

Chess 1.5 Fejlesztői Dokumentáció

Futtatási környezet:

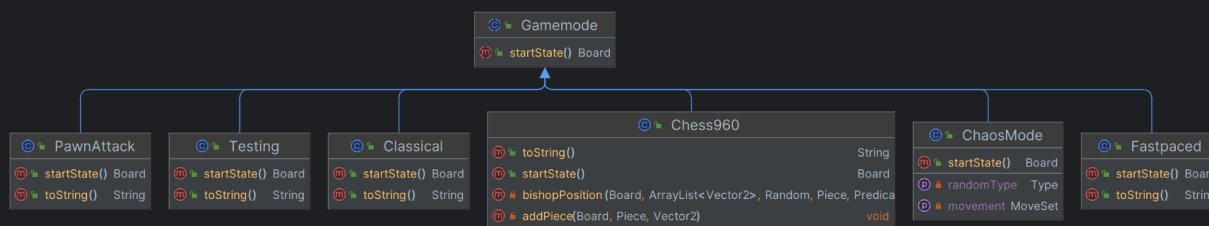
A *Chess 1.5* a *Java 17.0.2* verzióban *Maven* build tool használatával lett fejlesztve és tesztelve, az *IntelliJ IDEA* integrált fejlesztői környezet segítségével. Az applikáció a *JavaFX* könyvtárat használja a grafikai felületeken.

A *JavaFX* egy könnyű, gyors, bővíthető, és modern grafikai felület. A *Chess 1.5* applikáció hibrid módon használja, fxml fájlok és kód segítségével, a dinamikus megjelenítés érdekében.

Rendszer követelmények: legalább 4 magos processzor, és 8 GB memória.

Függőségek:

- *Javafx*: A grafikai megjelenítés
- *Json-simple*: A Json fájlok olvasásához
- *JUnit 4*: A teszteléshez



A **Gamemode** egy absztrakt osztály amiből az összes játékmód származik. Egy metódusa van a **startState()** amelyet minden osztály felülír, és ez felel a játéktábla felállításáért.

A legtöbb játékmód egy *JSON* fájlból tölti be a tábla felállását ezért ezek külön nem szerepelnek.

A *Chess960* játékmódhoz a tábla hátsó sorait össze kell keverni bizonyos szabályok szerint:

- A sötét futó a 4 sötét mező egyikére kerül véletlenszerűen.
- A világos futó a 4 világos mező egyikére kerül véletlenszerűen.
- A 2 huszár a maradék helyeken véletlenszerűen kerül elhelyezésre a vezérrel együtt
- A maradék helyre bástya-király-bástya hármas kerül ebben a sorrendben.

Ez a felállítás tükrözve van a mind a két játékosnak. A **bishopPosition()** metódus segít a futók elhelyezésében, az **addPiece()** pedig a táblára teszi fel a bábukat.

A chaos módban az összes bábu kinézete és mozgás készlete is véletlenszerűen van generálva. Minden pozíció tükrözve van a másik játékos számára. A **randomType()** egy véletlenszerű kinézetet ad vissza a bábunak, míg a **movement()** egy véletlenszerű mozgás készletet ad vissza.

Mindenhol a véletlenszerű választás egy kriptográfiai random **SecureRandom** generátorral van megoldva, mivel a sima **Random** nem volt elég gyors és így sokszor ugyan azt adta vissza.

A **toString** metódus csak tesztelés és fejlesztés közben van használva.

