

4.9.2.1 attCount

```
int EvalArgs::attCount[2]
```

Počítadlá útokov na kráľa.

4.9.2.2 attWeight

```
int EvalArgs::attWeight[2]
```

Hodnoty útokov na kráľa.

4.9.2.3 curPhase

```
int EvalArgs::curPhase = 0
```

Aktuálna fáza hry. S ubúdajúcim materiálom sa táto hodnota znižuje.

4.9.2.4 eg_pcsqEval

```
int EvalArgs::eg_pcsqEval[2]
```

Hodnoty piece-square tabuliek pre koncovú fázu hry.

4.9.2.5 matEval

```
int EvalArgs::matEval[2]
```

Materiálová evaluácia.

4.9.2.6 mg_pcsqEval

```
int EvalArgs::mg_pcsqEval[2]
```

Hodnoty piece-square tabuliek pre strednú fázu hry.

4.9.2.7 mobCount

```
int EvalArgs::mobCount[2 *6]
```

Počítadla mobilít pre jednotlivé typy figúrok.

Dokumentácia pre túto štruktúru (struct) bola generovaná z nasledujúceho súboru:

- evaluation.h

4.10 Dokumentácia triedy Evaluation

```
#include <evaluation.h>
```

Statické verejné metódy

- static int get_mg_pcsq (PieceColor color, PieceType type, char x, char y)
- static int get_eg_pcsq (PieceColor color, PieceType type, char x, char y)

Statické verejné atribúty

- static const int pieceMatValues [6] = { 100, 500, 320, 330, 900, 0 }
- static const int piecePhaseValues [6] = { 0, 2, 1, 1, 4, 0 }
- static const int startPhase
- static const int mg_pcsq [6][64]
- static const int eg_pcsq [6][64]
- static const int mg_pieceMobWeights [6] = { 2, 2, 2, 2, 2, 1 }
- static const int eg_pieceMobWeights [6] = { 4, 4, 4, 4, 4, 1 }
- static const int pieceMobPenalties [6] = { 0, 7, 0, 0, 14, 0 }
- static const int mg_pieceTropismWeights [5] = { 1, 2, 3, 2, 2 }
- static const int eg_pieceTropismWeights [5] = { 1, 1, 3, 1, 4 }
- static const int pieceAttWeights [5] = { 1, 3, 1, 3, 5 }
- static const int safetyTable [100]

4.10.1 Detailný popis

Trieda zodpovedná za konfigurácia evaluácie šachových pozícií.

4.10.2 Dokumentácia k metódam

4.10.2.1 get_eg_pcsq()

```
int Evaluation::get_eg_pcsq (
            PieceColor color,
            PieceType type,
            char x,
            char y )  [static]
```

Výpočet piece-square hodnoty danej figúrky na určitej pozícii v koncovej fáze hry.

Parametre

color	Vlastník danej figúrky.
type	Typ danej figúrky.
x	Koordinát X danej pozície.
y	Koordinát Y danej pozície.

Návratová hodnota

Požadovaná hodnota z piece-square tabuľky.

4.10.2.2 get_mg_pcsq()

```
int Evaluation::get_mg_pcsq (
            PieceColor color,
            PieceType type,
            char x,
            char y )  [static]
```

Výpočet piece-square hodnoty danej figúrky na určitej pozícii v strednej fáze hry.

Parametre

color	Vlastník danej figúrky.
type	Typ danej figúrky.
x	Koordinát X danej pozície.
y	Koordinát Y danej pozície.

Návratová hodnota

Požadovaná hodnota z piece-square tabuľky.

4.10.3 Dokumentácia k dátovým členom

4.10.3.1 eg_pcsq

```
const int Evaluation::eg_pcsq  [static]
```

Piece-square tabuľky koncovej fázy hry.

4.10.3.2 eg_pieceMobWeights

```
const int Evaluation::eg_pieceMobWeights = { 4, 4, 4, 4, 4, 1 }  [static]
```

Váhy mobilít koncovej fázy hry.

4.10.3.3 eg_pieceTropismWeights

```
const int Evaluation::eg_pieceTropismWeights = { 1, 1, 3, 1, 4 }  [static]
```

Váhy tropizmu koncovej fázy hry.

4.10.3.4 mg_pcsq

```
const int Evaluation::mg_pcsq  [static]
```

Piece-square tabuľky strednej fázy hry.

4.10.3.5 mg_pieceMobWeights

```
const int Evaluation::mg_pieceMobWeights = { 2, 2, 2, 2, 2, 1 }  [static]
```

Váhy mobilít strednej fázy hry.

4.10.3.6 mg_pieceTropismWeights

```
const int Evaluation::mg_pieceTropismWeights = { 1, 2, 3, 2, 2 }  [static]
```

Váhy tropizmu strednej fázy hry.

4.10.3.7 pieceAttWeights

```
const int Evaluation::pieceAttWeights = { 1, 3, 1, 3, 5 }  [static]
```

Váhy útokov figúrok.

4.10.3.8 pieceMatValues

```
const int Evaluation::pieceMatValues = { 100, 500, 320, 330, 900, 0 }  [static]
```

Materiálové hodnoty figúrok.

4.10.3.9 pieceMobPenalties

```
const int Evaluation::pieceMobPenalties = { 0, 7, 0, 0, 14, 0 }  [static]
```

Penalty mobilít figúrok.

4.10.3.10 piecePhaseValues

```
const int Evaluation::piecePhaseValues = { 0, 2, 1, 1, 4, 0 }  [static]
```

Fázové hodnoty figúrok.

4.10.3.11 safetyTable

```
const int Evaluation::safetyTable  [static]
```

Inicializátor:

```
= {
    0,   0,   1,   2,   3,   5,   7,   9,  12,  15,
    18,  22,  26,  30,  35,  39,  44,  50,  56,  62,
    68,  75,  82,  85,  89,  97, 105, 113, 122, 131,
    140, 150, 169, 180, 191, 202, 213, 225, 237, 248,
    260, 272, 283, 295, 307, 319, 330, 342, 354, 366,
    377, 389, 401, 412, 424, 436, 448, 459, 471, 483,
    494, 500, 500, 500, 500, 500, 500, 500, 500, 500,
    500, 500, 500, 500, 500, 500, 500, 500, 500, 500,
    500, 500, 500, 500, 500, 500, 500, 500, 500, 500,
    500, 500, 500, 500, 500, 500, 500, 500, 500, 500
}
```

Bezpečnostná tabuľka potrebná na výpočet bezpečnosti kráľa.

4.10.3.12 startPhase

```
const int Evaluation::startPhase  [static]
```

Inicializátor:

```
= 16 * Evaluation::piecePhaseValues[Pawn] + 4 * Evaluation::piecePhaseValues[Knight]
+ 4 * Evaluation::piecePhaseValues[Bishop] + 4 * Evaluation::piecePhaseValues[Rook] + 2 *
      Evaluation::piecePhaseValues[Queen]
```

Počiatočná fáza hry.

Dokumentácia pre túto triedu bola generovaná z nasledujúcich súborov:

- evaluation.h
- evaluation.cpp

4.11 Dokumentácia triedy FairyStockfish

`#include <fairyStockfish.h>`

Verejné metódy

- bool start (EssenceArgs essenceArgs)
- void exit ()
- bool sendCommand (std::string command)
- void setConfigFile (EssenceArgs essenceArgs)
- std::string waitForResponse (std::string target)
- std::string receiveResponse ()

Privátne atribúty

- PROCESS_INFORMATION **piProcInfo** {}
- HANDLE **handleStdinRead** = NULL
- HANDLE **handleStdinWrite** = NULL
- HANDLE **handleStdoutRead** = NULL
- HANDLE **handleStdoutWrite** = NULL

4.11.1 Detailný popis

Pomocou tejto triedy ovládame engine FairyStockfish.

4.11.2 Dokumentácia k metódam

4.11.2.1 exit()

```
void FairyStockfish::exit ( )
```

Touto metódou spustený engine terminujeme.

