EÖTVÖS LORÁND TUDOMÁNYEGYETEM

Informatikai Kar

Programozás Nyelvek és Fordítóprogramok Tanszék

Sakkjáték szoftveres megvalósítása Java nyelven

Témavezető

Pataki Norbert

docens

Készítette

Hollósi Tamás

Programtervező Informatikus Bsc.

Budapest, 2023

Tartalomjegyzék

1. Bevezetés

A szakdolgozatom célja egy olyan sakkjáték implementálása, ahol 2 mód elérhető. Az egyik módban két játékos játszik egymás ellen, ilyenkor a szoftver elősegíti a szabályos lépések elvégzését. A másik módban egy felhasználó játszik egy számítógép vezérelte játékos ellen. A számítógép által működtetett játékos egy heurisztikán keresztül dönt arról, hogy milyen lépést hajtson végre.

1.1 A sakk

A sakk táblajáték két személy részére, és egyben sportág is. A „sakk" szó – amely nemcsak a játékot jelenti, hanem azt a helyzetet is, amikor az ellenfél királya „ütésben van" – a perzsa „*shāh*" (شَاه) szóból ered, amely uralkodót jelent. A sakk története a legendák világába nyúlik vissza. Az ismert mese szerint egy brahmin, név szerint bizonyos Széta találta fel a sakkot.

A sakknak különféle változatai ismertek, ezek közül szakdolgozatomban az Európában legnépszerűbb modern sakkot használom.

A sakkjátékot két játékos játssza egymás ellen a négyzet alakú, nyolc sorra és nyolc oszlopra felosztott sakktáblán, 16–16, azaz összesen 32 bábuval. A két játékos bábui határozottan eltérő színűek: világos, sötét.

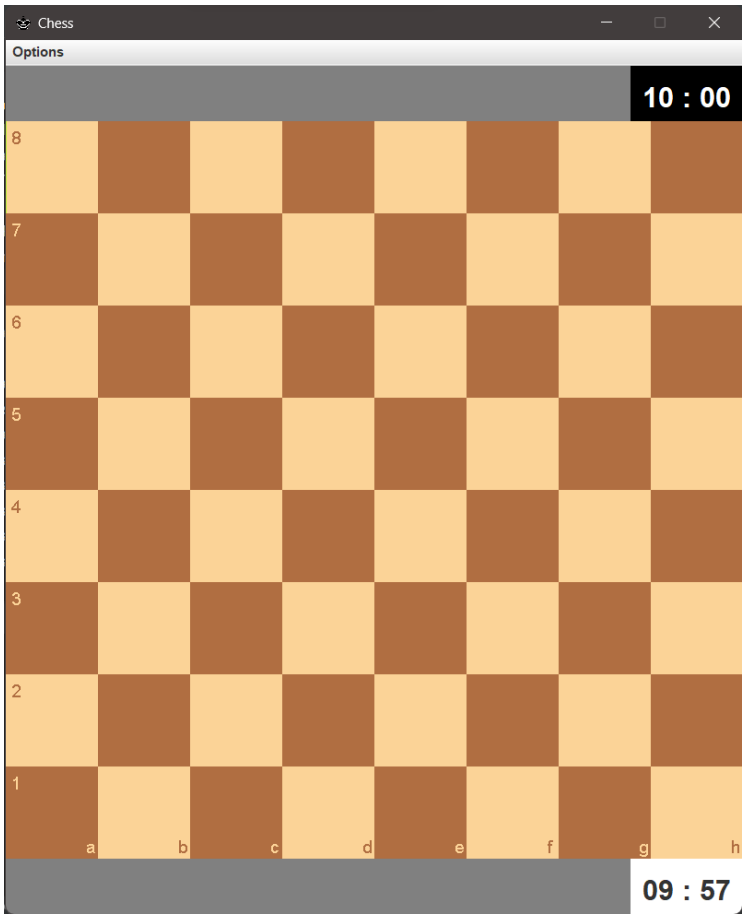
A játékosok felváltva lépnek, és mindkettőjük célja, hogy a másik fél király nevű figuráját a játékszabályok szerint bemattolják.

1.2 Szabályok

1.2.1 A tábla

A tábla mindig akkor áll helyesen, ha a bal alsó sarokban sötét, a jobb alsóban pedig világos színű mező van. Vízszintesen az 1-től 8-ig, arab számokkal jelölt sorok, függőlegesen az a–h betűkkel azonosított oszlopok vannak.

Szabályos sakktáblára példa az 1. ábrán látható.

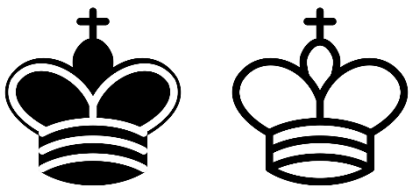


1. ábra: Szabályos sakktábla

1.2.2 A figurák

A játék kezdetén a világosnak és a sötétnek ugyanannyi figurája van: 1–1 király, 1–1 vezér (alternatív neve: „királynő”), 2–2 bástya (alternatív neve: „torony”), 2–2 huszár (alternatív neve: „ló”), 2–2 futó és 8–8 gyalog (alternatív neve: „paraszt”).

Király



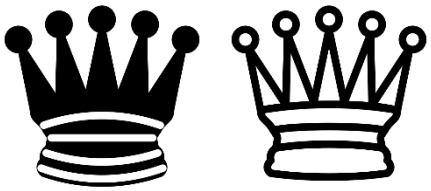
2. ábra: Király bábuk képei

A király a játék legfontosabb bábja, hiszen az egész játék a bemattolására irányul. Bármely irányban (vízszintesen, függőlegesen, átlósan) léphet, de csak egy mezőt, vagyis csak a közvetlen szomszédos mezőre, kivéve, ha sáncol.

