

Sakk dokumentáció
Piroska Lázár András
FYBLEU

Felhasználói útmutató

A használat egyszerű. Az alkalmazás elindításakor kirajzolódik a sakktábla egy új játékkal. A különböző bábuk kiválaszthatóak egy kattintással. Ha még egyszer rákattintunk a bábura a gép lép vele (ha lehetséges). Ha nem lehet lépni vele kiíródik, hogy az adott bábuval nem lehet lépni. Mindig csak az aktuálisan következő fél bábuját lehet kiválasztani. A játék végén kiírásra kerül a végeredmény (ki nyert, vagy döntetlen).

Lehetőség van az aktuális játékot elmenteni a save gombra nyomással. Ekkor felugrik a könyvtár, ahol el lehet navigálni a mentés helyére. Hasonlóan az open gombbal betölthetünk egy már elmentett játékot.

Osztályok

Piece(bábu)

Absztrakt osztály, melyből a többi bábu származik.

Van színe, amit egy enumeráció ír le (CColor), van helye a táblán (Place osztályból), 2 logikai változó (mozdult-e – moved, leüthető-e - mightkill).

A konstruktor a bábu színét, helyét veszi át.

Segédfüggvények:

private boolean oneMove(int c1, int c2, List<Move> moves, Board board, boolean enableCapture, boolean enableEmptyField) – ha a bábu az első két paraméter által megadott helyre mozogna, oda tud-e menni, ha saját bábu van ott nem mehet oda, ha ellenkező színű, akkor odamehet (beteszi a moves listába a célhelyet), de false-t ad vissza, mert nem lehet továbbmenni, ha nincs ott senki akkor true értékkel tér vissza, tehát tovább is lehet lépni.

private boolean onePlace(int c1, int c2, List<Place> places, Board board) – A célmezőt beleteszi a places listába, igazzal tér vissza, ha nincs rajta bábu.

protected void longStraightMove(int c1, int c2, List<Move> moves, Board board) – Addig tud menni a c1, c2, illetve azok többszörösei által meghatározott irányba amíg a oneMove segédfüggvény hamissal nem tér vissza.

protected boolean shortStraightMove(int c1, int c2, List<Move> moves, Board board, boolean enableCapture, boolean enableEmptyField) – meghívja a oneMove – ot egyszer, ezt használja majd a király, ló, gyalog; akik csak egyet léphetnek.

protected boolean shortStraightMove(int c1, int c2, List<Move> moves, Board board) – ugyanaz mint az előző, de ezt használja a ló és a király, mert itt nem különböztetjük meg az ütést és a lépést (enableCapture, enableEmptyfield itt true).

protected void longStraightPlaceCheck(int c1, int c2, List<Place> places, Board board) – Azokat a mezőket gyűjti össze, amelyek a c1, c2 irány szerint kijelölt úton elérhető a bábu számára.

protected boolean shortStraightPlaceCheck(int c1, int c2, List<Place> places, Board board) – csak a onePlace – t hívja meg egyszer.

A változók getterekkel, setterekkel érhetők el és módosíthatók.

Absztrakt függvények:

public abstract List<Move> getMoves(Board board, List<Move> moves); - a báb számára lehetséges lépéseket gyűjti össze

public abstract List<Place> getControlledPlaces(Board board, List<Place> places); - a bábu által kontrollált helyeket adja vissza (léphet oda, üthet..)

`public abstract String getFigure();` - a bábu megjelenítéséhez szükséges unicode stringet adja vissza

`public abstract float getValue();` - a bábu értéke

Bishop (futó)

Egy Piece – t valósít meg, a konstruktor minden bábunál ugyanúgy működik, a Piece konstruktorját használják (szín, sor, oszlop).

`public List<Move> getMoves(Board board, List<Move> moves)` – a `longStraightMove` segítségével a diagonális irányú lépéseket gyűjti össze.

`public List<Place> getControlledPlaces(Board board, List<Place> places)` – a `longStraightPlaceCheck` segítségével a diagonálisan elérhető helyeket gyűjti össze. (tehát ebbe benne vannak azok a helyek is melyeken a saját színű bábuk álnak, ellentétben a `getMoves` metódussal)

`public float getValue()` – a futó értéke 3.

`public String getFigure()` – a futó megfelelő színű unicode karakterét adja vissza.

King (király)

Szintén egy Piece-t valósít meg mint az összes többi bábu.

`public List<Move> getMoves(Board board, List<Move> moves)` – a `shortStraightMove` segítségével minden irányban elérhető lépéseket gyűjti össze.