4.11.2.2 receiveResponse()

```
std::string FairyStockfish::receiveResponse ( )
```

Od enginu prijmeme odpoveď.

Návratová hodnota

Prijatá správa.

4.11.2.3 sendCommand()

```
bool FairyStockfish::sendCommand (
            std::string command )
```

Enginu pošleme UCI príkaz.

Parametre

command	UCI príkaz.

Návratová hodnota

Ak bol príkaz úspešne odoslaný tak vráti true, v opačnom prípade vráti false.

4.11.2.4 setConfigFile()

```
void FairyStockfish::setConfigFile (
            EssenceArgs essenceArgs )
```

Vytvorenie konfiguračného súboru pre danú konfiguráciu esencií.

Parametre

essenceArgs	Konfigurácia esencií.

4.11.2.5 start()

```
bool FairyStockfish::start (
            EssenceArgs essenceArgs )
```

Engine spustíme s danou konfiguráciou esencií.

Parametre

essenceArgs	Konfigurácia esencií.

Návratová hodnota

Vráti hodnotu true ak bol engine úspešne spustený, v opačnom prípade vráti false.

4.11.2.6 waitForResponse()

```
std::string FairyStockfish::waitForResponse (
            std::string target )
```

Od enginu prijímame odpovede až kým nenájdeme očakávanú odpoveď.

Parametre

target	Začiatok odpovede na ktorú čakáme. (napr. bestmove)

Návratová hodnota

Prijatá správa.

Dokumentácia pre túto triedu bola generovaná z nasledujúcich súborov:

- fairyStockfish.h
- fairyStockfish.cpp

4.12 Dokumentácia štruktúry Ghost

```
#include <piece.h>
```

Verejné atribúty

- int x = -1
- int y = -1
- int parentX = -1
- int parentY = -1

4.12.1 Detailný popis

Figúrka zanechaná po vykonaní 'hasty' pohybu.

4.12.2 Dokumentácia k dátovým členom

4.12.2.1 parentX

```
int Ghost::parentX = -1
```

Koordinát X rodičovskej figúrky.

4.12.2.2 parentY

```
int Ghost::parentY = -1
```

Koordinát Y rodičovskej figúrky.

4.12.2.3 x

```
int Ghost::x = -1
```

Koordinát X tejto figúrky.

4.12.2.4 y

```
int Ghost::y = -1
```

Koordinát Y tejto figúrky.

Dokumentácia pre túto štruktúru (struct) bola generovaná z nasledujúceho súboru:

- piece.h

4.13 Dokumentácia štruktúry LegalMove

```
#include <mobility.h>
```

Verejné metódy

- LegalMove ()
- LegalMove (int x1, int y1, int x2, int y2, Mobility mobility)
- LegalMove (std::string value)
- std::string toString ()
- std::string toStringWithFlags (PieceColor color)

Verejné atribúty

- int x1 = 0
- int y1 = 0
- int x2 = 0
- int y2 = 0
- Mobility mobility
- PieceType promotion = Pawn
- Castling castling = none

4.13.1 Detailný popis

Štruktúra reprezentujúca legálny pohyb.

4.13.2 Dokumentácia konštruktoru a deštruktoru

4.13.2.1 LegalMove() [1/3]

```
LegalMove::LegalMove ( )
```

Predvolený konštruktor.

4.13.2.2 LegalMove() [2/3]

```
LegalMove::LegalMove (
            int x1,
            int y1,
            int x2,
            int y2,
            Mobility mobility )
```

Konštruktor pre dané hodnoty.

Parametre

x1	Koordinát X zdrojovej pozície.
y1	Koordinát Y zdrojovej pozície.
x2	Koordinát X cieľovej pozície.
y2	Koordinát Y cieľovej pozície.
mobility	Pravidlá pohybu.

4.13.2.3 LegalMove() [3/3]

```
LegalMove::LegalMove (
            std::string value )
```

Konštuktor pre algebraickú notáciu.

Parametre

value	Algebraická notácia pohybu.

4.13.3 Dokumentácia k metódam

4.13.3.1 toString()

```
std::string LegalMove::toString ( )
```

Konverzia pohybu do algebraickej notácie.

Návratová hodnota

Algebraická notácia pohybu.

4.13.3.2 toStringWithFlags()

```
std::string LegalMove::toStringWithFlags (
            PieceColor color )
```

Konverzia pohybu do algebraickej notácie s flagmi pohybu.

Parametre

color	Vlastník figúrky.

Návratová hodnota

Algebraická notácia pohybu.

4.13.4 Dokumentácia k dátovým členom

4.13.4.1 castling

```
Castling LegalMove::castling = none
```

Typ rošády.

4.13.4.2 mobility

```
Mobility LegalMove::mobility
```

Pravidlá pohybu.

4.13.4.3 promotion

```
PieceType LegalMove::promotion = Pawn
```

Typ figúrky po vykonaní promócie.

4.13.4.4 x1

```
int LegalMove::x1 = 0
```

Koordinát X zdrojovej pozície.

4.13.4.5 x2

```
int LegalMove::x2 = 0
```

Koordinát X cieľovej pozície.

4.13.4.6 y1

```
int LegalMove::y1 = 0
```

Koordinát Y zdrojovej pozície.

4.13.4.7 y2

```
int LegalMove::y2 = 0
```

Koordinát Y cieľovej pozície.

Dokumentácia pre túto štruktúru (struct) bola generovaná z nasledujúcich súborov:

- mobility.h
- mobility.cpp

4.14 Dokumentácia triedy Mobilities

`#include <mobility.h>`

Statické verejné metódy

- static void printMobilities (PieceType type, PieceEssence essence)

Statické verejné atribúty

- static std::vector< Mobility > mobilityConfig [6][3]

4.14.1 Detailný popis

Trieda zodpovedná za konfiguráciu mobilít figúrok.

4.14.2 Dokumentácia k metódam

4.14.2.1 printMobilities()

```
void Mobilities::printMobilities (
            PieceType type,
            PieceEssence essence )  [static]
```

Vypíše mobility pre danú figúrku.

Parametre

type	Typ figúrky.
essence	Esencia figúrky.

4.14.3 Dokumentácia k dátovým členom

4.14.3.1 mobilityConfig

```
std::vector< Mobility > Mobilities::mobilityConfig  [static]
```

Zoznam konfigurácií mobilít figúrok.

Dokumentácia pre túto triedu bola generovaná z nasledujúcich súborov:

- mobility.h
- mobility.cpp

4.15 Dokumentácia štruktúry Mobility

`#include <mobility.h>`

Verejné atribúty

- MovementType type = Move
- int start_x = 0
- int start_y = 0
- int direction_x = 0
- int direction_y = 0
- int limit = 0
- MobilityFlags flags {}

4.15.1 Detailný popis

Štruktúra pomocou ktorej môžeme dynamicky definovať pohyby figúrok.

4.15.2 Dokumentácia k dátovým členom

4.15.2.1 direction_x

```
int Mobility::direction_x = 0
```

Relatívny smer pohybu na osi X.

4.15.2.2 direction_y

```
int Mobility::direction_y = 0
```

Relatívny smer pohybu na osi Y.

4.15.2.3 flags

```
MobilityFlags Mobility::flags {}
```

Flagy pohybu pre špeciálne .

4.15.2.4 limit

```
int Mobility::limit = 0
```

Limit pohybu v danom smere. Hodnota 0 reprezentuje pohyb bez obmedzení.

4.15.2.5 start_x

```
int Mobility::start_x = 0
```

Relatívny smer pohybu na osi X prvého pohybu.

4.15.2.6 start_y

```
int Mobility::start_y = 0
```

Relatívny smer pohybu na osi Y prvého pohybu.

4.15.2.7 type

```
MovementType Mobility::type = Move
```

Typ pohybu.