<div><div>SettingsMenuController</div><div><div>SettingsMenuController()</div><div><div><div><div><div>fastpacedButton</div><div>Button</div></div><div><div>IDLE_CHESS960_BUTTON_STYLE</div><div>String</div></div><div><div>backButton</div><div>Button</div></div><div><div>IDLE_CLASSICAL_BUTTON_STYLE</div><div>String</div></div><div><div>IDLE_TESTING_BUTTON_STYLE</div><div>String</div></div><div><div>gameTimerEnabled</div><div>boolean</div></div><div><div>gameModesScroll</div><div>ScrollPane</div></div><div><div>testingButton</div><div>Button</div></div><div><div>isCastingEnabled</div><div>boolean</div></div><div><div>enpassantCheckBox</div><div>CheckBox</div></div><div><div>secondsSpinner</div><div>Spinner<Integer></div></div><div><div>chess960Button</div><div>Button</div></div><div><div>playButton</div><div>Button</div></div><div><div>promotionCheckBox</div><div>CheckBox</div></div><div><div>gameTimerMinutes</div><div>Integer</div></div><div><div>pawnAttackButton</div><div>Button</div></div><div><div>height</div><div>int</div></div><div><div>HOVERED_BUTTON_STYLE</div><div>String</div></div><div><div>bgPane</div><div>Pane</div></div><div><div>IDLE_CHAOS_MODE_BUTTON_STYLE</div><div>String</div></div><div><div>minutesSpinner</div><div>Spinner<Integer></div></div><div><div>isPromotionEnabled</div><div>boolean</div></div><div><div>calcHeight</div><div>int</div></div><div><div>castingCheckBox</div><div>CheckBox</div></div><div><div>gameTimerSeconds</div><div>Integer</div></div><div><div>classicalButton</div><div>Button</div></div><div><div>IDLE_FASTPACED_BUTTON_STYLE</div><div>String</div></div><div><div>IDLE_PAWNATTACK_BUTTON_STYLE</div><div>String</div></div><div><div>chaosModeButton</div><div>Button</div></div><div><div>selectedGameMode</div><div>String</div></div><div><div>timerCheckBox</div><div>CheckBox</div></div><div><div>isEnPassantEnabled</div><div>boolean</div></div><div><div>onClassicalSelected()</div><div>void</div></div><div><div>onCastingEnableTicked()</div><div>void</div></div><div><div>onChess960Selected()</div><div>void</div></div><div><div>setUpButton(String, String, String, int)</div><div>Button</div></div><div><div>onBackButtonPressed()</div><div>void</div></div><div><div>onPlayButtonPressed()</div><div>void</div></div><div><div>onFastpacedSelected()</div><div>void</div></div><div><div>disableEverything()</div><div>void</div></div><div><div>onPromotionEnableEnableTick()</div><div>void</div></div><div><div>setSelectedGameModeButton()</div><div>void</div></div><div><div>onPawnAttackSelected()</div><div>void</div></div><div><div>onTimerEnableTicked()</div><div>void</div></div><div><div>onTestingSelected()</div><div>void</div></div><div><div>initialize()</div><div>void</div></div><div><div>onEnPassantEnableTicked()</div><div>void</div></div><div><div>onChaosModeSelected()</div><div>void</div></div></div></div></div></div></div>	<div><div>UInterface</div><div><div>setCheck(Vector2)</div><div>void</div></div><div><div>remove(Vector2, Piece)</div><div>void</div></div><div><div>promote(Color, Vector2)</div><div>void</div></div><div><div>addPiece(Piece, Vector2)</div><div>void</div></div><div><div>endGame(Color, WinReason)</div><div>void</div></div></div> <div><div>ChessController</div><div><div>ChessController()</div><div><div><div><div><div>blackTakenScroll</div><div>ScrollPane</div></div><div><div>inputText</div><div>TextField</div></div><div><div>pressedHandler</div><div>EventHandler<MouseEvent></div></div><div><div>moveListElement</div><div>ListView<String></div></div><div><div>clockPane</div><div>Pane</div></div><div><div>engine</div><div>EngineInterface</div></div><div><div>main</div><div>Pane</div></div><div><div>whiteTakenScroll</div><div>ScrollPane</div></div><div><div>PROMOTIONPIECES</div><div>ArrayList<Piece></div></div><div><div>chessBoardPane</div><div>Pane</div></div></div></div><div><div>remove(Vector2, Piece)</div><div>void</div></div><div><div>setUpBoard()</div><div>void</div></div><div><div>addPiece(Piece, Vector2)</div><div>void</div></div><div><div>handlePieceClick(Vector2)</div><div>void</div></div><div><div>handleTakenList()</div><div>void</div></div><div><div>aiPromote(Vector2)</div><div>void</div></div><div><div>playerColor()</div><div>Color</div></div><div><div>removeFromTo()</div><div>void</div></div><div><div>movePiece(Vector2, Vector2)</div><div>void</div></div><div><div>handleTextMove(String)</div><div>void</div></div><div><div>handleTimerUpdate(Color)</div><div>void</div></div><div><div>displayPossibleMoves(ArrayList<Vector2>, Vector2)</div><div>void</div></div><div><div>generateMoveString(Move)</div><div>String</div></div><div><div>addClickEventToPiece(ImageView)</div><div>void</div></div><div><div>displayFromTo(Vector2, Vector2)</div><div>void</div></div><div><div>switchAiMoveColor()</div><div>void</div></div><div><div>addTaken(Piece)</div><div>void</div></div><div><div>updateMoveList(Move)</div><div>void</div></div><div><div>generateAdditional(StringBuilder, Move)</div><div>void</div></div><div><div>removePossibleMoves()</div><div>void</div></div><div><div>endGame(Color, WinReason)</div><div>void</div></div><div><div>setCheck(Vector2)</div><div>void</div></div><div><div>updateClickEventToPiece(Vector2)</div><div>void</div></div><div><div>promote(Color, Vector2)</div><div>void</div></div><div><div>clearTaken()</div><div>void</div></div><div><div>getKeysByValue(Map<T, E>, E)</div><div>List<T></div></div><div><div>initialize()</div><div>void</div></div><div><div>displayTaken()</div><div>void</div></div></div></div></div>	<div><div>Constants</div><div><div>Constants()</div><div><div><div><div><div>promotionList</div><div>HashMap<ImageView, Piece></div></div><div><div>logger</div><div>Logger</div></div><div><div>timerHBox</div><div>HBox</div></div><div><div>whiteTimerLabel</div><div>Label</div></div><div><div>isRunning</div><div>boolean</div></div><div><div>possibleMoves</div><div>HashMap<Vector2, ImageView></div></div><div><div>takenPieces</div><div>ArrayList<Piece></div></div><div><div>fromToMoves</div><div>HashMap<Vector2, ImageView></div></div><div><div>blackTimerRunOut</div><div>boolean</div></div><div><div>promotionUIBase</div><div>Pane</div></div><div><div>logFileHandler</div><div>FileHandler</div></div><div><div>FASTPACEDTIMEOUT</div><div>int</div></div><div><div>pieces</div><div>HashMap<Vector2, ImageView></div></div><div><div>AlgColor</div><div>Color</div></div><div><div>takenList</div><div>HashMap<Piece, ImageView></div></div><div><div>playedMoves</div><div>ArrayList<Move></div></div><div><div>endGameBase</div><div>StackPane</div></div><div><div>whiteTimerBox</div><div>Pane</div></div><div><div>timerThread</div><div>Thread</div></div><div><div>whiteToMove</div><div>boolean</div></div><div><div>DEVMODE</div><div>boolean</div></div><div><div>whiteTaken</div><div>Pane</div></div><div><div>whiteTimeInMills</div><div>long</div></div><div><div>pauseForPromotion</div><div>boolean</div></div><div><div>board</div><div>Board</div></div><div><div>alg</div><div>AlgorithmInterface</div></div><div><div>whiteSide</div><div>boolean</div></div><div><div>algMoveThreads</div><div>Thread</div></div><div><div>blackTimeInMills</div><div>long</div></div><div><div>fastPacedCounter</div><div>int</div></div><div><div>blackTimerBox</div><div>Pane</div></div><div><div>whiteTimerRunOut</div><div>boolean</div></div><div><div>blackTaken</div><div>Pane</div></div><div><div>blackTimerLabel</div><div>Label</div></div></div></div></div></div></div> <div><div>AudioPlayer</div><div><div>AudioPlayer()</div><div><div><div><div>playMoveSound()</div><div>void</div></div><div><div>playCaptureSound()</div><div>void</div></div><div><div>playStartSound()</div><div>void</div></div></div></div></div></div>	<div><div>TimerInit</div><div><div>TimerInit()</div><div><div><div><div>initFastPacedThread(ChessController, EngineInterface)</div><div>void</div></div><div><div>initThread(ChessController)</div><div>void</div></div><div><div>formatTime(long)</div><div>String</div></div><div><div>initStyles(ChessController)</div><div>void</div></div></div></div></div></div>
<div><div>General</div><div><div>General()</div><div><div><div><div><div>changeScene(Stage)</div><div>void</div></div><div><div>convertChessCoordToMove(String)</div><div>Vector2</div></div><div><div>threadStop()</div><div>void</div></div><div><div>reset(ChessController, EngineInterface)</div><div>void</div></div><div><div>getMoveFromText(String, EngineInterface)</div><div>Move</div></div><div><div>addEventHandlers(Scene, EventHandler<KeyEvent>[])</div><div>void</div></div><div><div>getPieceTypeString(Piece)</div><div>String</div></div><div><div>getPieceColorString(Piece)</div><div>String</div></div><div><div>guiReset(ChessController, KeyEvent, EngineInterface)</div><div>void</div></div><div><div>convertMoveToChessCord(Vector2)</div><div>String?</div></div></div></div></div></div></div>	<div><div>MainMenuController</div><div><div>MainMenuController()</div><div><div><div><div><div>IDLE_BUTTON_STYLE</div><div>String</div></div><div><div>HOVERED_BUTTON_STYLE</div><div>String</div></div><div><div>multiplayerButton</div><div>Button</div></div><div><div>aiButton</div><div>Button</div></div></div></div><div><div>initialize()</div><div>void</div></div><div><div>onAiButtonPressed()</div><div>void</div></div><div><div>onMultiplayerButtonPressed()</div><div>void</div></div></div></div></div>		