A király mozgását korlátozza, hogy sakkba nem léphet, azaz nem állhat olyan mezőre, amelyen ki lehetne ütni.

A bábuk képei a 2. ábrán láthatóak.

Vezér

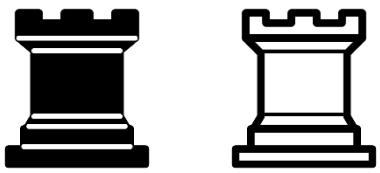


3. ábra: Vezér bábuk képei

A vezér mozgása olyan, mintha egyesítettünk volna egy futót egy bástyával, azaz egyenesen vagy átlósan bármely irányban, bármennyi mezőt léphet, mindaddig, amíg a tábla széléhez nem ér, vagy egy másik figura nem kerül az útjába

A bábuk képei a 3. ábrán láthatóak.

Bástya



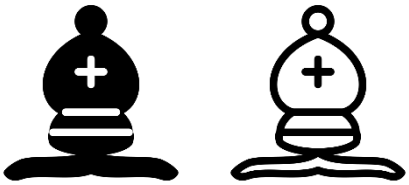
4. ábra: Bástya bábuk képei

Bármennyi mezőt léphet, de csak függőleges és vízszintes irányban, átlósan nem.

A megnyitásban a világossal játszó játékos bástyái az a1-es és a h1-es, a sötéttel játszó félé pedig az a8-as és a h8-as mezőkön helyezkednek el.

A bábuk képei a 4. ábrán láthatóak.

Futó

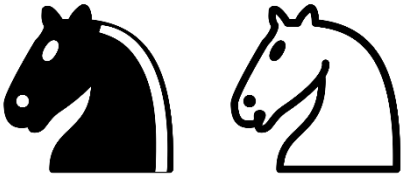


5. ábra: Futó bábuk képei

A futók átlós irányban léphetnek, bármennyi mezőt, amíg egy másik báb nem kerül útjába. A játékosok a játék elején két-két futóval rendelkeznek, melyek közül az egyik csak a sötét, a másik csak a világos mezőkön közlekedik. A futók a megnyitásban a világossal játszó félnek a c1-es és az f1-es, a sötét bábukkal játszó játékosnak pedig a c8-as és az f8-as mezején helyezkednek el.

A bábuk képei az 5. ábrán láthatóak.

Huszár

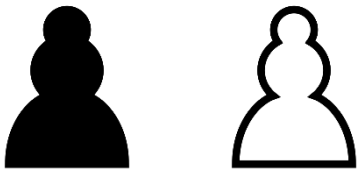


6. ábra: Huszár bábuk képei

A róla elnevezett „lóugrásban” lép: vízszintesen jobbra vagy balra két mezőt, majd függőlegesen fel vagy le egyet (vagy fordítva: függőlegesen fel vagy le kettőt és vízszintesen jobbra vagy balra egyet). A játék elején mindkét fél két-két huszárral rendelkezik, amelyek világos esetében a b1-es és g1-es, sötétnél a b8-as és g8-as mezőkön állnak.

A bábuk képei a 6. ábrán láthatóak.

Gyalog



7. ábra: Gyalog bábuk képei

A gyalog kizárólag előre léphet. A kiindulási helyéről mind a nyolc gyalog tetszés szerint egy vagy két mezőt léphet előre, de a továbbiakban lépésenként mindig csak egy mezőt haladhat előre. Ütni azonban csak jobbra vagy balra átlósan tud, szintén csak egy mezőnyi távolságra. Ha a gyalog áthaladt az egész táblán és eljutott az ellenfél alapsorába, átváltozik tisztté.

A bábuk képei a 7. ábrán láthatóak.

Különleges lépések

Sáncolás

A király és az egyik bástya együttes lépése. Ezt világos is, sötét is mindössze egyszer teheti meg a játék során. A király a kiválasztott bástya felé lép két mezőt, a kiválasztott bástya pedig a király által átlépett, tehát a királlyal szomszédos túloldali mezőre kerül.

A sáncolás csak akkor lehetséges, ha az alábbi feltételek mindegyike igaz:

- a király még nem lépett a játszma folyamán,
- az a bástya, amellyel sáncolni szeretnénk, még nem lépett a játszma folyamán,
- a király és a bástya között nem áll sem saját, sem ellenséges báb,
- a sáncolás előtt a király nem áll sakkban, a sáncolással nem kerül sakkba.

En passant

Ha a kiindulási mezőjéről kettőt lépő gyalogunkkal áthaladunk egy ellenséges gyalog ütésmezején, akkor az ellenfél gyalogunkat a következő lépésben – de csakis akkor – leütheti. Az ütés pontosan úgy történik, mintha az ütött gyalog csak egyet lépett volna.

Átalakulás

Ha a gyalog beér az ellenfele alapsorára, akkor át kell változtatni a gyaloggal azonos színű tisztté: vezérré, bástyává, huszárrá vagy futóvá, a játékos választása szerint, függetlenül attól, hogy az adott tisztből hány van még játékban.