`public List<Place> getControlledPlaces(Board board, List<Place> places)` – a `shortStraightPlaceCheck` segítségével minden irányban elérhető helyeket gyűjti össze.

`public String getFigure()` – a király megfelelő színű unicode karakterét adja vissza.

`public float getValue()` – a király értéke 100 (hiszen ezen a bábun múlik a játék).

Knight (ló/husár)

`public List<Move> getMoves(Board board, List<Move> moves)` – A `shortStraightMove` segítségével a ló által megléphető (max) 8 db lépést gyűjti össze. Ezeknél mindig a sor/oszlop változik 1-et vagy 2-őt.

`public List<Place> getControlledPlaces(Board board, List<Place> places)` – a `shortStraightPlaceCheck` segítségével a ló által elérhető (max) 8 db helyet gyűjti össze.

`public String getFigure()` – a ló megfelelő színű unicode karakterét adja vissza.

`public float getValue()` – a ló értéke 3.

Pawn (gyalog)

`public List<Move> getMoves(Board board, List<Move> moves)` – A `shortStraightMove` segítségével a gyalog által megléphető lépéseket gyűjti össze. Ha még nem mozdult a gyalog akkor kettőt is tud előre lépni. Itt fontos, hogy a diagonális iránynál az `enableCapture` `false`, mert diagonálisan ütni, de lépni nem.

`public List<Place> getControlledPlaces(Board board, List<Place> places)` - `shortStraightPlaceCheck` segítségével csak a diagonális irányban elérhető helyeket gyűjti össze.

`public String getFigure()` – a gyalog megfelelő színű unicode karakterét adja vissza.

`public float getValue()` – a gyalog értéke 1.

Queen (királynő/vezér)

Ugyanúgy működik mint a király, csak `shortStraightMove` és `shortStraightPlaceCheck` helyett `longStraightMove` és `longStraightPlaceCheck`-et használ, mert minden irányba hosszan is tud lépni. Továbbá a királynő értéke 10.

Rook (Bástya)

`public List<Move> getMoves(Board board, List<Move> moves)` – a bástya 4 irányba mozoghat, ezeket a lépéseket gyűjti össze `longStraightMove` segítségével.

`public List<Place> getControlledPlaces(Board board, List<Place> places)` – `longStraightPlaceCheck` segítségével mind a 4 irányban összegyűjti a bástya által elérhető helyeket.

`public String getFigure()` – a bástya színének megfelelő unicode karakter.

`public float getValue()` – a bástya értéke 5.

Fields (mező(k))

A sakktábla egy mezőjét reprezentálja. TreeMap<>-et használ, ahol a kulcs a hely, az érték pedig az ott lévő bábú.

public CColor getColor(int rank, int file) - visszaadja az adott mezőn álló bábú színét, illetve CColor.UNDEFINED-et, ha nincs bábú.

public String getFigure(int rank, int file) - visszaadja az adott mezőn álló bábú megjelenítéséhez használt unicode stringet, illetve üreset, ha nincs bábú.

public float getValue(int rank, int file) - visszaadja az adott mezőn álló bábú értékét, illetve 0-t, ha nincs bábú.

public boolean getMoved(int rank, int file) - visszaadja, hogy az adott mezőn álló bábú mozgott-e valaha, illetve true-t, ha nincs bábú a mezőn.

public boolean isEmpty(int rank, int file) - visszaadja, hogy üres-e a mező, vagy van rajta bábú.

Place (hely)

Helyet ír le rank – file (x,y koordináta), hozzá getterek és setterek.

boolean equals(Place place) – azt vizsgálja, hogy a paraméterül kapott hely koordinátái ugyanazok-e mint az övé.

public void setPos(int rank, int file) throws InvalidParameterException – Inicializál egy helyet, ha akármelyik koordináta a 0-7-en kívül esik InvalidParameterException-t dob., itt meg van engedve viszont, hogy a rank és file -1 legyen, mert a megjelenítésnél így jelöljük a kijelölt mezőt.

public void addPos(int rank, int file) throws InvalidParameterException – koordinátát módosít(hozzáad valamennyit), ennek eredményeképpen, ha 0-7 intervallumon kívül esik bármelyik koordináta kivételt dob.

public boolean onBoard() – a táblán van-e az adott mező (a kijelölt mező majd a megjelenítésnél -1,-1, akkor nincs kijelölt mező)

Move

2 Place-t ír le – honna indul (Move őse), hova érkezik (Place to). Ennek is van értéke (value). Ha ez egy bábú leütése, akkor a leütött bábú értéke.

public Move(int rank, int file, boolean castlingWay) – Sáncolás lépést hoz létre, annak a király-mozgását tárolja el. Ha a sor vagy oszlop nem egyezik a király kezdeti pozíciójával InvalidParameterException-t dob. A harmadik paraméter a sáncolás irányát határozza meg.