A ui nál a következő osztályok voltak használva.

Minden grafikus felület *javafx* el készült, és fxml fájlokból van betöltve az alapjuk.

Minden fxml hez tartozik egy kontroller ami a funkcionalitás ért felel. Az **initialize()** metódus az fxml megnyitásakor automatikusan meghívásra kerül.

A **MainMenuController** a játék főmenüjében való funkcionalitás ért felel. A főmenübe két gomb van, a *multiplayer*, és a *singleplayer* vagy *ai* gomb. Ha a felhasználó megnyomja a multipalyer gombot akkor az **onMultiplayerButtonPressed()** metódus kerül meghívásra, amely megnyitja a játékmódok és beállítások menüt. Ha a

singleplayer gombot nyomja meg a felhasználó akkor az **onAiButtonPressed()** metódus kerül meghívásra, ami beállítja a **RuleSet**et és megnyitja a sakk játékműzöt.

A **SettingsMenuController** a játékműd választó és beállítások menű ért felel. A játékműdok választója dinamikusan kerül a kijelzőre, a **setUpButton()** metódus segítségével az **initialize()** metódusból. Ha a felhasználó rákattint valamelyik játékműdra akkor az meghívja a **on<gamemode>Selected()** metódust, amely beállítja a megfelelő alapokat ahhoz a játékműdhoz, és meghívja a

setSelectedGamemodeButton() metódust, ami a stílust kezeli. A **disableEverything()** metódus mindent egy előre meghatározott értékre állít, ezt később a játékműdok felül írhatják. Ha a felhasználó megnyomja a *Back* gombot akkor az meghívja az **onBackButtonPressed()** metódust, ami betölti a fő menűt. Az összes *checkbox* ami a beállítások oldalon van meghív egy **on<Setting>EnableTicked()** metódust ami ezt egy változóra fordítja átt. A két spinner egy *value factory*-t használ megadott minimum és maximum értékekkel. Ha a felhasználó megnyomja a *Play* gombot akkor az **onPlayButtonPressed()** metódus betölti a kiválasztott játékműdot a **RuleSet**be a többi beállítással együtt, majd betölti a sakk játékműzöt.

A **Constants** az összes változót tárolja, ami a játék folyamán több class ból is meg van hívva.

A **General** olyan metódusokat tartalmaz amelyeket többször is meg hívunk de nem létfontosságúak a felhasználói felülethez. A **changeScene()** metódus a fő menűt tölti be ha a felhasználó a játék végén megnyomja a return to menu gombot. A két metódus (**convertMoveToChessCoord()**, **convertChessCoordToMove()**) a sakk által használt olvasható koordináta rendszer és a *Chess 1.5* által használt **Vector2** koordináta rendszer között vált.

A **reset()** és **guiReset()** metódusok a játékot visszaállítják, ha ugyan azokkal a szabályokkal szeretne a felhasználó játszani. A **threadStop()** az alkalmazásból való kilépésnél megállítja az összes használt szálat. A **getPieceTypeString()** és **getPieceColorString()** egy bizonyos bábusnak az adatait adja vissza szövegként. A **getMoveFromText()** pedig egy lépést ad vissza a betáplált szöveg alapján.

A **TimerInit** osztály a sakk időzítő beállításáért felelős. Az **initStyles()** metódus a grafikai alapokat állítja fel az órának, míg az **initThread()** az óra működéséért felelős szálat állítja fel. Az **initFastPacedThread()** egy olyan szálat állít fel ami 2 másodpercenként meg hív egy lép metódust, ami véletlenszerűen lép a felhasználó helyett. A **formatTime()** csak visszaad egy olvasható idő értéket egy long ban tárolt időből. Az időt egy longban tároljuk mivel ezredmásodpercekben tároljuk az időt pontosság miatt. Az **AudioPlayer** felel a hangok lejátszásáért. A **play<soundName>Sound()** metódus megkeresi és lejátsza a hangot. Három féle hangot támogat, egy lépés, egy leütés, és egy játék indítás hangot. A hang fájlokat a *lichess.com* online sakk oldalról vettük fel. A **UIInterface** definiálja azokat a metódusokat a sakk játék felületen, amiket az engine meg tud hívni, bizonyos esetekben.