Dokumentácia pre túto štruktúru (struct) bola generovaná z nasledujúceho súboru:

- mobility.h

4.16 Dokumentácia štruktúry MobilityFlags

```
#include <mobility.h>
```

Verejné atribúty

- bool initiative = false
- bool hasty = false
- bool uninterruptible = false
- bool vigilant = false

4.16.1 Detailný popis

Štruktúra s mobilitnými flagmi. Sú potrebné na definíciu špeciálnych pohybov ako En Passant.

4.16.2 Dokumentácia k dátovým členom

4.16.2.1 hasty

```
bool MobilityFlags::hasty = false
```

Pohyb s týmto flagom po sebe na zanecháva figúrku o 1 pohyb pred cieľovým políčkom. Túto figúrku je možné napadnúť útokom s 'vigilant' flagom.

4.16.2.2 initiative

```
bool MobilityFlags::initiative = false
```

Pohyb s týmto flagom je možné vykonať len vtedy, keď figúrka sa ešte nepohla.

4.16.2.3 uninterruptible

```
bool MobilityFlags::uninterruptible = false
```

Tento pohyb je možné vykonať len na finálnom políčku pohybu.

4.16.2.4 vigilant

```
bool MobilityFlags::vigilant = false
```

Pohyb s týmto flagom môže napadnúť figúrku zanechanú pohybom s 'hasty' flagom.

Dokumentácia pre túto štruktúru (struct) bola generovaná z nasledujúceho súboru:

- mobility.h

4.17 Dokumentácia triedy MoveGenerator

```
#include <board.h>
```

Statické verejné metódy

- static void clearMoves (Board *board)
- static void generateMoves (Board *board)
- static void generateMoves (Board *board, Piece piece, char x, char y)

4.17.1 Detailný popis

Pomocná trieda na generáciu pohybov šachovnice

4.17.2 Dokumentácia k metódam

4.17.2.1 clearMoves()

```
void MoveGenerator::clearMoves (
            Board * board )  [static]
```

Táto funkcia odstráni všetký vygenerované pohyby a resetuje všetky relevantné evaluačné argumenty.

Parametre

board	Šachovnica kde odstraňujeme vygenerované pohyby.

4.17.2.2 generateMoves() [1/2]

```
void MoveGenerator::generateMoves (
            Board * board )  [static]
```

Táto funkcia vygeneruje všetky pseudolegálne pohyby.

Parametre

board	Šachovnica kde generujeme pohyby.

4.17.2.3 generateMoves() [2/2]

```
void MoveGenerator::generateMoves (
            Board * board,
            Piece piece,
            char x,
            char y )  [static]
```

Táto funkcia vygeneruje všetky pseudolegálne pohyby pre figúrku na konkrétnej pozícii.

Parametre

board	Šachovnica kde generujeme pohyby.
piece	Figúrka ktorej generujeme pohyby.
x	Koordinát X kde sa nachádza naša figúrka.
y	Koordinát Y kde sa nachádza naša figúrka.

Dokumentácia pre túto triedu bola generovaná z nasledujúcich súborov:

- board.h
- moveGenerator.cpp

4.18 Dokumentácia štruktúry PerformanceArgs

```
#include <searching.h>
```

Verejné metódy

- void printPerformance ()
- void printPerformance (std::chrono::high_resolution_clock::time_point stop, long long durationCur)

Verejné atribúty

- std::chrono::high_resolution_clock::time_point start = std::chrono::high_resolution_clock::now()
- int positionsCur = 0
- int positionsTotal = 0
- long long durationTotal = 0

Statické verejné atribúty

- static std::mutex mutex

4.18.1 Detailný popis

Výkonnostné argumenty prehľadávacej funkcie.

4.18.2 Dokumentácia k metódam

4.18.2.1 printPerformance() [1/2]

```
void PerformanceArgs::printPerformance ( )
```

Vypíše aktuálny výkon prehľadávacej funkcie.

4.18.2.2 printPerformance() [2/2]

```
void PerformanceArgs::printPerformance (
            std::chrono::high_resolution_clock::time_point stop,
            long long durationCur )
```

Vypíše aktuálny výkon prehľadávacej funkcie.

Parametre

stop	Konečná časová značka.
durationCur	Čas výpočtu.

4.18.3 Dokumentácia k dátovým členom

4.18.3.1 durationTotal

```
long long PerformanceArgs::durationTotal = 0
```

Celkový čas strávený prehľadávaním pozícií.

4.18.3.2 mutex

```
std::mutex PerformanceArgs::mutex  [static]
```

Mutex na uzamknutie výkonnostných argumentov.

4.18.3.3 positionsCur

```
int PerformanceArgs::positionsCur = 0
```

Počet pozícií, ktoré boli aktuálne prehľadané.

4.18.3.4 positionsTotal

```
int PerformanceArgs::positionsTotal = 0
```

Počet všetkých pozícii, ktoré boli prehľadané.

4.18.3.5 start

```
std::chrono::high_resolution_clock::time_point PerformanceArgs::start = std::chrono::high_↩
resolution_clock::now()
```

Počiatočná časová značka.

Dokumentácia pre túto štruktúru (struct) bola generovaná z nasledujúcich súborov:

- searching.h
- searching.cpp

4.19 Dokumentácia štruktúry Piece

```
#include <piece.h>
```

Verejné metódy

- char toChar ()

Verejné atribúty

- PieceColor color = White
- PieceType type = Pawn
- PieceEssence essence = Classic

4.19.1 Detailný popis

Štruktúra reprezentujúca šachovú figúrku.

4.19.2 Dokumentácia k metódam

4.19.2.1 toChar()

```
char Piece::toChar ( )
```

Konverzia figúrky do znaku.

Návratová hodnota

Znak reprezentujúci figúrku

4.19.3 Dokumentácia k dátovým členom

4.19.3.1 color

```
PieceColor Piece::color = White
```

Vlastník figúrky.

4.19.3.2 essence

```
PieceEssence Piece::essence = Classic
```

Esencia figúrky.

4.19.3.3 type

```
PieceType Piece::type = Pawn
```

Typ figúrky.

Dokumentácia pre túto štruktúru (struct) bola generovaná z nasledujúcich súborov:

- piece.h
- piece.cpp

4.20 Dokumentácia triedy Random

```
#include <random.h>
```

Statické verejné metódy

- static void initSeed ()
- static bool coinFlip ()
- static int generateRandomNumber (int lowerBound, int upperBound)
- template<typename T >
  static T getRandomElement (const std::vector< T > &vec)

Statické privátne atribúty

- static std::mt19937 gen

4.20.1 Detailný popis

Trieda zodpovedná za generáciu náhodných čísiel.

4.20.2 Dokumentácia k metódam

4.20.2.1 coinFlip()

```
bool Random::coinFlip ( )  [static]
```

Generácia náhodného čísla od 0 po 1.

Návratová hodnota

Náhodne vygenerované číslo.

4.20.2.2 generateRandomNumber()

```
int Random::generateRandomNumber (
            int lowerBound,
            int upperBound )  [static]
```

Generácia náhodného čísla v danom rozsahu.

Parametre

lowerBound	Spodná hranica náhodného čísla
upperBound	Vrchná hranica náhodného čísla

Návratová hodnota

Náhodne vygenerované číslo

4.20.2.3 getRandomElement()

```
template<typename T >
T Random::getRandomElement (
            const std::vector< T > & vec )  [inline], [static]
```

Výber náhodného elemetu z vektora T.

Parametre

vec	Zoznam prvkov T.

Návratová hodnota

Náhodne vybraný prvok T.

4.20.2.4 initSeed()

```
void Random::initSeed ( )  [static]
```

Inicializácia náhodného seedu.

4.20.3 Dokumentácia k dátovým členom

4.20.3.1 gen

```
std::mt19937 Random::gen  [static], [private]
```

Generátor náhodných čísiel.

Dokumentácia pre túto triedu bola generovaná z nasledujúcich súborov:

- random.h
- random.cpp

4.21 Dokumentácia štruktúry SearchArgs

```
#include <searching.h>
```

Verejné atribúty

- int curDepth = 0
- int maxDepth = 0
- int alpha = INT_MIN
- int beta = INT_MAX

4.21.1 Detailný popis

Argumenty prehľadávacej funkcie.