Board

Fields - Egy TreeMap<Place, Piece> leszármazott. A nextplayer pedig az épp következő lépést lépő fél (szín).

matchResult – aktuális állás (enummal)

A konstruktor egy 8*8-as üres field tömböt hoz létre. Fieldek vannak benne, de bábuk nem.

public Field getField(Place place) – Egy mezőt ad vissza a helye alapján.

public Piece getPiece(Place place) – a mezőn lévő bábu

public void setPiece(Piece piece) – adott mezőre lép egy bábu

public void setEmpty(int rank, int file) – üres lesz az adott mező

public CColor getActualPlayer() – a mostani játékost (szín) adja vissza

public boolean isRightPlace(int rank, int file) – A mezőn olyan színű bábu van-e mint az aktuális játékos, ha pl fehér játékos fekete vagy üres helyet jelöl ki akkor azzal ne foglalkozzon.

public MatchResult getMatchResult() – a játék eredménye

public String getMatchResultString() – az előző String formában.

public String getFigure(int rank, int file) – adott mezőhöz tartozó unicode string

public void reset() – fölteszi a bábukat (induló állás)

private List<Move> getMoves(Piece piece) – Egy adott Piece lehetséges lépéseit gyűjti össze, ha az nem null, ha null akkor az aktuális játékos lehetséges lépéseit, azaz minden bábuval megnézi hova lehet lépni, ezeket visszaadja.

private CColor opponentColor() – a nem aktuális játékos színe

private List<Place> getControlledPlaces() – a nem aktuális játékos által által kontrollált mezők összegyűjtése.

private Place getKingPlace(CColor color) – adott színű király helye

private boolean checkChess() – megvizsgálja, hogy sakkbán van-e az aktuális játékos királya

private void rollbackMove(Move move, Piece piece, Piece savedPiece) – visszaállít egy lépést, azaz a piece kerül vissza a move forrásába, és a savedPiece a move céljába.

private boolean performMove(Move move, boolean needCheck, boolean testOnly) – egy lépést végrehajt, ellenőrzi, hogy sakkbá lép-e, ha tesztként hajtja(testOnly == true) végre végrehajtja és utána visszalépi, ez azért kell, hogy vissza tudja adni, hogy végrehajtható-e a lépés. Visszatér azzal, hogy sikeresen végrehajtható-e a lépés.

`private void setResult()` – Ha valamelyik félnek nincs több lépése hívjuk, megnézi, hogy a mostani állásban sakkban van-e valaki, vagy nem ezek szerint lesz veszített/döntetlen a meccs.

`private void swapPlayer()` – megváltoztatja az aktuális játékost, megnézi, hogy folytatódik-e a meccs.

`private boolean checkEndOfGame()` – ezt a függvényt használja az előző (is), lekéri az adott szín összes lépését, ha nincs akkor vége a játéknak.

`public boolean automaticMove(Place from)` – az adott bábu lehetséges lépéseit kéri le, ebből kiválasztja a legmagasabb értékűt: van a move-nak egy értéke (ha leüt egy bábút a leütött bábu értéke), ha ütésből kilép, akkor a saját bábu értéke hozzáadódik az értékhez, ha ütésbe lép bele levonódik, ha a lépéssel sakkot ad még fél pontot hozzáad. Tized ponttal díjazza, ha valamit leüt, így sokszor a védett bábút is inkább ütni fogja. A sáncolás is díjazott, így a király – ha lehet – inkább sáncolni fog (persze az ütés így is preferált). Ezekből a legmagasabb értékűt kiválasztja (ha több is van véletlenszerűen). Lényegében végignézi a move-listát és megpróbálja a legjobbat kiválasztani.

`public void getCastlingMoves(Piece piece, List<Place> places, List<Move> moves)` – Ha a király már mozgott, vagy a kiválasztott bábu nem is király nem történik semmi. Megnézi, hogy rövid oldalon van-e ellenfél által kontrollált mező, vagy van-e bábu (for ciklussal), ha a ciklus sikeresen végigment, hozzáadja a rosálást a lehetséges lépésekhez. Ugyanezt utána megteszi hosszú oldallal is.

FieldView

JButtonból származik, van 4 szín a fehér, kijelölt fehér, sötét, kijelölt sötét.