A `ChessController` a sakk játék felület funkcionalitását kezeli, és a `UIInterface`t implementálja. Az `initialize()` metódus létrehoz egy sakk motort és felállítja a táblát a kiválasztott játék módnak megfelelően. A `remove` metódussal egy bábút lehet levenni a tábláról, és ha megadjuk neki a bábunak a referenciáját is akkor hozzáadja a leütött bábuk kijelzőhöz. A `promote()` metódus felajánlja a felhasználónak a promócióhoz választható bábukat ez lehet a vezér, bástya, huszár, és futó. Majd kiválasztás után hozzáadja a táblához. Az `aiPromote()` akkor kerül meghívásra ha az algoritmus promócióhoz jut, akkor automatikusan a promóciós bábu a vezér lesz. Az `addPiece()` metódussal a táblához tudunk hozzáadni bábút. Az `endGame()` metódus pedig a játék vége kijelzőt jeleníti meg a sakktábla helyén. Így az óra a leütött bábuk és a lépés előzmények még láthatók a jobb oldalon. A lépés logikáért a `movePiece()` felel. Ez a metódus továbbá kezeli a leütést, és a sakk algoritmus lépésének meghívását. A leütött bábu kijelző kezeléséért a `displayTaken()`, `addToTaken()`, `clearTaken()`, és `handleTakenList()` metódusok felelnek, ebben a sorrendben kijelzi a kész listát, hozzáad a listához, törli a listát, és kezeli a folyamatot. A lépés előzmények kijelzik a játékos előnyét és hátrányát, anyag pontokban mérve. A kezeléséhez `generateAdditional()` legenerálja az előny hátrány szövegeket, a `generateMoveString()` létrehoz a szöveget egy lépésből, az `updateMoveList()` pedig frissíti a lista nézetet. A `handleTextMove()` metódus felel a chat ablakon beérkező parancsok és lépések kezelésében és értelmezésében. A `displayFormTo()` metódus a legutóbbi lépés kezdő és vég pozícióját jelöli meg sárga négyzetekkel, míg a `displayPossibleMoves()` a kiválasztott bábuk lehetséges lépéseit jelzi ki. A `handlePieceClick()` és `updateClickEventToPiece()` metódusok a kattintásért felelősek.

A `WinReason` enum a játék vége `endGame()` metódusba vannak használva, azt mutatják hogy mi okozta a játék végét.

A `JsonToBoard` felelős azért hogy beolvassa a json fájlt és egy `Board`ot csináljak belőle amit feltölt a megfelelő helyen a megadott bábukkal. A `getPieceType()` metódus egy szöveg alapján visszaad egy nézetet a bábunak, míg a `getMoveSet()` egy szöveg bemenetből, egy `MoveSet`et ad vissza. A `jsonToBoard()` metódus statikus, és a neki megadott json fájlból fogja betölteni az adatokat.

A `BoardVisualizr` a fejlesztés és tesztelés során van használva, a konzolon szöveges reprezentációban megmutatja a jelenlegi tábla felállást. A `printBoard()` metódusnak megadott Board lesz kiírva a konzolra. A `getColorChar()` egy `Piece` ből kiveszi a színét és egy karakterként visszaadja, míg a `getLookChar()` a kinézetét adja vissza egy karakterként.

A `customFormatter()` a log fájlok formázásáért felel. Egy log fájl csak akkor készül ha az applikációban be van kapcsolva a fejlesztői mód.

A tábla reprezentálása

A sakktáblát a **Board** osztály reprezentálja.

Egy 8x8-as méretű kétdimenziós tömbben tárolja a tábla elemeit.

Az elemek **Boardelement** típusúak.

Az elemeket lekérni a táblából **int**ekkel vagy **Vector2** típussal lehetséges.

A **Boardelement** osztály a sakktábla elemeit valósítja meg.

Egy **Boardelement** lehet üres, üres mező esetén,
vagy lehet a nem üres leszármazottja a **Piece**.

A táblán lévő sakkbábuk **Piece** típusúak.

Egy bábut a következő tulajdonságai határoznak meg:

color	a bábu színe	Color felsorolási típus
look	a kinézete	Look felsorolási típus
movement	a mozgáskészlete	Moveset
isKing	a bábu-e a király	boolean

Ezekon felül egy bábuhoz tartozik még a **pin** **Vector2** érték, ami a bábu kötésének az irányát tárolja (**null** értéket, ha nincs kötésben).

Valamint egy **boolean** típusú változó, amire a motorban van szükség az *en passant* és a *sáncolás* lehetőségének számításához.

A bábuk lépéskészlete a **Moveset** osztályból származtatott *singleton* osztályok valamelyike.

Ennek a megoldásnak köszönhetően a bábuk kinézetei és lépései elkülöníthetők, ezáltal új játékmódokra van lehetőség.

A lépéskészletek paraméterei a mozgások és támadások támadások irányait tároló **moves** és **attacks** **Arraylist**ek.

Alapértelmezetten a **moves** tárolja a támadásokat is, az **attacks**ra az olyan bábuk esetén van szükség, amik eltérő helyekre ütnek és lépnek (paraszt). Ez esetben az **attackDifferent** mező igaz értékre állítandó.

Továbbá a **whiteDifferent** mezővel jelezhető, ha a fehér bábunak fordítottak a lépési, a **repeating** pedig az, ha a bábu csúszik, tehát a lépések iránya ismétlődik.

A **Move** osztály lépések tárolására alkalmas.

Egy lépéshez minden esetben szükséges a kiinduló és végpontja, ezek **Vector2** típusúak.

Emellett egy lépéshez nem kötelezően tartozhat egy **color** és **score**, az előbbire a vizuális megjelenítéshez, az utóbbira a sakkozó algoritmusban történő előrendezésben van szükség.