4.21.2 Dokumentácia k dátovým členom

4.21.2.1 alpha

```
int SearchArgs::alpha = INT_MIN
```

Alpha hodnota v pruning algoritme.

4.21.2.2 beta

```
int SearchArgs::beta = INT_MAX
```

Beta hodnota v pruning algoritme.

4.21.2.3 curDepth

```
int SearchArgs::curDepth = 0
```

Aktuálna hĺbka prehľadávania.

4.21.2.4 maxDepth

```
int SearchArgs::maxDepth = 0
```

Maximálna hĺbka prehľadávania.

Dokumentácia pre túto štruktúru (struct) bola generovaná z nasledujúceho súboru:

- searching.h

4.22 Dokumentácia triedy SearchManager

```
#include <searching.h>
```

Statické verejné metódy

- static std::pair< bool, LegalMove > calculateBestMove (Board board, int depth, bool useEnsemble, bool debug)
- static std::pair< bool, LegalMove > calculateBestMove_threads (Board board, int depth, int threadCount, bool useEnsemble, bool debug)
- static int minimax (Board board, PieceColor playerColor, SearchArgs searchArgs, PerformanceArgs *performanceArgs, bool useEnsemble, bool debug)

Statické verejné atribúty

- static TranspositionCache cache = TranspositionCache(4096)
- static Ensemble ensemble = Ensemble()

Statické privátne atribúty

- static const int qLimit = 0

4.22.1 Detailný popis

Trieda zodpovedná za prehľadávanie pozícií na šachovnici.

4.22.2 Dokumentácia k metódam

4.22.2.1 calculateBestMove()

```
std::pair< bool, LegalMove > SearchManager::calculateBestMove (
            Board board,
            int depth,
            bool useEnsemble,
            bool debug )  [static]
```

Sériový výpočet najlepšieho pohybu.

Parametre

board	Šachovnica kde hľadáme najlepší pohyb.
depth	Hĺbka do ktorej vykonávame prehľadávanie.
useEnsemble	Bit ktorým môžeme nastaviť či chceme namiesto štandardnej evaluačnej funkcie použiť súborové učenie.
debug	Bit ktorým môžeme nastaviť či chceme vypisovať debug správy.

Návratová hodnota

Vráti pár bitu, ktorý reprezentuje či sme našli legálny pohyb a najlepšieho možného pohybu.

4.22.2.2 calculateBestMove_threads()

```
std::pair< bool, LegalMove > SearchManager::calculateBestMove_threads (
            Board board,
            int depth,
            int threadCount,
            bool useEnsemble,
            bool debug )  [static]
```

Paralelný výpočet najlepšieho pohybu.

Parametre

board	Šachovnica kde hľadáme najlepší pohyb.
depth	Hĺbka do ktorej vykonávame prehľadávanie.
threadCount	Počet paralelných vlákien, ktoré hľadajú najlepší pohyb.
useEnsemble	Bit ktorým môžeme nastaviť či chceme namiesto štandardnej evaluačnej funkcie použiť súborové učenie.
debug	Bit ktorým môžeme nastaviť či chceme vypisovať debug správy.

Návratová hodnota

Vráti pár bitu, ktorý reprezentuje či sme našli legálny pohyb a najlepšieho možného pohybu.

4.22.2.3 minimax()

```
int SearchManager::minimax (
            Board board,
            PieceColor playerColor,
            SearchArgs searchArgs,
            PerformanceArgs * performanceArgs,
            bool useEnsemble,
            bool debug )  [static]
```

Rekurzívny algoritmus, ktorý vypočíta hodnotu najlepšieho možného pohybu.

Parametre

board	Šachovnica kde hľadáme najlepší pohyb.
playerColor	Farba maximalizujúceho hráča.
searchArgs	Argumenty prehľadávania.
performanceArgs	Výkonnostné argumenty.
useEnsemble	Bit ktorým môžeme nastaviť či chceme namiesto štandardnej evaluačnej funkcie použiť súborové učenie.
debug	Bit ktorým môžeme nastaviť či chceme vypisovať debug správy.

Návratová hodnota

Hodnota najlepšieho pohybu.

4.22.3 Dokumentácia k dátovým členom

4.22.3.1 cache

```
TranspositionCache SearchManager::cache = TranspositionCache(4096)  [static]
```

Cache s transpozíciami.

4.22.3.2 ensemble

```
Ensemble SearchManager::ensemble = Ensemble()  [static]
```

Súbor modelov, ktorý sa používa v 'ensemble' AI na evaluáciu.

4.22.3.3 qLimit

```
const int SearchManager::qLimit = 0  [static], [private]
```

Limit Q-searchu (zabraňuje horizont efekt).

Dokumentácia pre túto triedu bola generovaná z nasledujúcich súborov:

- searching.h
- searching.cpp

4.23 Dokumentácia triedy SimulationManager

```
#include <simulation.h>
```

Statické verejné metódy

- static void simulateGames (int gameCounter, EssenceArgs essenceArgs, int malakhDepth, int fairy↩ StockfishDepth, bool useEnsemble, bool useDB, std::string outputFilename)

4.23.1 Detailný popis

Trieda zodpovedná za simuláciu hier medzi enginom Malakh a FairyStockfish.

4.23.2 Dokumentácia k metódam

4.23.2.1 simulateGames()

```
void SimulationManager::simulateGames (
            int gameCounter,
            EssenceArgs essenceArgs,
            int malakhDepth,
            int fairyStockfishDepth,
            bool useEnsemble,
            bool useDB,
            std::string outputFilename )  [static]
```

Pomocou tejto metódy spustíme simuláciu hier medzi enginami Malakh a FairyStockfish.

Parametre

gameCounter	Počet hier, ktoré majú byť odohrané.
essenceArgs	Nastavenie esencií.
malakhDepth	HĺBka prehľadávania enginu Malakh.
fairyStockfishDepth	HĺBka prehľadávanie enginu FairyStockfish.
useEnsemble	Hodnota pomocou ktorej môžeme nastaviť 'basic' (false) alebo 'ensemble' (true) AI režim pre engine Malakh.
useDB	Hodnota pomocou ktorej môžeme vypnúť alebo zapnúť ukladanie podrobných výsledkov simulácie do databázy.
outputFilename	Výstupný súbor do ktorého zapíšeme výsledky simulácie.

Dokumentácia pre túto triedu bola generovaná z nasledujúcich súborov:

- simulation.h
- simulation.cpp

4.24 Dokumentácia štruktúry Transposition

```
#include <transpositions.h>
```

Verejné atribúty

- int value = 0
- int depth = -1

4.24.1 Detailný popis

Štruktúra reprezentujúca transpozíciu.

4.24.2 Dokumentácia k dátovým členom

4.24.2.1 depth

```
int Transposition::depth = -1
```

HĺBka do ktorej bola táto pozícia prehľadaná.

4.24.2.2 value

```
int Transposition::value = 0
```

Evaluácia danej šachovej pozície.

Dokumentácia pre túto štruktúru (struct) bola generovaná z nasledujúceho súboru:

- transpositions.h

4.25 Dokumentácia triedy TranspositionCache

`#include <transpositions.h>`

Verejné metódy

- TranspositionCache (int capacity)
- Transposition get (unsigned long long key)
- void put (unsigned long long key, Transposition value)

Privátne atribúty

- std::unordered_map< unsigned long long, Transposition > cache
- std::list< unsigned long long > order
- int capacity
- std::mutex * mutex

4.25.1 Detailný popis

Trieda reprezentujúca LRU cache v ktorej ukladáma transpozície.

4.25.2 Dokumentácia konštruktoru a deštruktoru

4.25.2.1 TranspositionCache()

```
TranspositionCache::TranspositionCache (
            int capacity )
```

Konštruktor inicializuje cache danej veľkosti.

Parametre

capacity	Kapacita našej cache.