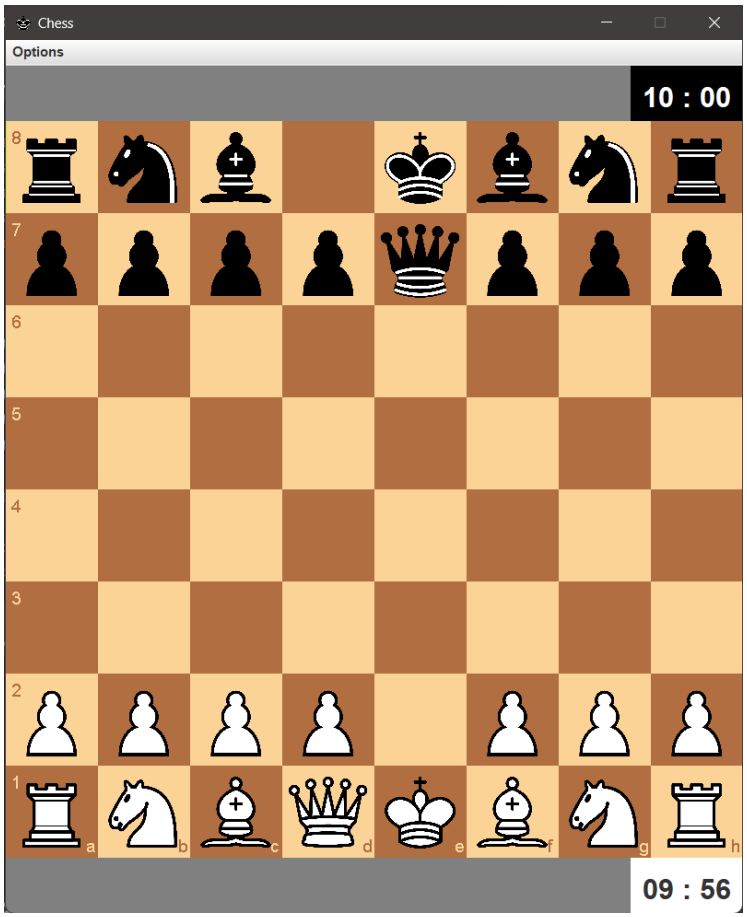
1.2.3 Sakk, Matt, Patt

Sakk

Akkor mondjuk, hogy a király sakkban van, ha az egy vagy több ellenséges báb által támadva van, még akkor is, ha ezek a bábok nem tudnak lépni. Olyan lépést tilos tenni, mely a királyt sakkba helyezi, vagy sakkban hagyja. A sakk elhárításának három módja:

- A sakkadó báb leütése
- A sakkadó báb és a király közé egy másik bábu helyezése, blokkolás
- A királlyal egy meg nem támadott mezőre lépünk.

Sakk helyzetre példa a 8. ábrán látható.



8. ábra: Sakk helyzet példa

Matt

A sakkjáték célja az ellenfél mattolása. Ha a játékos nem tudja a király támadását elhárítani, mattot kapott, és a játszmának azonnal vége a mattot adó játékos győzelmével.

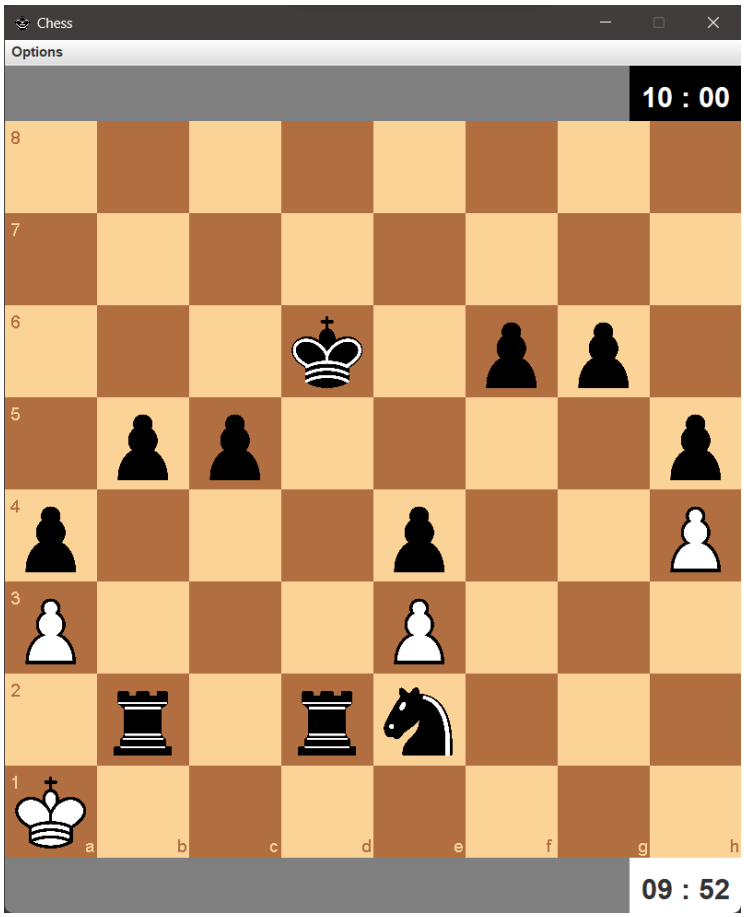
Matt helyzetre példa a 9. ábrán látható.

9. ábra: Matt helyzet példa

Patt

A játszma döntetlen, ha a soron következő játékos nem tud szabályos lépést tenni, de a királya nem áll sakkban. Ezt nevezzük pattnak. A játszmának ebben az esetben is azonnal vége döntetlen eredménnyel.

Patt helyzetre példa a 10. ábrán látható.



10. ábra: Patt helyzet példa

2 Felhasználói dokumentáció

Ez a fejezet tartalmaz minden olyan információt, ami a program futtatásához szükséges. A program olyan sakkjátékot valósít meg, ahol két lehetséges játékmód elérhető. Az egyik módban két felhasználó játszik körökre osztva. Ilyenkor a szoftver elősegíti a szabályos lépések elvégzését és a játékállás megjelenítését. A másik módban egy felhasználó játszik egy számítógép vezérelte játékos ellen.

2.1 Rendszerkövetelmények

A programot Windows 11 operációs rendszeren, IntelliJ IDEA 2022.3.1 (Community Edition) környezetben, Java nyelven fejlesztettem.