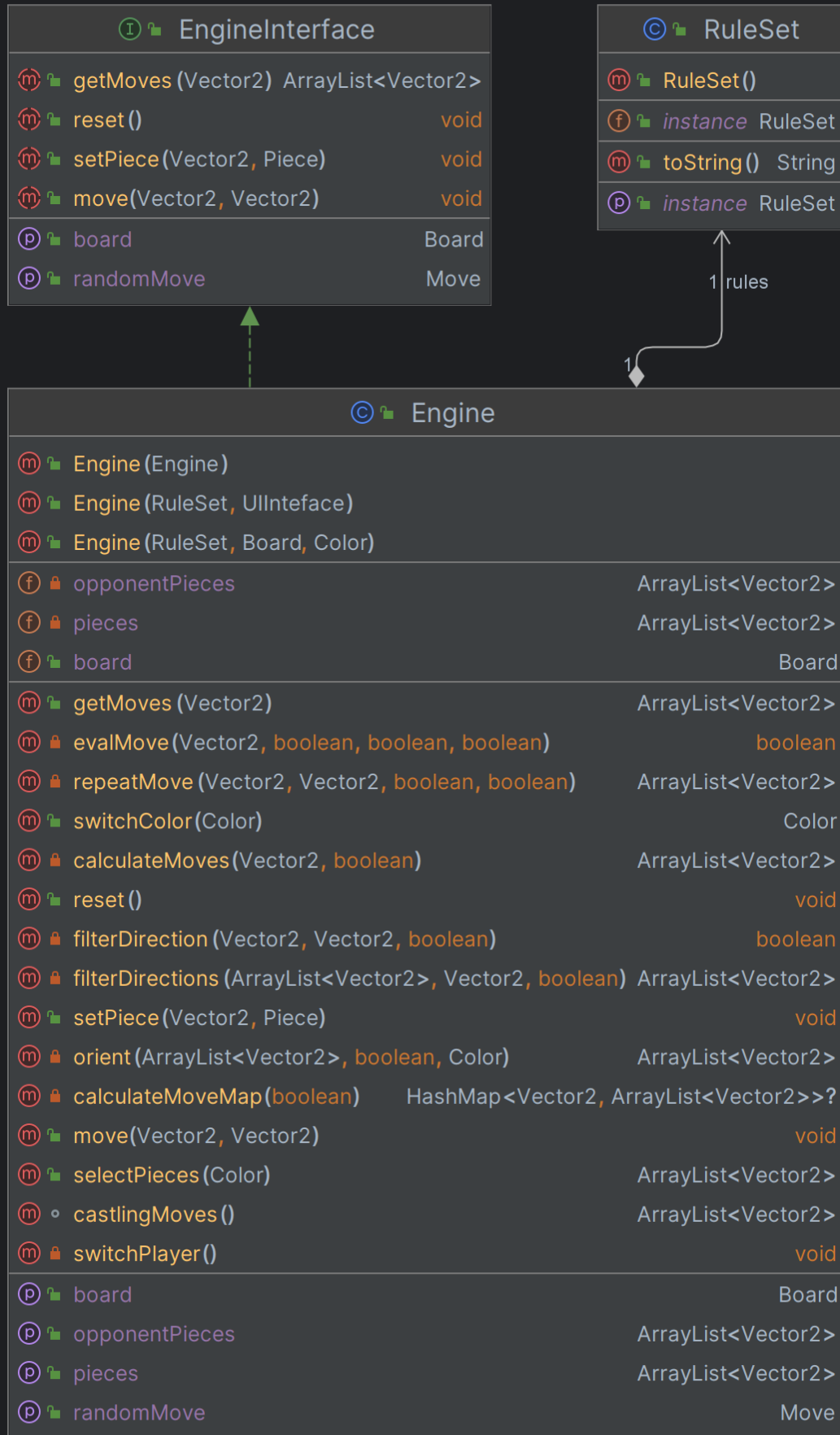
A **Vector2** osztály kétdimenziós vektorok használatát teszi lehetővé.

Alapvetően egy **x** és egy **y** **int** típusú koordinátát tárol.

Különböző metódusokat is tartalmaz, amivel műveleteket végezhetünk egy vagy több **Vector2** objektumon.

Ezek részletes leírása a **Javadoc**ban megtekinthető.

A sakkmotor



Az **Engine** osztállyal legális lépések számíthatók ki és játszhatók le egy táblán. Az osztály az **EngineInterface** implementálja.

Az objektum példányosításához szükség van egy **RuleSet** típusú játékszabályokat tároló objektumra.

Emelezt szükséges megadni egy **UIInterface** implementáló osztályt vagy egy táblát és a lépéssel következő játékos színét.

Elérhető egy másoló *constructor* is, ami egy másik **Engine** objektumot másol.

A példányosítást követően a motor rendelkezik egy meglévő táblával, vagy pedig a szabályok alapján létrehoz egyet.

Ezt követi a legális lépések kiszámítása, hogy ezeket a **getMoves** metódussal bármelyik bábura le lehessen kérni.

A számítás első fázisa az ellenség által támadható mezők keresése.

Erre a király lépéseinek korlátozására és a sakkhelyzetekkel való elszámolás miatt van szükség.

A **calculateMoveMap** metódus kerül meghívásra, *onlyAttacks* paramétere igaz értéket kap.

A számítás során az ellenség összes bábujának összes lehetséges lépése bekerül az *previousAttacks Hashmap*-be, amennyiben nem esnek ki a tábláról.

A csúszó bábuk esetében minden irány lépései az első szembejövő bábuval záródóan kerülnek be, de ha a következő bábu az adott irányban a király, akkor az előző bábu kötésbe kerül.

Az ellenség támadásainak ismeretében kezdődik meg az aktuális játékos lépéseinek kiszámítása.

A számítást ismét a **calculateMoveMap** metódus végzi, a *possibleMoves Hashmap*-et tölti fel.

Először a király lépései kerülnek felmérésre, itt csak olyan pozíciók elérhetőek, amiket az ellenség nem támad (hiszen ekkor a király leüthető lenne a következő lépésben)

Amennyiben a király már támadás alatt (tehát sakokban) van, rögzítésre kerül, hogy honnan érkezik ez a támadás.

Ha több mint egy bábu tartja sakokban a királyt, akkor a többi bábu lépései mind illegálisak. Be kell látnunk, hogy ilyen helyzetben csak akkor oldhatjuk fel a sakkot, ha a királlyal lépünk ki belőle.

Ellenkező esetben feloldható a sakk a király és az őt sakokban tartó bábu közötti mezőkre lépéssel, beleértve a leütést is.

A bábuk lépéseinek kiszámítása azzal az egy extra feltétellel történik, hogy a kötésben lévő bábuk csak a kötés irányában mozoghatnak, beleértve a leütést is. Ezt az irányt a `pin` mezőjük tárolja.

Az `Engine` a lépéseket a táblán a `move` metódussal végzi.

A metódus két paramétere a lépést végző bábu koordinátáit tároló `from`, és a lépés végpontját tároló `to` `Vector2` objektumok.