4.25.3 Dokumentácia k metódam

4.25.3.1 get()

```
Transposition TranspositionCache::get (
            unsigned long long key )
```

Pomocou tejto metódy z cache vyberieme transpozíciu. Ak ju nájdeme, tak ju presunieme na vrch nášho zoznamu.

Parametre

key	Hash hľadanej transpozície.

Návratová hodnota

Nájdená transpozícia. Ak žiadna nebola nájdená tak hĺbka prehľadávania vrátenej transpozície je -1.

4.25.3.2 put()

```
void TranspositionCache::put (
            unsigned long long key,
            Transposition value )
```

Pomocou tejto metódy do cache vložíme novú transpozíciu. Ak je naše cache plné, tak z cache odstránime transpozícia podľa LRU princípu.

Parametre

key	Hash novej transpozície.
value	Hodnota novej transpozície.

4.25.4 Dokumentácia k dátovým členom

4.25.4.1 cache

```
std::unordered_map<unsigned long long, Transposition> TranspositionCache::cache  [private]
```

Hashmapa v ktorej ukladáme všetky transpozície. Kľúčom tejto mapy je hash šachovej pozície.

4.25.4.2 capacity

```
int TranspositionCache::capacity  [private]
```

Maximálna kapacita našej cache.

4.25.4.3 mutex

```
std::mutex* TranspositionCache::mutex  [private]
```

Mutex pre úpravu cache v paralelnom vyhľadávaní.

4.25.4.4 order

```
std::list<unsigned long long> TranspositionCache::order  [private]
```

Zoznam reprezentuje poradie transpozícii na základe LRU princípu.

Dokumentácia pre túto triedu bola generovaná z nasledujúcich súborov:

- transpositions.h
- transpositions.cpp

4.26 Dokumentácia triedy uci

`#include <uci.h>`

Verejné metódy

- void run ()
- bool parseCommand (std::string command)

Privátne atribúty

- EssenceArgs essenceConfig
- Board board
- bool useEnsemble = false

4.26.1 Detailný popis

Trieda zodpovedná za UCI režim aplikácie.

4.26.2 Dokumentácia k metódam

4.26.2.1 parseCommand()

```
bool uci::parseCommand (
            std::string command )
```

Táto metóda spracuváva UCI príkazy.

Parametre

command	Prijatý príkaz

Návratová hodnota

Ak prijatý príkaz terminuje aplikáciu tak 1, v opačnom prípade 0.

4.26.2.2 run()

```
void uci::run ( )
```

Touto metódou spustíme UCI režim kde aplikácia čaká na UCI príkazy.

4.26.3 Dokumentácia k dátovým členom

4.26.3.1 board

```
Board uci::board  [private]
```

Stav šachovnice aktuálnej hry.

4.26.3.2 essenceConfig

```
EssenceArgs uci::essenceConfig  [private]
```

Konfigurácia esencií aktuálnej hry.

4.26.3.3 useEnsemble

```
bool uci::useEnsemble = false  [private]
```

Nastavenie AI režimu. Ak je hodnota false, tak používame 'basic' režim. Ak je hodnota true, tak používame 'ensemble' režim.

Dokumentácia pre túto triedu bola generovaná z nasledujúcich súborov:

- uci.h
- uci.cpp

4.27 Dokumentácia triedy ZobristHashing

```
#include <hashing.h>
```

Statické verejné metódy

- static void init ()

Statické verejné atribúty

- static unsigned long long squares [64 ∗2 ∗6]
- static unsigned long long notMoved [64]
- static unsigned long long ghosts [64 ∗64]
- static unsigned long long turn

Statické privátne metódy

- static unsigned long long rand64 ()

Statické privátne atribúty

- static bool isInitalized = false

4.27.1 Detailný popis

Trieda zodpovedná za implementáciu zobristovho hashovania.

4.27.2 Dokumentácia k metódam

4.27.2.1 init()

```
void ZobristHashing::init ( )  [static]
```

Pomocou tejto funkcie inicializujeme hodnoty zobristovho hashovania.

4.27.2.2 rand64()

```
unsigned long long ZobristHashing::rand64 ( )  [static], [private]
```

Generácia pseudonáhodného čísla.

Návratová hodnota

64-bitové číslo

4.27.3 Dokumentácia k dátovým členom

4.27.3.1 ghosts

```
unsigned long long ZobristHashing::ghosts  [static]
```

Hodnoty reprezentujúce špeciálnu figúrku na danej pozícii.

4.27.3.2 isInitalized

```
bool ZobristHashing::isInitalized = false  [static], [private]
```

Bit pomocou ktorého vieme či sme vykonali inicializáciu.

4.27.3.3 notMoved

```
unsigned long long ZobristHashing::notMoved  [static]
```

Hodnoty reprezentujúce nepohnuté figúrky na danej pozícii.

4.27.3.4 squares

```
unsigned long long ZobristHashing::squares  [static]
```

Hodnoty reprezentujúce konkrétne figúrky na danej pozícii.

4.27.3.5 turn

```
unsigned long long ZobristHashing::turn  [static]
```

Hodnota reprezentujúca hráča na rade.

Dokumentácia pre túto triedu bola generovaná z nasledujúcich súborov:

- hashing.h
- hashing.cpp

Kapitola 5

Dokumentácia súborov

5.1 Dokumentácia súboru bitboard.h

Definícia bitboardovej reprezentácie šachovnice.

Triedy

- struct BitBoard

5.1.1 Detailný popis

Definícia bitboardovej reprezentácie šachovnice.

Autor

Martin Šváb

Dátum

Máj 2024

5.2 bitboard.h

Zobraziť dokumentáciu tohoto súboru.

```
00001 #pragma once
00002
00003 /*****************************************************************/
00015 struct BitBoard {
00016 public:
00020     unsigned long long value = 0;
00021
00028     unsigned long long getBit(char x, char y);
00029
00036     void setBit(char x, char y);
00037
00044     void clearBit(char x, char y);
00045
00052     void printBits();
00053
00060     bool getKingAttack(char x, char y);
00061
00068     char getTropism(char x, char y);
00069 private:
00070
00077     unsigned long long getMask(char x, char y);
00078
00084     unsigned long long getKingPattern(unsigned long long value);
00085
00091     unsigned long long getTropismPattern(unsigned long long value);
00092 };
```

5.3 Dokumentácia súboru board.h

Definícia šachovnice a generácia pohybov.

```
#include <bitboard.h>
#include <piece.h>
#include <mobility.h>
#include <evaluation.h>
#include <hashing.h>
#include <vector>
```

Triedy

- struct EssenceArgs
- class Board
- class MoveGenerator

Enumerácie

- enum GameResult { Unresolved , WhiteWin , BlackWin , Stalemate }

5.3.1 Detailný popis

Definícia šachovnice a generácia pohybov.

Autor

Martin Šváb

Dátum

Máj 2024

5.3.2 Dokumentácia enumeračných typov

5.3.2.1 GameResult

```
enum GameResult
```

Enumerátor reprezentujúci výsledok hry.

Hodnoty enumerácií

Unresolved	Hra ešte nebola ukončená.
WhiteWin	Hra bola terminovaná výhrou bieleho hráča.
BlackWin	Hra bola terminovaná výhrou čierneho hráča.
Stalemate	Hra bola terminovaná remízou.