A program futtatásához szükséges:

JDK 18 vagy frissebb verzió

Windows operációs rendszer esetén, legalább Windows 7 vagy annál frissebb rendszer verzió

A szoftvert nem kell telepíteni, a java -jar Chess.jar paranccsal indítható.

2.2 A program felépítése

2.2.1 Játék menü

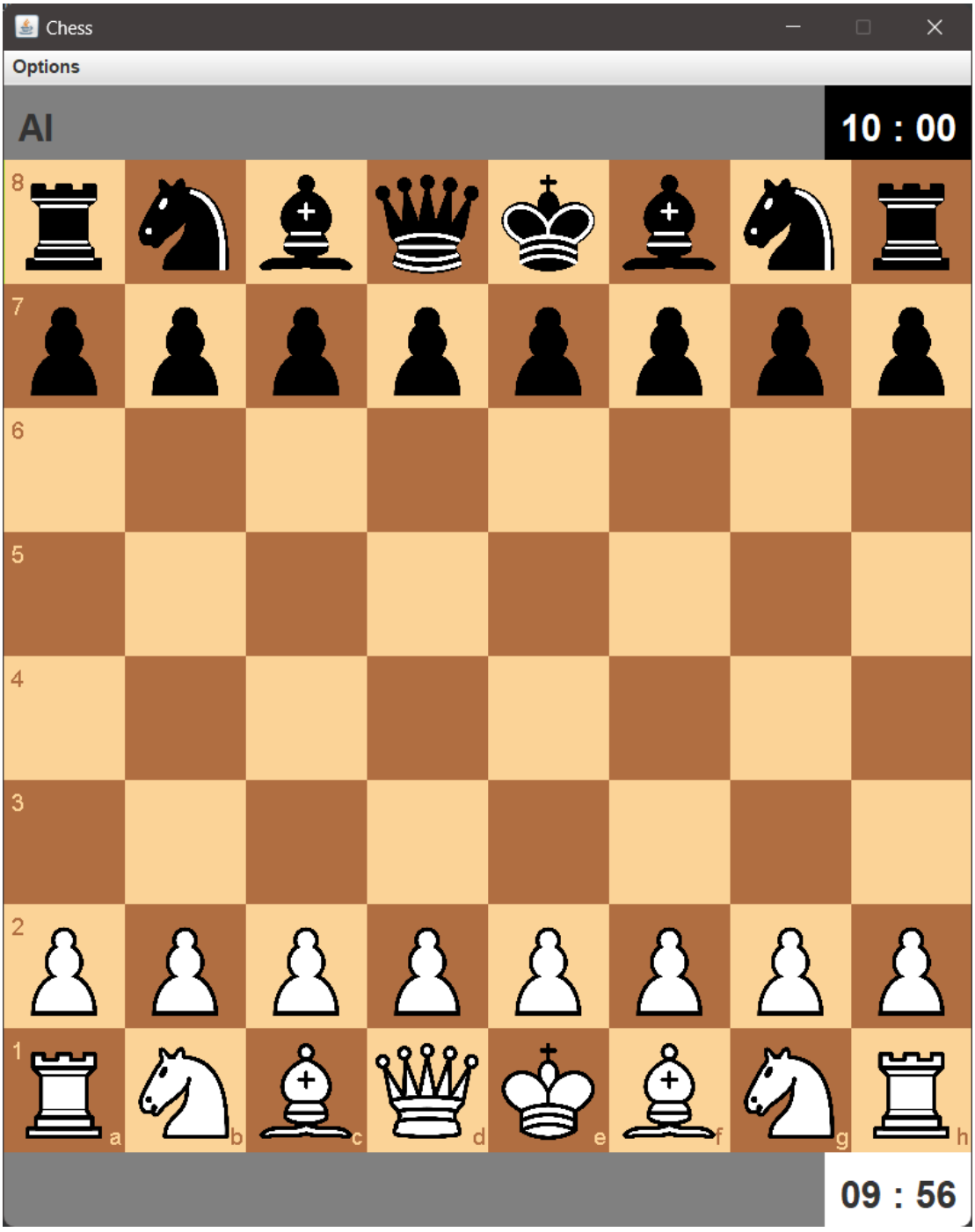
A program elindulása után megjelenik a 11. ábrán látható menü, ahol a két játékmód közül lehet választani, valamint a start gombbal elindítható a játék.

11. ábra: Menü a játékmód kiválasztásához

A start gomb megnyomása után megjelenik a 12. ábrán látható tábla és elkezdődik a játék.

A játék elindítása után azonnal elkezd visszaszámolni a világos játékos órája 10 percről.

A lépés megtétele után a sötét játékos órája indul és a világosé megáll.



12. ábra: Sakk játékfelület

2.2.2 Menü

Az ablakon lévő menüből játék közben bármikor újraindítható a játék.

2.2.3 Játékmódok

Single-player játékmód esetén a játékos a világos bábukkal lép, a sötét bábukat egy számítógép vezérelte játékos kezeli.

Multiplayer módban két játékos tud egymás ellen játszani.

2.2.4 Lépések

Mindkét módban az aktuális játékos saját bábujára kattintva megjelennek a bábu lehetséges szabályos lépései. A kijelölt bábut szürke háttérrel jelöljük, a lehetséges szabályos lépéseit pedig kék színnel. Az előző játékos utolsó lépését pedig zöld színnel jelöljük.

Mindhárom lépést a 13. ábra szemlélteti.