A lépés során a bábu átkerül a kiindulópontjából a végpontjába. Leütés esetén a leütött bábút felülírja a mozdított bábu.

Promóció esetén az `Engine` értesíti a `UIRef` változóval vonatkoztatott osztályt, null érték esetén pedig az egyszerűség kedvéért *királynővé* lépteti elő a *parasztot*.

Az *en passant* és a *sáncolás* során a táblán lévő többi bábu is megfelelően mozgásra/törlésre kerül.

A lépés befejeztével ismét számításra kerülnek a lehetséges támadások, majd az aktuális játékos színe cserélődik, ez a `nextPlayer` mező és a `switchPlayer` metódus segítségével történik, ezután a lehetséges lépések számítása következik.

A motor tartalmaz még egy `getRandomMove` metódust.

A metódus egy random `Move` típusú lépéssel tér vissza, amit az aktuális játékos tud meglépni.

Létezését a *FastPaced* játékmód tette szükségessé.

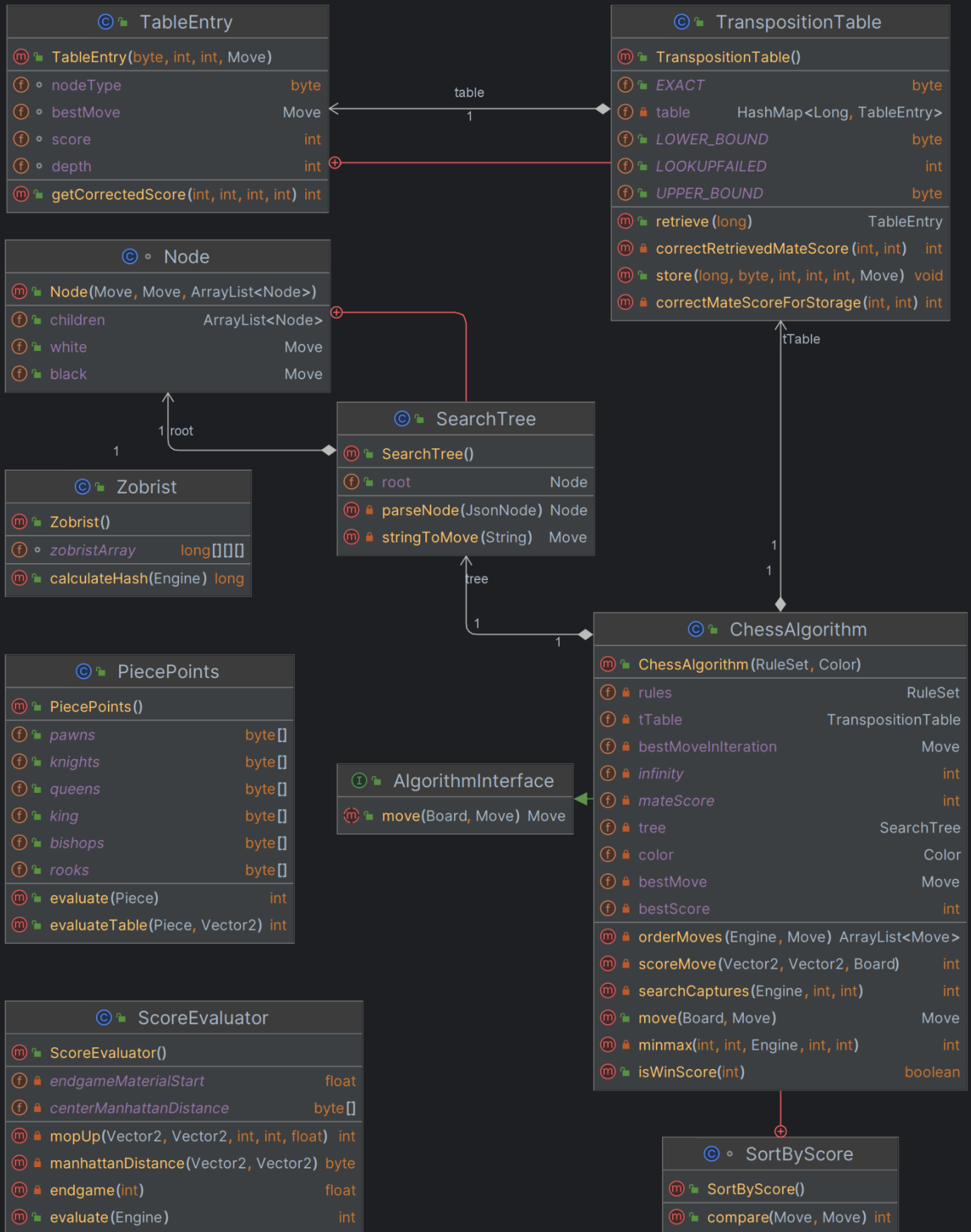
A random lépés kiválasztása során minden lépésnek egyenlő esélye van, nincs torzítás a több lépéssel rendelkező bábuk felé.

Az osztály többi metódusa a lépések számításához szükséges.

Az ismétlődő kódsorok mind külön metódusként szerepelnek, ezzel elősegítve a kódbázis átláthatóságát és olvashatóságát.

Ezek részletes leírása a [Javadoc](#)-ban megtekinthető.

A sakkozó algoritmus



A `ChessAlgorithm` osztály lehetővé teszi, hogy a *computer* ellen játszassuk a sakkjátékot.

A `move` metódusa egy tábla és a legutóbbi lépés alapján egy lépéssel tér vissza. Ennek meghatározás a következőképpen történik:

1. Tankönyvnek megfelelő lépések:

A bevált nyitásokat a sakkozó algoritmus egy előre elkészített adatbázis alapján játsza. A `SearchTree` osztály egy *keresőfa*, aminek minden pontja a világos játékos lépésre adható optimális választ tartalmazza.

A játék kezdetekor a sakkozó algoritmus a *keresőfa* alapján válaszol a felhasználó lépéseire, minden alkalommal a *fát* helyettesítve azzal a *részfával*, amelyik a világos lépésnek megfelel.