5.4 board.h

Zobraziť dokumentáciu tohoto súboru.

```
00001 #pragma once
00002
00003 /*****************************************************************/
00011 #include <bitboard.h>
00012 #include <piece.h>
00013 #include <mobility.h>
00014 #include <evaluation.h>
00015 #include <hashing.h>
00016 #include <vector>
00017
00018
00022 enum GameResult {
00026     Unresolved,
00030     WhiteWin,
00034     BlackWin,
00038     Stalemate
00039 };
00040
00044 struct EssenceArgs {
00048     PieceEssence whitePawn = Classic;
00049
00053     PieceEssence whiteRook = Classic;
00054
00058     PieceEssence whiteKnight = Classic;
00059
00063     PieceEssence whiteBishop = Classic;
00064
00068     PieceEssence blackPawn = Classic;
00069
00073     PieceEssence blackRook = Classic;
00074
00078     PieceEssence blackKnight = Classic;
00079
00083     PieceEssence blackBishop = Classic;
00084 };
00085
00089 class Board {
00090 public:
00094     PieceEssence essenceConfig[2*6] = {};
00095
00099     int essenceCounts[3];
00100
00104     BitBoard pieces[2*6];
00105
00109     BitBoard colors[2];
00110
00114     BitBoard attacks[2];
00115
00119     BitBoard allPieces;
00120
00124     BitBoard notMoved;
00125
00129     Ghost ghost;
00130
00134     int pieceCounts[2*6] = {};
00135
00139     int inputArray[2*8*8];
00140
00144     std::vector<LegalMove> moves[2];
00145
00149     bool movesValidated = false;
00150
00154     PieceColor curTurn = White;
00155
00159     EvalArgs eval{};
00160
00164     BoardHash hash{};
00165
00169     int halfmoveClock = 0;
00170
00174     int fullmoveClock = 1;
00175
00181     void initBoard(EssenceArgs essenceArgs);
00182
00190     void initBoard(EssenceArgs essenceArgs, BitBoard pieces[12], PieceColor curTurn);
00191
00195     void printBoard();
00196
00200     void printMoves();
00201
00208     int evalBoard(PieceColor color);
00209
```

```
00215     bool isQuiet();
00216
00226     bool makeMove(char x1, char y1, char x2, char y2);
00227
00238     bool makeMove(char x1, char y1, char x2, char y2, PieceType promotion);
00239
00246     bool makeMove(LegalMove move);
00247
00255     std::pair<bool, Piece> getPiece(char x, char y);
00256
00262     std::vector<LegalMove> getLegalMoves();
00263
00269     std::string toString();
00270
00276     GameResult getResult();
00277 private:
00283     void setEssenceConfig(EssenceArgs essenceArgs);
00284
00288     void clearBoard();
00289
00299     void addPiece(PieceColor color, PieceType type, char x, char y, bool isNew);
00300
00308     std::pair<bool, Piece> removePiece(char x, char y);
00309
00313     void refreshAggregations();
00314
00322     void setInputArrayPiece(char pieceIndex, char x, char y);
00323
00329     void setInputArrayColor(PieceColor color);
00330 };
00331
00335 class MoveGenerator {
00336 public:
00342     static void clearMoves(Board* board);
00343
00349     static void generateMoves(Board* board);
00350
00359     static void generateMoves(Board* board, Piece piece, char x, char y);
00360 };
```

5.5 Dokumentácia súboru database.h

Spojenie s databázou PostgreSQL pre ukladanie výsledkov simulácie.

```
#include <board.h>
#include <pqxx/pqxx>
#include <string>
#include <vector>
#include <pqxx/transaction.hxx>
```

Triedy

- struct ConnectionString
- class DatabaseManager
- class DatabaseConnection

5.5.1 Detailný popis

Spojenie s databázou PostgreSQL pre ukladanie výsledkov simulácie.

Autor

Martin Šváb

Dátum

Máj 2024

5.6 database.h

Zobraziť dokumentáciu tohoto súboru.

```
00001 #pragma once
00002
00003 /*****************************************************************/
00011 #define NOMINMAX
00012 #include <board.h>
00013 #include <pqxx/pqxx>
00014 #include <string>
00015 #include <vector>
00016 #include <pqxx/transaction.hxx>
00017
00018
00022 struct ConnectionString {
00023     /*
00024     * Meno databázy
00025     */
00026     std::string dbname;
00027
00031     std::string user;
00032
00036     std::string password;
00037
00041     std::string hostaddr;
00042
00046     std::string port;
00047
00053     std::string toString();
00054 };
00055
00059 class DatabaseManager {
00060 public:
00064     static ConnectionString connectionString;
00065
00069     static void initConnectionString();
00070 private:
00076     static void loadEnvFromFile(std::string filePath);
00077 };
00078
00082 class DatabaseConnection {
00083 public:
00087     DatabaseConnection();
00088
00095     int getIdEssenceConfig(EssenceArgs essenceArgs);
00096
00100     void updateScores();
00101
00111     void addBoardResult(Board board, int idEssenceConfig, GameResult gameResult);
00112 private:
00116     pqxx::connection connection;
00117
00121     pqxx::work txn;
00122 };
```

5.7 Dokumentácia súboru ensemble.h

Implementácia súborového učenia.

```
#include <torch/script.h>
```

Triedy

- class Ensemble

5.7.1 Detailný popis

Implementácia súborového učenia.

Autor

marti

Dátum

May 2024

5.8 ensemble.h

Zobraziť dokumentáciu tohoto súboru.

```
00001 #pragma once
00002
00003 /*****************************************************************/
00011 #include <torch/script.h>
00012
00013
00017 class Ensemble {
00018 public:
00022     Ensemble();
00023
00031     double forward(int* inputArray, int* essenceCounts);
00032 private:
00036     torch::jit::script::Module models[3];
00037 };
```

5.9 Dokumentácia súboru evaluation.h

Evaluácia šachovnice.

```
#include <piece.h>
```

Triedy

- struct EvalArgs
- class Evaluation

5.9.1 Detailný popis

Evaluácia šachovnice.

Autor

Martin Šváb

Dátum

Máj 2024

5.10 evaluation.h

Zobraziť dokumentáciu tohoto súboru.

```
00001 #pragma once
00002
00003 /*************************************************************************/
00011 #include <piece.h>
00012
00013
00017 struct EvalArgs {
00021     int matEval[2];
00022
00026     int curPhase = 0;
00027
00031     int mg_pcsqEval[2];
00032
00036     int eg_pcsqEval[2];
00037
00041     int mobCount[2*6];
00042
00046     int attCount[2];
00047
00051     int attWeight[2];
00052 };
00053
00057 class Evaluation {
00058 public:
00062     static const int pieceMatValues[6];
00063
00067     static const int piecePhaseValues[6];
00068
00072     static const int startPhase;
00073
00077     static const int mg_pcsq[6][64];
00078
00082     static const int eg_pcsq[6][64];
00083
00087     static const int mg_pieceMobWeights[6];
00088
00092     static const int eg_pieceMobWeights[6];
00093
00097     static const int pieceMobPenalties[6];
00098
00102     static const int mg_pieceTropismWeights[5];
00103
00107     static const int eg_pieceTropismWeights[5];
00108
00112     static const int pieceAttWeights[5];
00113
00117     static const int safetyTable[100];
00118
00128     static int get_mg_pcsq(PieceColor color, PieceType type, char x, char y);
00129
00139     static int get_eg_pcsq(PieceColor color, PieceType type, char x, char y);
00140 };
```

5.11 Dokumentácia súboru fairyStockfish.h

Ovládač pre engine FairyStockfish.

```
#include <Windows.h>
#include <string>
#include <vector>
#include <board.h>
```

Triedy

- class FairyStockfish

5.11.1 Detailný popis

Ovládač pre engine FairyStockfish.

Autor

Martin Šváb

Dátum

Máj 2024

5.12 fairyStockfish.h

Zobraziť dokumentáciu tohoto súboru.

```
00001 #pragma once
00002
00003 /*****************************************************************/
00011 #define NOMINMAX
00012 #include <Windows.h>
00013 #include <string>
00014 #include <vector>
00015 #include <board.h>
00016
00017
00021 class FairyStockfish {
00022 public:
00029     bool start(EssenceArgs essenceArgs);
00030
00034     void exit();
00035
00042     bool sendCommand(std::string command);
00043
00049     void setConfigFile(EssenceArgs essenceArgs);
00050
00057     std::string waitForResponse(std::string target);
00058
00064     std::string receiveResponse();
00065 private:
00066     PROCESS_INFORMATION piProcInfo{};
00067     HANDLE handleStdinRead = NULL;
00068     HANDLE handleStdinWrite = NULL;
00069     HANDLE handleStdoutRead = NULL;
00070     HANDLE handleStdoutWrite = NULL;
00071 };
```

5.13 Dokumentácia súboru hashing.h

Implementácia hashovanie šachovnice.

```
#include <piece.h>
```

Triedy

- class ZobristHashing
- struct BoardHash

5.13.1 Detailný popis

Implementácia hashovanie šachovnice.