13. ábra: Bábu kijelölése és lehetséges lépések megjelenítése

Kijelölt bábu esetén következő lehetőségek vannak:

- Kék mezőre kattintáskor végrehajtódik a kijelölt bábu lépése,
- Másik azonos színű bábura kattintáskor azt a bábut jelöljük ki,
- Üres mezőre, a kijelölt bábura vagy ellenkező színű bábura kattintva megszűnik a kijelölés.

A programban a speciális sakk lépések is lehetségesek:

- sánc: a királyt kijelölve jobbra, illetve balra két mezővel odébb kattintva megtörténik a lépés,
- en passant: a gyalog bábut kijelölve az ellenfél gyalogja mögé kattintva megtörténik a lépés,
- átalakulás: A gyaloggal az ellenfél alapsorára lépve automatikusan vezérré változik.

Ha a játékos sakkban van, csak azok a lépések jelennek meg, amellyel megszűnik a sakk.

Amennyiben nincs ilyen lépés, vagy lejár a játékos órája, a játéknak vége, és a 14. ábrán látható ablak jelenik meg, megjelölve a győztest, ahol lehetőség van új játékot indítani.

14. ábra: Játék vége ablak

3 Fejlesztői dokumentáció

Ez a fejezet részletesen tartalmazza a program felépítését és az osztályok bemutatását.

3.1 Megoldási terv

A szoftvert Java nyelven fejlesztettem, felhasználtam a Java alapkönyvtárat és a grafikus felülethez a Swing keretrendszert, a teszteléshez pedig JUnit keretrendszert használtam.

Megoldási tervem implementálása során az objektum orientált programozás irányelveit követtem.

3.2 A játékprogram

3.2.1 Chess osztály

A program indítóosztálya, itt példányosítom a Menu osztályt, melynek hatására megjelenik a játékmód kiválasztásához szükséges menü.

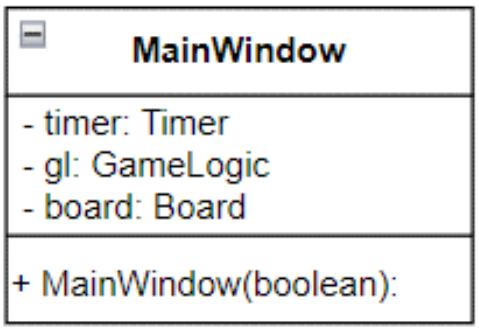
3.2.2 Menu osztály

Az osztályt a JFrame-ből származtatom, példányosítása hatására megjelenik a menü grafikus felület. A start gomb megnyomására az ablak bezáródik, és példányosítom a MainWindow osztályt.

3.2.3 MainWindow osztály

A MainWindow osztályt JFrame-ből származtatom. Tartalmaz egy GameLogic és egy Board objektumot. Megjeleníti a grafikus felületen a játékosok óráit, számítógép vezérelte játékos esetén egy AI szöveget. Ebben az osztályban hozom létre az ablakon lévő menüt.

Az osztály osztálydiagramja a 15. ábrán látható.



15. ábra: Main osztály osztálydiagramja

private final Timer timer:

Az osztály időzítője. Példányosításkor 1000 milliszekundumunként csökkentem az aktuális játékos idejét. Ha letelik az idő, vagy a játéknak vége, megjelenítem a játék vége ablakot és elindítok egy új játékot.

Adattagok:

private final GameLogic gl:

A GameLogic típusú objektum definiálása.

private final Board board:

A Board típusú objektum definiálása.

Függvények:

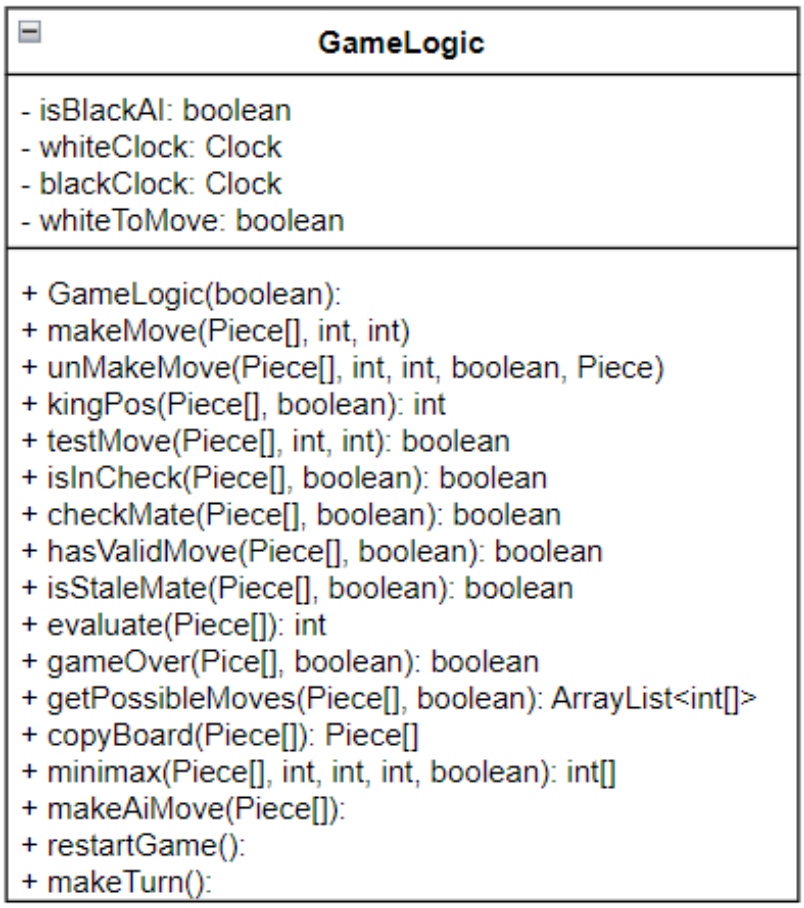
public MainWindow(boolean isBlackAi):

Az osztály konstruktora. A paraméterrel példányosítom a GameLogic és a Board osztályt. Hozzáadom az ablakhoz a játékosok óráit, a menüt és a táblát, és beállítom az ablak ikonját.