Megkapja a BoardView-t, a mező pozícióját a konstruktorban. Ha osztható ketővel a sor és oszlop összege akkor fehér színű a mező, amúgy fekete.

`public void setHighlighted(boolean highlighted)` – beállítja, hogy kijelölt -e az adott mező

`public void setFigure(String figureText, int size)` – beállítja a gomb szövegét(a figurát jelölő unicode karakter)

BoardView

JFrame-ből származik, implementál egy ComponentListener-t, ennek csak a componentResize függvényét használja.

JLayeredPane layeredPane erre tesszük rá a táblát, gombokat, stb., a ComponentListenert kezeli. A JPanel chessBoard maga a tábla megjelenítésére szolgál, erre tesszük a FieldView-kat. Van save, open gomb. Label felirat kiírja, hogy ki lép ki nyert stb. Van egy Place, a kijelölt helyhez kell. Van egy Board, ami a tényleges sakktábla.

`public BoardView(Board board)` a konstruktor létrehozza a szükséges gombokat, stb, hozzárendeli a megfelelő Listenereket a gombokhoz (open, save-nek van ActionListener, JLayeredPane-nek van ComponentListenerje (a resize miatt)).

class SaveFile implements ActionListener – a save gomb lenyomásakor elmenti az aktuális állást. A showSaveDialog megadja a tállózás helyét.

class OpenFile implements ActionListener – fájl megnyitásához, a fájlt a Boardba olvassa be, ezután újra felrajzolja a táblát.

static private int borderedValue(int value, int min, int max) – Segédfüggvény, az első paraméter értékét a min-max intervallumra korlátozza

private void setBoardSize(int width, int height, boolean forced) – A Sakktábla méretét állítja be, felette 100 pixelnyi hellyel a gomboknak. A tábla szélessége és magassága mindig 100-zal osztható, a max és min méret megadható az elején, alapból 300 és 700

public void ReDrawBoard(String additionalInfo) – Mind a 64 fieldView feliratát beállítja (bábuk/üres), méretezi a feliratot, kiírja egy label-re, hogy mi az aktuális helyzet (fehér jön, stb.).

public void setField(int rank, int file) – kijelöltté teszi az adott mezőt, ha a mezőn olyan bábu van mint amilyen színű játékos következik lépni.

public void clickField(int rank, int file) – a fieldView hívja ezt a függvényt amikor rákattintottak, megnézi, hogy van-e kijelölt mező, ha nincs hívja a setField-et, amúgy a performMove-ot.

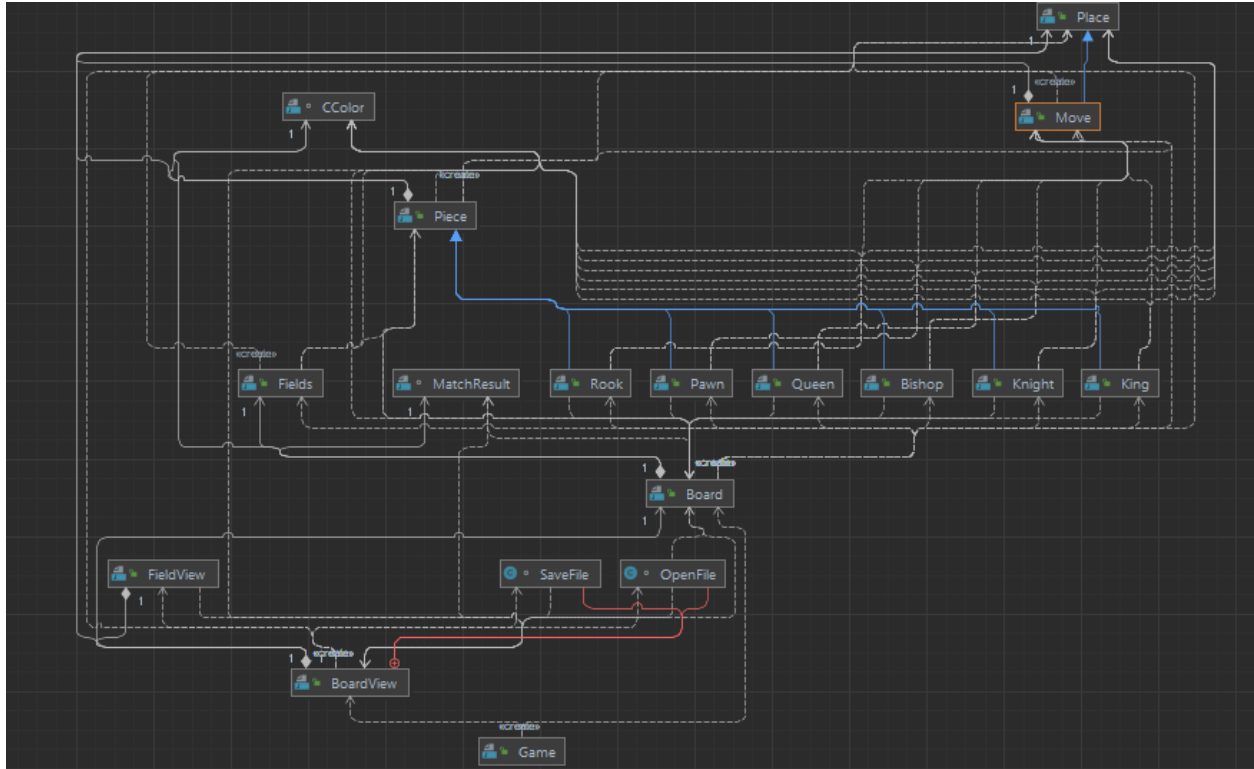
public void performMove(int rank, int file) - Ha másra kattintottak mint a kijelölt akkor megszünteti a kijelölést, majd a setField-et hívja. Ha ugyanarra kattintott mint a kijelölt, akkor megszünteti a kijelölést, végrehajtja a programmal a lépést, majd újrarajzolja a táblát. Ha a lépést nem lehet végrehajtani, vagy véget ért a meccs akkor ezt kiírja.