Amennyiben nincs meg az adott lépés az adatbázisban, a *fára* való hivatkozás, a *tree* mező értéke `null`-ra kerül és a megkezdődik a második fázis.

2. A legjobb lépés keresése

A keresés iteratív módon történik. Először egy, majd kettő, majd három... lépés mélységig játsza le az összes lehetséges lépést az algoritmus.

Három másodperc letelte után a legutóbbi befejezett iterációban talált legjobb lépés kerül visszaadásra.

A keresést a rekurzívan meghívott `minmax` metódus végzi. A rekurziót megállító alapeset pedig a `searchCaptures` szintén rekurzív metódust hívja meg, ami a leütéseket játsza le mindaddig amíg egy csendes pozíció nem kerül felfedezésre (egy olyan tábla ahol a következő lépésben nem lehetséges bábut leütni). Matt esetén 10,000 pont mínusz a mélység kerül értékelésre és a keresés azon az ágon befejeződik.

Ha a keresés egy csendes pozícióba ér a megfelelő mélységet elérve akkor az ott lévő tábla pontozásra kerül a `ScoreEvaluator` osztály `evaluate` metódusával. A pontszám az adott pozícióban lépéssel következő játékos nézőpontjából számított.

Az pozíció pontszáma a rekurzív meghívások által kialakított fán felfelé haladva kerül értékelésre. Az értékelés során a *felhasználó* a saját pontjait növelni, az ellenfele pontszámát pedig csökkenteni szeretné, a *computer* pedig pontosan az ellenkezőjét.

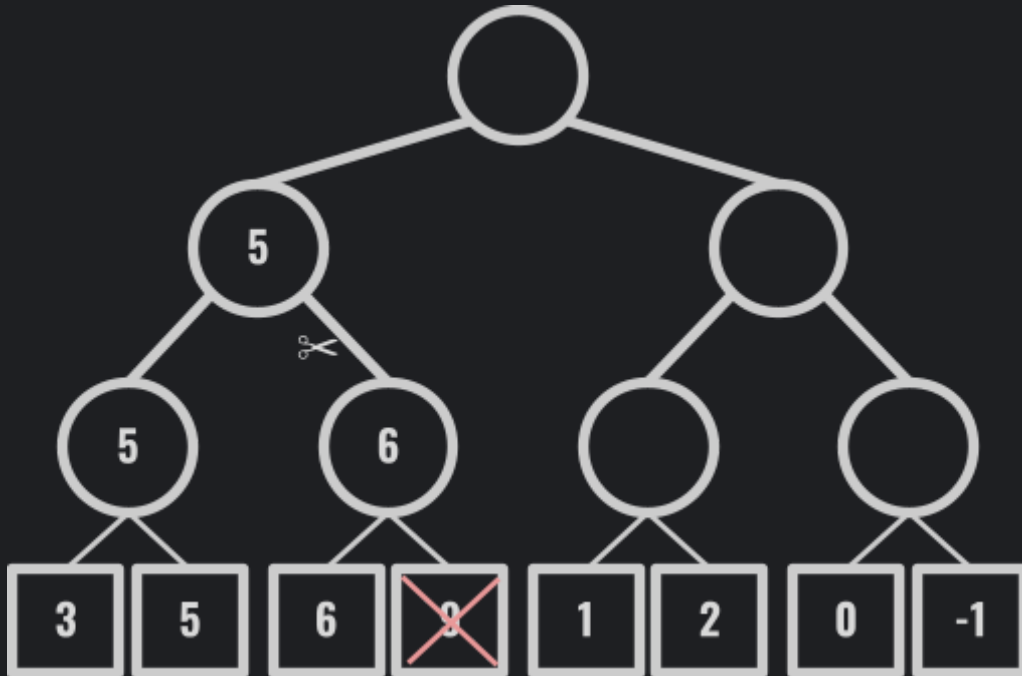
A megvalósításban a *computer* a maximalizáló és a *felhasználó* a minimalizáló fél. Ez a váltakozás a *negamax* algoritmus segítségével történik. A `minmax` metódus önmaga meghívásának visszatérési értékét negatív előjellel dolgozza fel.

Optimalizációs módszerek:

Az algoritmus a számítási sebesség, ezáltal a jobb lépések kiválasztása érdekében több optimalizációs megoldással van ellátva.

1. Alpha-beta metszés

Az alpha-beta metszés egy egyszerű észrevételen alapul, aminek segítségével nagyságrendekkel csökkenthető a keresésben résztvevő pozíciók száma, az eredmény befolyásolása nélkül.



Az ábra segítségével belátható, hogy ez miért lehetséges.

Az evaluáció alulról felfelé történik. Az első döntésnél a maximalizáló játékos a 3 és az 5 közül kiválasztja az 5-t. A következő részében az első levél értéke 6, a nagyobb részében pedig a minimalizáló játékos dönt, aki ezt a lépést tehát semmiképpen nem fogja meglépni, hiszen legalább 6-os pontszámot hozhat ki belőle az ellenfele, ami mindenképp rosszabb, mint a másik lépése ahol legjobb esetben is csak 5-öst. Levághatjuk tehát ezt az egész részét, nincs szükség tovább keresni benne.

A kódban az **alpha** és **beta** változók a maximalizáló illetve a minimalizáló játékos legjobb opciónak értékeit tárolják. Az alapértelmezett verzióban a vágás akkor történik, ha **beta** értéke kisebb, vagy egyenlő mint az **alpha**.

A negamax algoritmus megvalósításához **alpha** és **beta** értékei minden meghíváskor felcserélődnek és előjelet váltanak, a vágás pedig **alpha** \geq **beta** esetén történik.

2. Előrendezés

Az alpha-beta metszés hatékonysága azon alapul, hogy mennyi vágás történik a keresés során. A vágások pedig akkor történnek ha a lépések sorrendjében egy jobb lépés előbbre van egy rosszabbnál.

Habár a lépések pontszáma természetesen nem ismert, ezért van szükség a keresésre, az algoritmus felgyorsítható egy becslés alapú rendezéssel.

Minden lépés kap egy pontszámot, ami alapján csökkenő sorrendbe állíthatóak.