3.2.4 GameLogic osztály

Ez a program legfontosabb osztálya. A GameLogic osztály felelős a Board osztály logikájáért. Ennek a segédosztálynak az a feladata, hogy bábuk mozgatását kezelje. Tartalmazza a játékosok óráját. Itt számítom ki a számítógép által vezérelt játékos lépéseit.

Az osztály osztálydiagramja a 16. ábrán látható.



16. ábra: GameLogic osztály osztálydiagramja

Adattagok:

private final boolean isBlackAI:

Ez a változó mutatja, hogy a sötét játékos számítógép által vezérelt.

private final Clock whiteClock:

A világos játékos orájának definiálása.

private final Clock blackClock:

A sötét játékos órájának definiálása.

private boolean whiteToMove:

Azt jelöli, hogy az aktuális játékos világos-e.

Függvények:

public GameLogic(boolean isBlackAi):

Az osztály konstruktora. Paraméterül kapott változóval beállítom az isBlackAI adattag értékét, példányosítom a játékosok óráit, beállítom az aktuális játékost.

public void makeMove(Piece[] board, int from, int to):

Kapott paraméterek:

Piece[] board: Bábukból álló 64 hosszú tömb.

int from: Azt az indexet jelöli a táblán, ahol a bábu áll.

int to: Azt az indexet jelöli a táblán, ahova a bábuval lép.

Ebben a metódusban helyezem át a bábukat a táblán.

Itt kezelem a speciális lépéseket: sánc, en passant, átalakulás.

public void unMakeMove(Piece[] board, int from, int to, boolean hasMoved, Piece toPiece)

Kapott paraméterek:

Piece[] board: Bábukból álló 64 hosszú tömb.

int from: Azt az indexet jelöli a táblán, ahonnét lépett a bábu.

int to: Azt az indexet jelöli a táblán, ahova lépett a bábu.

boolean hasMoved: Azt jelöli, hogy a bábu a lépés előtt mozgott-e már.

Piece toPiece: Az a bábu, ami azon a helyen állt, ahova lépünk.

A makeMove() metódus ellentétje. Ebben a metódusban kerülnek vissza a helyére a bábuk a lépés előtti pozícióba.

public int kingPos(Piece[] board, boolean isWhite):

Kapott paraméterek:

Piece[] board: Bábukból álló 64 hosszú tömb.

boolean isWhite: azt jelöli, hogy az aktuális játékos világos-e.

Segédfüggvény, amivel visszaadom a paraméterben kapott színű játékos a királyának pozícióját. Végigmegyek a paraméterben kapott táblán és ha találok olyan bábut, aminek a típusa KING és megfelelő színű, akkor azt visszaadom.

public boolean testMove(Piece[] board, int from, int to):

Kapott paraméterek:

Piece[] board: Bábukból álló 64 hosszú tömb.

int from: Azt az indexet jelöli a táblán, ahol a bábu áll.

int to: Azt az indexet jelöli a táblán, ahova a bábuval lép.

Segédfüggvény, amivel megadom, hogy a paraméterül kapott lépés szabályos-e. Meghívom a makeMove() metódust. Ezután az unMakeMove() metódus meghívásával visszalépek a bábuval, és, ha sakkban volt, hamisat adok vissza, különben igazat.

public boolean isInCheck(Piece[] board, boolean isWhite):

Kapott paraméterek:

Piece[] board: Bábukból álló 64 hosszú tömb.

boolean isWhite: azt jelöli, hogy az aktuális játékos világos-e.

Segédfüggvény, amivel megadom, hogy a paraméterül kapott játékos sakkban van-e.

Lekérem a játékos királyának pozícióját a kingPos() függvény segítségével. Végigmegyek a táblán, és ha találok olyan bábut, aminek a lehetséges szabályos lépései tartalmazzák a király pozícióját, akkor igazat adok vissza, különben hamisat.

public boolean checkMate(Piece[] board, boolean isWhite)

Kapott paraméterek:

Piece[] board: Bábukból álló 64 hosszú tömb.

boolean isWhite: azt jelöli, hogy az aktuális játékos világos-e.

Segédfüggvény, amivel megadja, hogy a paraméterül kapott játékosnak van-e szabályos lépése, hogy kitérjen a sakkból.

Végigmegyek a táblán, és ha találok olyan paraméterül kapott színű bábut, aminek van lehetséges szabályos lépése, akkor hamisat adok vissza, különben igazat.

public boolean hasValidMove(Piece[] board, int pos):

Kapott paraméterek:

Piece[] board: Bábukból álló 64 hosszú tömb.

int pos: A tábla egy indexe.

Segédfüggvény, amivel megadom, hogy a tábla egy indexén lévő bábunak van-e szabályos lépése.

Végigmegyek a tábla pos indexén lévő bábu lehetséges lépésein, és visszaadom, hogy van-e köztük szabályos lépés.

public boolean isStaleMate(Piece[] board, boolean isWhite):

Kapott paraméterek:

Piece[] board: Bábukból álló 64 hosszú tömb.

boolean isWhite: azt jelöli, hogy az aktuális játékos világos-e.

Segédfüggvény, amivel megadom, hogy a táblán patthelyzet jött-e létre.

Végigmegyek a táblán, és ha találok olyan bábut, aminek van lehetséges szabályos lépése, akkor hamisat adok vissza, különben igazat.

public int evaluate(Piece[] board):

Kapott paraméterek:

Piece[] board: Bábukból álló 64 hosszú tömb.