public void componentResized(ComponentEvent e) – ComponentListener metódusa, ablak újraméretezésekor hívódik, meghívja a setSize függvényt, ablakméreteknek megfelelően állítja be a táblaméretet.

UML diagram

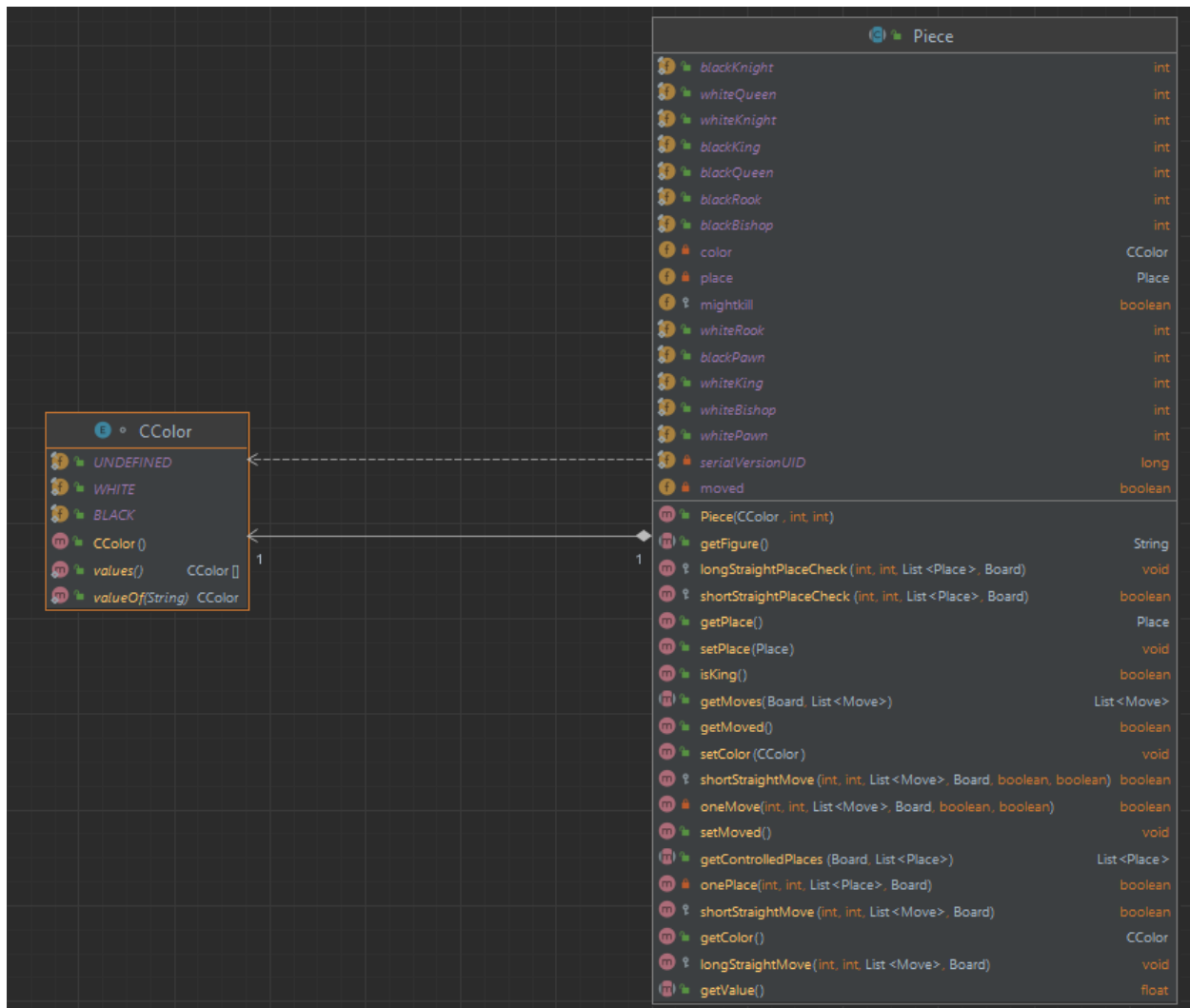
Az osztályok metódusai, változói külön-külön, hogy átláthatóbb legyen.