Autor

Martin Šváb

Dátum

Máj 2024

5.14 hashing.h

Zobraziť dokumentáciu tohoto súboru.

```
00001 #pragma once
00002
00003 /*****************************************************************/
00011 #include <piece.h>
00012
00013
00017 class ZobristHashing {
00018 public:
00022     static unsigned long long squares[64*2*6];
00023
00027     static unsigned long long notMoved[64];
00028
00032     static unsigned long long ghosts[64*64];
00033
00037     static unsigned long long turn;
00038
00042     static void init();
00043 private:
00047     static bool isInitalized;
00048
00054     static unsigned long long rand64();
00055 };
00056
00060 struct BoardHash {
00064     unsigned long long value;
00065
00073     void switchSquare(Piece piece, char x, char y);
00074
00081     void switchNotMoved(char x, char y);
00082
00088     void switchGhost(Ghost ghost);
00089
00093     void switchTurn();
00094 };
```

5.15 Dokumentácia súboru mobility.h

Definícia mobilít figúrok.

```
#include <piece.h>
#include <vector>
#include <string>
```

Triedy

- struct MobilityFlags
- struct Mobility
- struct LegalMove
- class Mobilities

Enumerácie

- enum MovementType { Move , Attack , AttackMove }
- enum Castling { none , kingSide , queenSide }

5.15.1 Detailný popis

Definícia mobilít figúrok.

Autor

Martin Šváb

Dátum

Máj 2024

5.15.2 Dokumentácia enumeračných typov

5.15.2.1 Castling

`enum Castling`

Enumerátor reprezentujúci typ rošády.

Hodnoty enumerácií

none	Pohyb nie je rošáda.
kingSide	Pohyb je rošáda z kráľovej strany.
queenSide	Pohyb je rošáda z kráľovninej strany.

5.15.2.2 MovementType

`enum MovementType`

Enumerátor reprezentujúci typ mobility.

Hodnoty enumerácií

Move	Tento pohyb umožňuje len presunúť sa na prázdnu pozíciu.
Attack	Tento pohyb umožňuje len napadnúť protivníkovu figúrku.
AttackMove	Tento pohyb umožňuje presunúť sa na prázdnu pozíciu a napadnúť protivníkovu figúrku.

5.16 mobility.h

Zobraziť dokumentáciu tohoto súboru.

```
00001 #pragma once
00002
00003 /*****************************************************************/
00011 #include <piece.h>
00012 #include <vector>
00013 #include <string>
00014
00015
00019 enum MovementType {
00023     Move,
00027     Attack,
00031     AttackMove
00032 };
00033
00037 enum Castling {
00041     none,
00045     kingSide,
00049     queenSide
00050 };
00051
00055 struct MobilityFlags {
00059     bool initiative = false;
00060
00064     bool hasty = false;
00065
00069     bool uninterruptible = false;
00070
00074     bool vigilant = false;
00075 };
00076
00077
00081 struct Mobility {
00085     MovementType type = Move;
00086
00090     int start_x = 0;
00091
00095     int start_y = 0;
00096
00100     int direction_x = 0;
00101
00105     int direction_y = 0;
00106
00110     int limit = 0;
00111
00115     MobilityFlags flags{};
00116 };
00117
00121 struct LegalMove {
00125     LegalMove();
00126
00136     LegalMove(int x1, int y1, int x2, int y2, Mobility mobility);
00137
00143     LegalMove(std::string value);
00144
00148     int x1 = 0;
00149
00153     int y1 = 0;
00154
00158     int x2 = 0;
00159
00163     int y2 = 0;
00164
00168     Mobility mobility;
00169
00173     PieceType promotion = Pawn;
00174
00178     Castling castling = none;
00179
00185     std::string toString();
00186
00193     std::string toStringWithFlags(PieceColor color);
00194 };
00195
00199 class Mobilities {
00200 public:
00204     static std::vector<Mobility> mobilityConfig[6][3];
00205
00212     static void printMobilities(PieceType type, PieceEssence essence);
00213 };
```

5.17 Dokumentácia súboru piece.h

Definícia šachových figúrok.

```
#include <string>
```

Triedy

- struct Piece
- struct Ghost

Enumerácie

- enum PieceColor { White , Black }
- enum PieceType {
  Pawn , Rook , Knight , Bishop ,
  Queen , King }
- enum PieceEssence { Classic , Red , Blue }

Funkcie

- char getPieceIndex (char color, char type)
- PieceColor stringToColor (std::string value)
- PieceType stringToType (std::string value)
- PieceEssence stringToEssence (std::string value)
- std::string colorToString (PieceColor value)
- std::string typeToString (PieceType value)
- std::string essenceToString (PieceEssence value)

Premenné

- PieceColor opponent [ ]
- char colorChars [ ]
- char essenceChars [ ]
- char typeChars [ ]

5.17.1 Detailný popis

Definícia šachových figúrok.

Autor

Martin Šváb

Dátum

Máj 2024

5.17.2 Dokumentácia enumeračných typov

5.17.2.1 PieceColor

```
enum PieceColor
```

Vlastník figúrky.

Hodnoty enumerácií

White	Biely hráč.
Black	Čierny hráč.

5.17.2.2 PieceEssence

```
enum PieceEssence
```

Esencia figúrky.

Hodnoty enumerácií

Classic	Klasická esencia.
Red	Červená esencia.
Blue	Modrá esencia.

5.17.2.3 PieceType

```
enum PieceType
```

Typ figúrky.

Hodnoty enumerácií

Pawn	Pešiak.
Rook	Veža.
Knight	Rytier.
Bishop	Strelec.
Queen	Kráľovná.
King	Kráľ.

5.17.3 Dokumentácia funkcií

5.17.3.1 colorToString()

```
std::string colorToString (
            PieceColor value )
```

Konverzia farby figúrky na slovo.

Parametre

value	Farba šachovej figúrky.

Návratová hodnota

Slovo reprezentujúce farbu figúrky.

5.17.3.2 essenceToString()

```
std::string essenceToString (
            PieceEssence value )
```

Konverzia esencie figúrky na slovo.

Parametre

value	Esencia šachovej figúrky.

Návratová hodnota

Slovo reprezentujúce esenciu figúrky.

5.17.3.3 getPieceIndex()

```
char getPieceIndex (
            char color,
            char type )
```

Táto funkcia vráti id figúrky.

Parametre

color	Farba šachovej figúrky.
type	Typ šachovej figúrky.

Návratová hodnota

Identifikátor šachovej figúrky

5.17.3.4 stringToColor()

```
PieceColor stringToColor (
            std::string value )
```

Konverzia stringu na farbu figúrky.

Parametre

value	Slovo reprezentujúce farbu figúrky.

Návratová hodnota

Farba šachovej figúrky.

5.17.3.5 stringToEssence()

```
PieceEssence stringToEssence (
            std::string value )
```

Konverzia stringu na esenciu figúrky.

Parametre

value	Slovo reprezentujúce esenciu figúrky.

Návratová hodnota

Esencia šachovej figúrky.

5.17.3.6 stringToType()

```
PieceType stringToType (
            std::string value )
```

Konverzia stringu na typ figúrky.

Parametre

value	Slovo reprezentujúce typ figúrky.

Návratová hodnota

Typ šachovej figúrky.

5.17.3.7 typeToString()

```
std::string typeToString (
            PieceType value )
```

Konverzia typu figúrky na slovo.

Parametre

value	Typ šachovej figúrky.

Návratová hodnota

Slovo reprezentujúce typ figúrky.

5.17.4 Dokumentácia premenných

5.17.4.1 colorChars

```
char colorChars[]  [extern]
```

Zoznam znakov reprezentujúci farbu figúrky.