A becsült pontszám a következők alapján készül el:

- Leütés esetén a két bábu pontértékének különbsége
- Promóció esetén a királynő pontszáma (9)
- Pontlevonás paraszt által támadott mezőre lépés esetén

Ezzel a becsült pontszámítással és rendezéssel lényeges optimalizáció érhető el.

3. Transzpozíciós tábla

A játék során előfordulhat, hogy ugyanazt a táblapozíciót többször kell kiértékelni. Az iteratív mélyítés során pedig egyenesen garantált. A pozíciókhoz tartozó pontszámoknak, vagy metszés esetén ezek alsó vagy felső korlátainak tárolásával jelentős teljesítménynövekedés érhető el.

Az iteratív mélyítés pedig gyorsabb lesz, mintha egyből egy adott mélységre keresnénk, hiszen a már tárolt pozíciókhoz hozzárendelhetjük a legoptimálisabb lépéseket is. Amennyiben a tárolt pontszám pontos a többi lépés kihagyható, ellenkező esetben pedig az optimális lépés előresorolható az előrendezésben.

A táblákat először hasítani szükséges a gyors kereséshez. Ehhez a program a **Zobrist** hasító algoritmust használja. Ez egy sakktáblából egy 64 bites számot állít elő, amit a program **long** típusban tárol. Figyelembe véve a táblán lévő bábukat, a következő játékos színét, a *sáncolási* jogokat és ha lehetséges az *en passant* oszlopát.

A tábla kódja egy **HashMap**ben kerül az azt leíró **TableEntry** objektummal kapcsolatba.

A `TableEntry` tartalmaz egy pontszámot, az ertároló keresés mélységét (pontosan azt, hogy mennyire messze van a keresés vége, a rekurzió alapesete), az optimális lépést és azt, hogy a pontszám pontos vagy csak egy alsó/felső korlátja az elérhető lépésekből szerezhető pontszámoknak.

A `TableEntry` által tárolt pontszám felhasználásához a `getCorrectedScore` metódus alkalmazott. A metódus paraméterei megegyeznek a `minmax` metódus paramétereivel. A keresésnek megfelelő pontszámot ad vissza. Amennyiben az eltárolt pontszám alacsonyabb mélységgel rendelkezik, mint az őt igénlő keresés a visszaadott érték a `LOOKUPFAILED` konstans.

Ha a keresés alatt vágás történik, akkor az ottani `beta` egy alsó korlátja lesz az elérhető pontszámoknak, a `LOWER_BOUND` konstans értékét kapja meg a `nodeType` mező. Ez a típus egyedül arra használható, hogy a visszaolvasáskor az eltárolt pontszámnál kisebb `beta` érték esetén vágás ejthető, ilyenkor a `getCorrectedScore` által visszaadott érték a keresés `beta` értéke.

Ha a keresés alatt mindegyik lépés rosszabb, mint az egyik szomszédos részfa lépései akkor az `alpha` érték kerül tárolásra `UPPER_BOUND` típusú bejegyzéssel. Visszatéréskor az ennél nagyobb `alpha` esetében ejthető meg vágás az `alpha` visszaadásával.

Ha az előzőek nem következnek be a keresésben akkor az adott táblából elérhető legjobb pontszám kerül a `score` mezőbe, visszaadáskor a vágások lehetőségét fenntartva kerül elszámolásra még a `getCorrectedScore` metódusban.

A keresőalgoritmus a közvetlen mattot 10,000 ponttal értékeli. A matt minnél mélyebben van annál kevesebb pontot ér, a gyökértől való távolságot levonva a 10,000-ből. Így az gyorsabb mattok prioritást élvezhetnek. A transzpozíciós táblában való tároláshoz ezt viszont el kell tüntetni, majd a lekéréskor a lekérést végző keresés mélységével visszaadni. Erre szolgál a `correctMateScoreForStorage` és a `correctRetrievedMateScore` metódusok, amelyek az előjelet megőrizve alakítják a matt pontszámát a megfelelő értékekre.

Az algoritmus metódusainak részletes leírása a [Javadoc](#)ban megtekinthető.

Tesztelési Terv

Board class tesztek:

- getElement tesztelése, hogy jól konvertálja-e át a sakk koordinátát, az alkalmazásunk által használt koordináta rendszerre
- At tesztelése, hogy a Vector2ről jól alakít-e át az alkalmazásunk által használt koordináta rendszerre
- toString tesztelése, hogy a Board megfelelően tárolja-e és írja ki a tartalmát.

Vector2 class tesztek:

- add tesztelése, két Vector2 összeadásának tesztelése
- flip tesztelése, egy Vector2 megfordítása az y irányban
- toString tesztelése, a megfelelően tárolt adatok kiírása
- scaleBy tesztelése, egy Vector2 megnövelése konstanssal
- normalize tesztelése, egy Vector2 normalizálása
- sameDirection tesztelése, két Vector2 ugyan abba az irányba van-e
- outOfBounds tesztelése, hogy a Vector2 a sakk tábla korlátain belülre mutat-e

JsonToBoard class tesztek:

- getPieceType tesztelése, minden piece típusra a megfelelő szöveget adja-e vissza
- getMoveSet tesztelése, minden piece moveset típusra a megfelelő szöveget adja-e vissza

BordVisualizer class tesztek:

- getLookChar tesztelése, minden piece kinézetre a megfelelő karaktert adja vissza
- getColorChar tesztelése, minden piece színre a megfelelő karaktert adja vissza

Szituáció tesztek:

- Pinning tesztek, meghatározzák, hogy a sakkban a pin funkció jól működik-e
- kingMovesInCheck tesztek, meghatározzák, hogy a királynak a lépései megfelelőek akkor is ha sakkban van
- kingMoveInChess tesztek, meghatározzák, hogy a királynak a lépései megfelelőek akkor is ha sakkban van

Engine class tesztek:

- getMoves tesztelése, megvizsgálja hogy a geMoves metódus az elvárt lépéseket adja vissza
- move tesztelése, megvizsgálja hogy a lépés logika megfelelően működik-e
- Reset tesztelése, meghatározza hogy a reset / újraindítás megfelelően működik-e

Algorithm class tesztek:

- testSearch teszt, megvizsgálja hogy egy előre meghatározott idő alatt sikerül-e lefutnia a keresésnek