Osztályok közötti kapcsolatok:

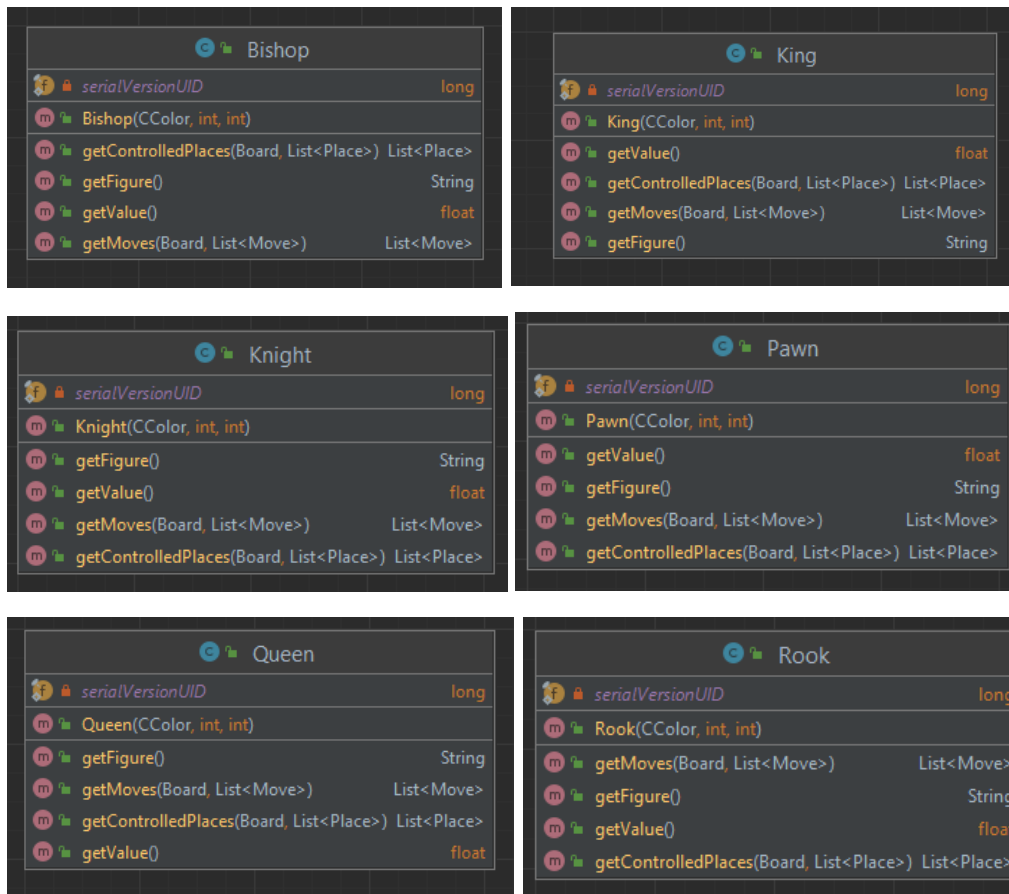


Maguk az osztályok:

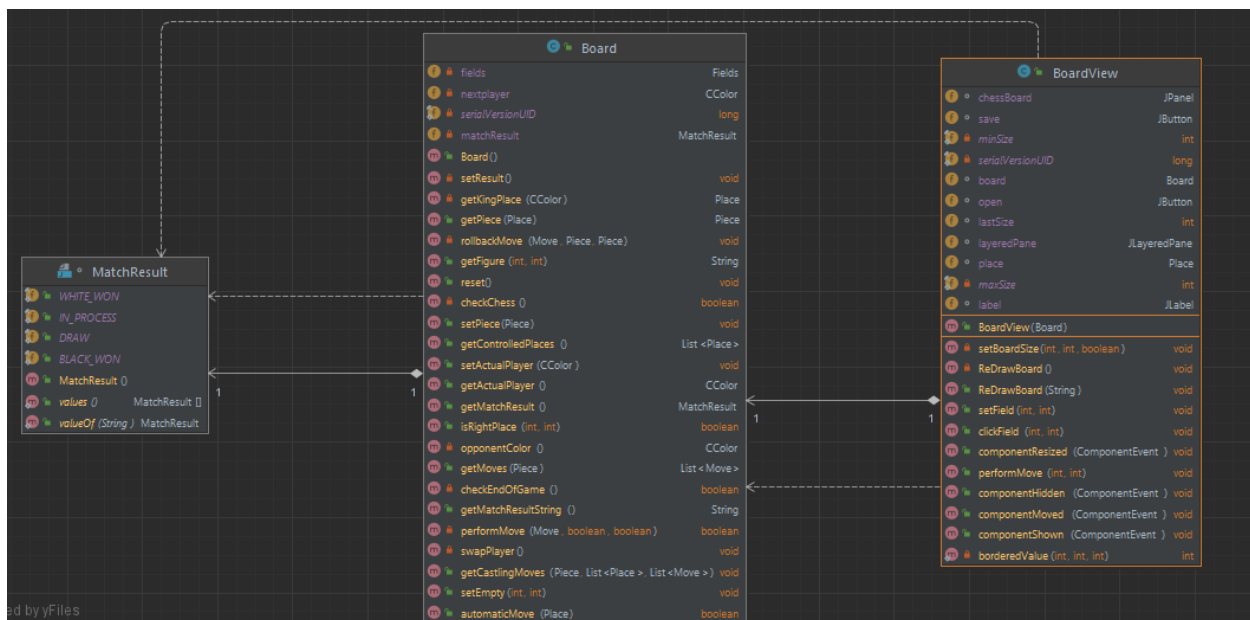
Piece:











Piece leszármazottjai:



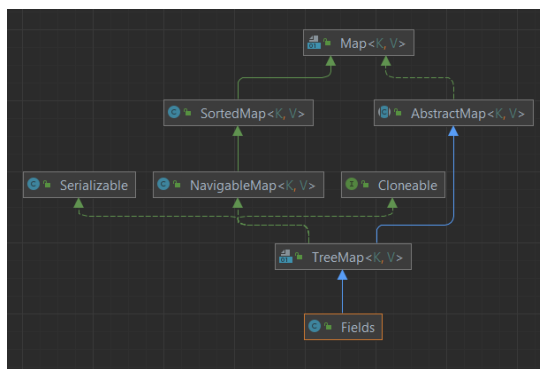
Board, MatchResult és BoardView:




















Fields:

Fields		
 	<code>serialVersionUID</code>	<code>long</code>
	<code>Fields()</code>	
	<code>isEmpty(int, int)</code>	<code>boolean</code>
	<code>getValue(int, int)</code>	<code>float</code>
	<code>getFigure(int, int)</code>	<code>String</code>
	<code>getColor(int, int)</code>	<code>CColor</code>
	<code>getMoved(int, int)</code>	<code>boolean</code>