Segédfüggvény, amivel visszaadom a táblán lévő bábuk értékét. Különböző típusú bábuk értékét a PieceType enumban tárolom.

Végigmegyek a táblán és a világos bábuk értékeinek összegéből kivonom a sötét bábuk értékének összegét.

public boolean gameOver(Piece[] board, boolean isWhite):

Kapott paraméterek:

Piece[] board: Bábukból álló 64 hosszú tömb.

boolean isWhite: azt jelöli, hogy az aktuális játékos világos-e.

Segédfüggvény, amivel megadom, hogy a játéknak vége van-e.

Visszaadom, hogy matt, vagy patt helyzet alakult-e ki.

public ArrayList<int[]> getPossibleMoves(Piece[] board, boolean isWhite)

Kapott paraméterek:

Piece[] board: Bábukból álló 64 hosszú tömb.

boolean isWhite: azt jelöli, hogy az aktuális játékos világos-e.

Segédfüggvény, amivel megadom a paraméterül kapott színű játékos összes lehetséges lépését.

Végigmegyek a táblán és visszaadom a megfelelő színű bábuk összes lehetséges szabályos lépését.

public Piece[] copyBoard(Piece[] board):

Kapott paraméterek:

Piece[] board: Bábukból álló 64 hosszú tömb.

Segédfüggvény, amivel visszaadom a paraméterül kapott tábla másolatát.

Végigmegyek a táblán, és visszaadok egy új táblát a bábuk másolatával.

public int[] minimax(Piece[] board, int depth, int alpha, int beta, boolean maximizingPlayer):

Kapott paraméterek:

Piece[] board: Bábukból álló 64 hosszú tömb.

int depth: Az algoritmus futásának a mélysége.

int alpha: Optimalizációs paraméter.

int beta: Optimalizációs paraméter.

boolean maximizingPlayer: Az algoritmus aktuális ágában a játékos színét jelzi.

Olyan függvény, amivel megadom a paraméterül kapott színű játékos lehetséges legjobb lépését adott mélységig.

Játék vége, vagy 0 depth esetén a tábla kiértékelésével térek vissza.

Igaz maximizingPlayer paraméterrel meghívva egy maximum; hamissal pedig minimum kiválasztást hajtok végre a megfelelő színű játékos lehetséges lépései közül és visszaadom a legjobb lépést.

public void makeAiMove(Piece[] board):

Kapott paraméterek:

Piece[] board: Bábukból álló 64 hosszú tömb.

Végrehajtom a táblán a legjobb lépést.

Lekérem a minimax() függvény értékét a sötét játékos részére, és végrehajtom a visszaadott lépést. A programomban az algoritmus jelenleg 4 mélységig fut, ami a kódban változtatható a depth változó átírásával.

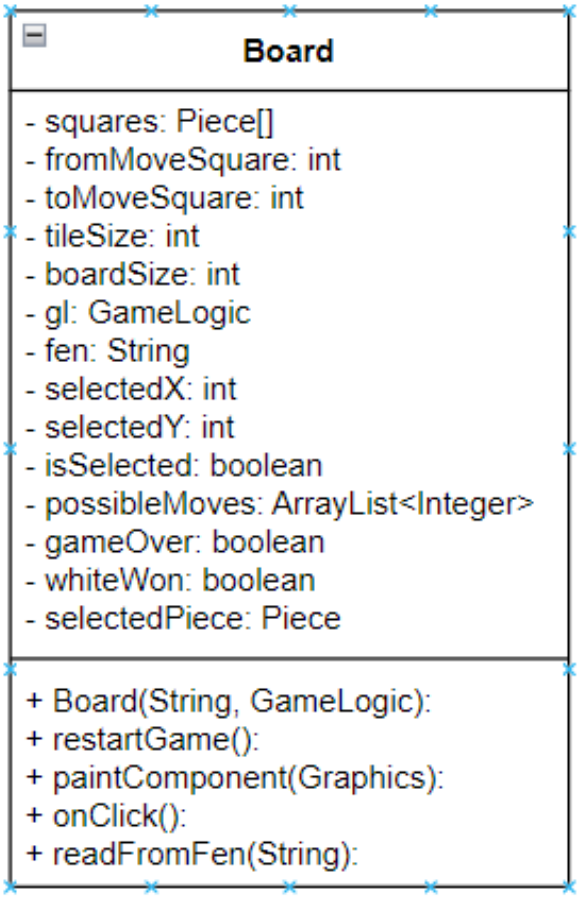
public void restartGame():

Segédfüggvény, amivel újraindítom a játékosok óráit, és a kezdőjátékost beállítom világosra.

3.2.5 Board osztály

Ezzel az osztállyal reprezentálom a táblát, itt tárolom a táblán lévő bábukat és itt kezelem a felhasználó kattintásait.

Az osztály osztálydiagramja a 17. ábrán látható.



17. ábra: Board osztály osztálydiagramja

Adattagok:

private Piece[] squares:

64 hosszú Piece típusú tömb, a tábla mezőin lévő bábukat tartalmazza.

private static int fromMoveSquare, toMoveSquare:

Az éppen nem aktuális játékos utolsó lépésének koordinátáit jelöli.

private final int tileSize:

A táblán lévő mezők méretét jelöli.

private final int boardSize:

A tábla egy sorban vagy oszlopban levő mezők számát jelöli.

private final GameLogic gl:

A paraméterül kapott GameLogic típusú objektumot itt tárolom el.

private final String fen:

Ez alapján töltöm fel a táblát bábukkal.

private int selectedX, selectedY:

A táblán aktuálisan kiválasztott mező koordinátái.

private boolean isSelected:

Azt jelöli, hogy van-e kiválasztott mező a táblán.

private ArrayList<Integer> possibleMoves:

A táblán kiválasztott mezőn lévő bábu lehetséges lépéseit tartalmazza.

private boolean gameOver:

A játék végét jelöli.

private boolean whiteWon:

Játék vége esetén a nyertes játékos színét jelöli.

private Piece selectedPiece:

A kijelölt mezőn lévő bábut jelöli.