5.17.4.2 essenceChars

```
char essenceChars[]  [extern]
```

Zoznam znakov reprezentujúci esenciu figúrky.

5.17.4.3 opponent

```
PieceColor opponent[]  [extern]
```

Zoznam, ktorý umožňuje konverziu na protivníkovu farbu figúrky.

5.17.4.4 typeChars

```
char typeChars[]  [extern]
```

Zoznam znakov reprezentujúci typ figúrky.

5.18 piece.h

Zobraziť dokumentáciu tohoto súboru.

```
00001 #pragma once
00002
00003 /*****************************************************************/
00011 #define NOMINMAX
00012 #include <string>
00013
00014
00018 enum PieceColor {
00022     White,
00026     Black
00027 };
00028
00032 enum PieceType {
00036     Pawn,
00040     Rook,
00044     Knight,
00048     Bishop,
00052     Queen,
00056     King
00057 };
00058
00062 enum PieceEssence {
00066     Classic,
00070     Red,
```

```
00074     Blue
00075 };
00076
00080 struct Piece {
00084     PieceColor color = White;
00085
00089     PieceType type = Pawn;
00090
00094     PieceEssence essence = Classic;
00095
00101     char toChar();
00102 };
00103
00107 struct Ghost {
00111     int x = -1;
00112
00116     int y = -1;
00117
00121     int parentX = -1;
00122
00126     int parentY = -1;
00127 };
00128
00132 extern PieceColor opponent[];
00133
00137 extern char colorChars[];
00138
00142 extern char essenceChars[];
00143
00147 extern char typeChars[];
00148
00156 char getPieceIndex(char color, char type);
00157
00164 PieceColor stringToColor(std::string value);
00165
00172 PieceType stringToType(std::string value);
00173
00180 PieceEssence stringToEssence(std::string value);
00181
00188 std::string colorToString(PieceColor value);
00189
00196 std::string typeToString(PieceType value);
00197
00204 std::string essenceToString(PieceEssence value);
```

5.19 Dokumentácia súboru random.h

Generácia náhodných čísiel.

```
#include <random>
#include <vector>
```

Triedy

- class Random

5.19.1 Detailný popis

Generácia náhodných čísiel.

Autor

Martin Šváb

Dátum

Máj 2024

5.20 random.h

Zobraziť dokumentáciu tohoto súboru.

```
00001 /*****************************************************************/
00009 #pragma once
00010
00011 #include <random>
00012 #include <vector>
00013
00014
00018 class Random {
00019 public:
00023     static void initSeed();
00024
00030     static bool coinFlip();
00031
00039     static int generateRandomNumber(int lowerBound, int upperBound);
00040
00047     template <typename T>static T getRandomElement(const std::vector<T>& vec);
00048 private:
00052     static std::mt19937 gen;
00053 };
00054
00055 template<typename T>
00056 inline T Random::getRandomElement(const std::vector<T>& vec)
00057 {
00058     if (vec.empty()) {
00059         return T();
00060     } else {
00061         return vec[generateRandomNumber(0, vec.size() - 1)];
00062     }
00063 }
```

5.21 Dokumentácia súboru searching.h

Prehľadávanie šachovnice.

```
#include <board.h>
#include <transpositions.h>
#include <chrono>
#include <ensemble.h>
```

Triedy

- struct SearchArgs
- struct PerformanceArgs
- class SearchManager

5.21.1 Detailný popis

Prehľadávanie šachovnice.

Autor

Martin Šváb

Dátum

Máj 2024

5.22 searching.h

Zobraziť dokumentáciu tohoto súboru.

```
00001 #pragma once
00002
00003 /*****************************************************************/
00011 #include <board.h>
00012 #include <transpositions.h>
00013 #include <chrono>
00014 #include <ensemble.h>
00015
00016
00020 struct SearchArgs {
00024     int curDepth = 0;
00025
00029     int maxDepth = 0;
00030
00034     int alpha = INT_MIN;
00035
00039     int beta = INT_MAX;
00040 };
00041
00045 struct PerformanceArgs {
00049     std::chrono::high_resolution_clock::time_point start = std::chrono::high_resolution_clock::now();
00050
00054     int positionsCur = 0;
00055
00059     int positionsTotal = 0;
00060
00064     long long durationTotal = 0;
00065
00069     static std::mutex mutex;
00070
00074     void printPerformance();
00075
00082     void printPerformance(std::chrono::high_resolution_clock::time_point stop, long long durationCur);
00083 };
00084
00088 class SearchManager {
00089 public:
00093     static TranspositionCache cache;
00094
00098     static Ensemble ensemble;
00099
00109     static std::pair<bool, LegalMove> calculateBestMove(Board board, int depth, bool useEnsemble, bool
      debug);
00110
00121     static std::pair<bool, LegalMove> calculateBestMove_threads(Board board, int depth, int
      threadCount, bool useEnsemble, bool debug);
00122
00134     static int minimax(Board board, PieceColor playerColor, SearchArgs searchArgs, PerformanceArgs*
      performanceArgs, bool useEnsemble, bool debug);
00135 private:
00139     static const int qLimit = 0;
00140 };
```

5.23 Dokumentácia súboru simulation.h

Prostredie na simuláciu šachových hier medzi enginom Malakh a FairyStockfish.

```
#include <board.h>
```

Triedy

- class SimulationManager

5.23.1 Detailný popis

Prostredie na simuláciu šachových hier medzi enginom Malakh a FairyStockfish.

Autor

Martin Šváb

Dátum

Máj 2024

5.24 simulation.h

Zobraziť dokumentáciu tohoto súboru.

```
00001 /*****************************************************************/
00009 #pragma once
00010
00011 #include <board.h>
00012
00013
00017 class SimulationManager {
00018 public:
00030     static void simulateGames(int gameCounter, EssenceArgs essenceArgs, int malakhDepth, int
      fairyStockfishDepth, bool useEnsemble, bool useDB, std::string outputFilename);
00031 };
```

5.25 Dokumentácia súboru transpositions.h

Implementácia transpozičných tabuliek pre zrýchlenie prehľadávania.

```
#include <unordered_map>
#include <mutex>
```

Triedy

- struct Transposition
- class TranspositionCache

5.25.1 Detailný popis

Implementácia transpozičných tabuliek pre zrýchlenie prehľadávania.

Autor

Martin Šváb

Dátum

Máj 2024

5.26 transpositions.h

Zobraziť dokumentáciu tohoto súboru.

```
00001 /*****************************************************************/
00009 #pragma once
00010
00011 #include <unordered_map>
00012 #include <mutex>
00013
00014
00018 struct Transposition {
00022     int value = 0;
00023
00027     int depth = -1;
00028 };
00029
00033 class TranspositionCache {
00034 public:
00040     TranspositionCache(int capacity);
00041
00048     Transposition get(unsigned long long key);
00049
00056     void put(unsigned long long key, Transposition value);
00057 private:
00061     std::unordered_map<unsigned long long, Transposition> cache;
00062
00066     std::list<unsigned long long> order;
00067
00071     int capacity;
00072
00076     std::mutex* mutex;
00077 };
```

5.27 Dokumentácia súboru uci.h

UCI protokol umožňuje komunikáciu medzi našim enginom a GUI alebo inými enginami.

```
#include <searching.h>
#include <string>
```

Triedy

- class uci

5.27.1 Detailný popis

UCI protokol umožňuje komunikáciu medzi našim enginom a GUI alebo inými enginami.

Autor

Martin Šváb

Dátum

Máj 2024

5.28 uci.h

Zobraziť dokumentáciu tohoto súboru.

```
00001 #pragma once
00002
00003 /*****************************************************************/
00011 #include <searching.h>
00012 #include <string>
00013
00014
00018 class uci {
00019 public:
00023     void run();
00024
00031     bool parseCommand(std::string command);
00032 private:
00036     EssenceArgs essenceConfig;
00037
00041     Board board;
00042
00046     bool useEnsemble = false;
00047 };
```

Register