+Fields használ TreeMap-et



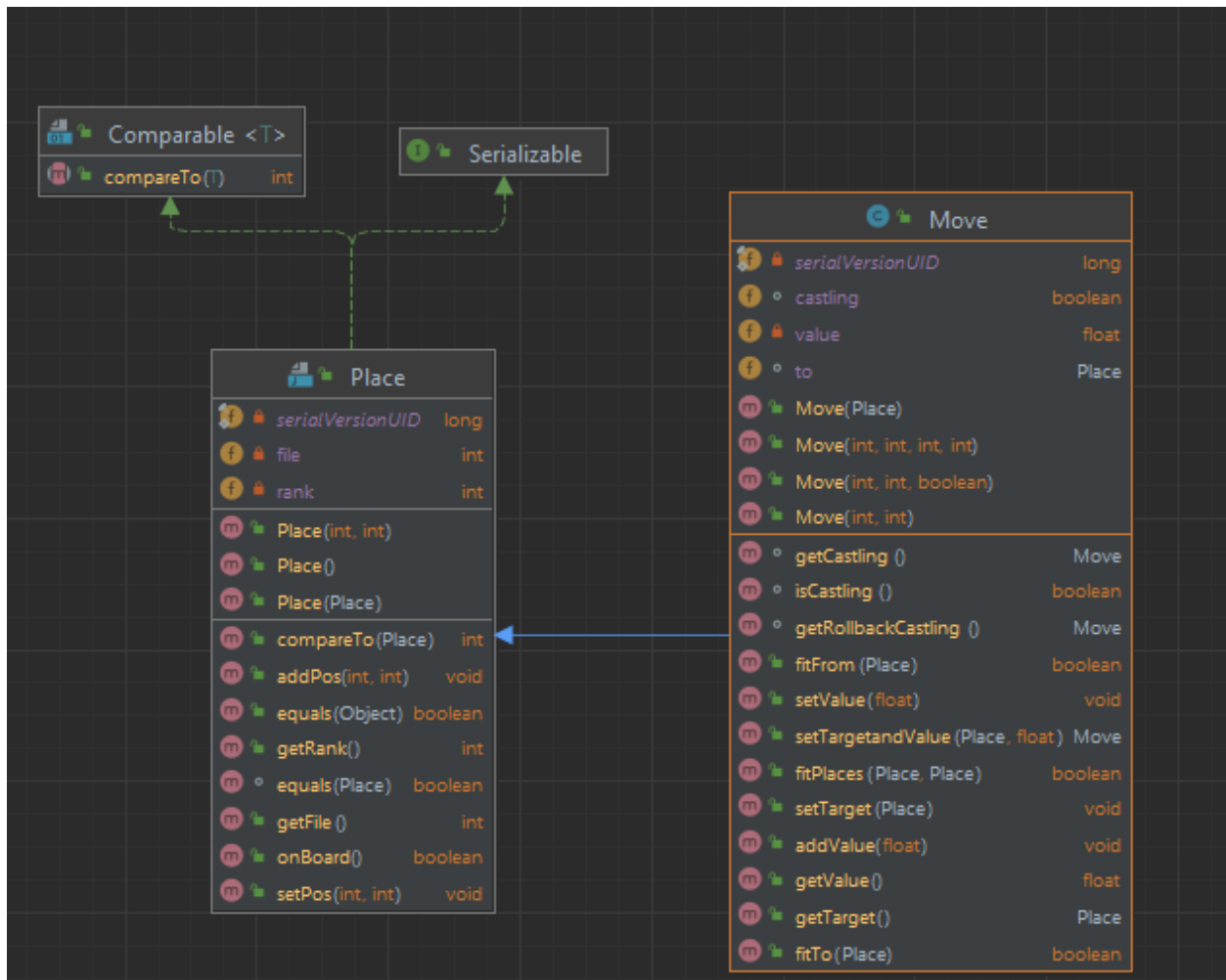
FieldView:

FieldView		
 	<code>serialVersionUID</code>	<code>long</code>
 	<code>LIGHT_GRAY_HL</code>	<code>Color</code>
 	<code>boardView</code>	<code>BoardView</code>
 	<code>WHITE</code>	<code>Color</code>
 	<code>isWhite</code>	<code>boolean</code>
 	<code>LIGHT_GRAY</code>	<code>Color</code>
 	<code>WHITE_HL</code>	<code>Color</code>
	<code>FieldView(int, int, BoardView)</code>	
	<code>setFigure(String, int)</code>	<code>void</code>
	<code>setHighlighted(boolean)</code>	<code>void</code>

Game:

Game		
	<code>Game()</code>	
	<code>main(String[])</code>	<code>void</code>

Move és Place:



Teszték

BoardTest:

Felállít egy pozíciót, ahol egyes lépések adottak az automaticMove algoritmus által, ezeket vizsgálja. A vizsgált metódus tehát az automaticMove, de ezen keresztül a többi függvényt is megvizsgálja.

MoveTest:

A Move konstruktorokat, fitPlaces, isCastling, getCastling, getRollbackCastling metódusok tesztelése

PlaceExceptionTest:

A Place osztály konstruktorát, addPos, setPos függvényeit teszteli hibás paraméterekre, kivételeket vár el.

PlaceTest:

A Place osztály konstruktorát, addPos, setPos, equals (override), compareTo (override) függvényeit teszteli. Megfelelő paramétereket vár el, tehát a pozíció a táblán van, vagy a speciális (-1, -1) koordinátán.