Függvények:

public Board(String startFen, GameLogic gl):

Az osztály konstruktora. A paraméterekkel beállítom a fen és gl adattagokat, majd az ablak méretét. Ezután a readFromFen() függvény meghívásával feltöltöm a táblát

bábukkal. Következő lépésben egy MouseListener felhasználásával kezelem a felhasználó kattintásait. Ebben a függvényben hívom meg az onClick() metódust, ahol végrehajtom a megadott lépéseket. Ezen kívül itt kérem le a kijelölt bábu lehetséges szabályos lépéseit.

public void restartGame():

Olyan függvény, amivel törlöm a táblán lévő bábukat eltároló tömb tartalmát. Ebben a függvényben újból feltöltöm a táblát a kezdőállásba a readFromFen() függvény meghívásával.

@Override

protected void paintComponent(Graphics g):

A JPanelből származó függvény. A tábla kirajzolását valósítja meg:

Itt rajzolom ki a tábla hátteréhez a négyzeteket.

Kirajzolom a kijelölt bábu hátterének a színét.

Itt rajzolom ki a sorokat és oszlopokat jelölő betűket és számokat.

Kijelölt bábu esetén kirajzolom a bábu lehetséges szabályos lépéseit.

Kirajzolom az előző játékos utolsó lépését.

A táblán lévő bábuk képeit is itt rajzolom ki.

private void onClick(int prevSelectedX, int prevSelectedY):

Segédfüggvény, paraméterben megkapom az előző kattintás koordinátáit. Ebben a függvényben történik a szabályos lépések végrehajtása. Beállítom az aktuális lépés koordinátáit a kirajzoláshoz. Itt végzem a sakk-, matt- és patthelyzet ellenőrzését. Single-player módban indított játék esetén történik a számítógép által vezérelt játékos lépése.

public void readFromFen(String startFen):

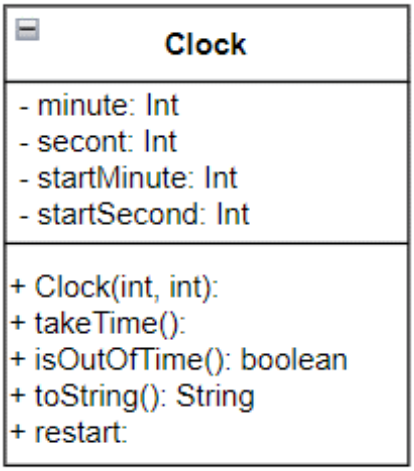
Olyan segédfüggvény, amiben a paraméterül kapott szöveg alapján helyezem el a bábukat a táblán.

A megvalósításhoz a szabványos FEN jelölést használtam. A függvényben végigmegyek a paraméterül kapott szövegen és beállítom az aktuális karakterhez megfelelő bábut a tábla megfelelő pozíciójára.

3.2.6 Clock osztály

Ezzel az osztállyal kezelem a játékosok óráit.

Az osztály osztálydiagramja a 18. ábrán látható.



18. ábra: Clock osztály osztálydiagramja

Adattagok:

private int minute:

Az óra aktuális perc értékét tárolja.

private int second:

Az óra aktuális másodperc értékét tárolja.

private final int startMinute:

Az óra kezdő perc érdékést tárolja.

private final int startSecond:

Az óra kezdő másodperc értékét tárolja.

Függvények:

public Clock(int minute, int second):

Az osztály konstruktora. Itt állítom be az óra aktuális és kezdő perc és másodperc értékeit.

public void takeTime():

Segédfüggvény, amivel a játékos óráját számoltatom vissza.

public boolean isOutOfTime():

Visszaadom, hogy az idő lejárt-e.

@Override

public String toString():

Ez a függvény szöveges formátumban adja vissza az osztály perc és másodperc értékét 2 számjegyre formázva.

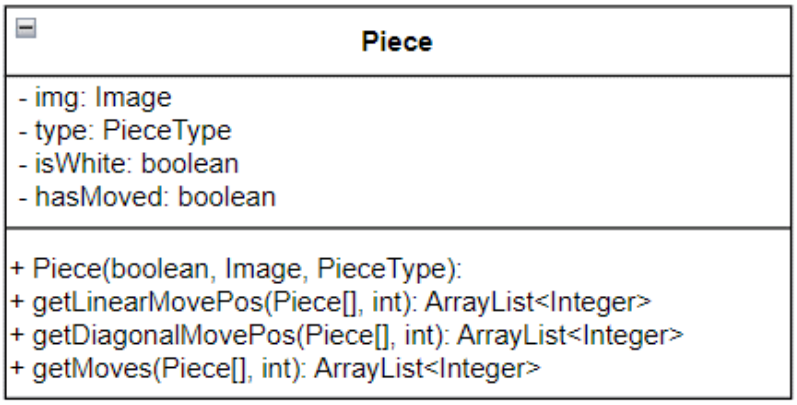
public void restart():

Segédfüggvény, amivel beállítom a minute és second adattagokat a kezdő értékre.

3.2.7 Piece osztály

A Piece osztály tartalmaz minden olyan függvényt és adattagot, amelyre mindegyik bábunak szüksége van. Ebből az ősosztályból származnak az egyes alosztályok, melyek a konkrét bábukat reprezentálják: Pawn, Bishop, Knight, Rook, Queen és King osztályok mindegyike ebből az osztályból származik.

Az osztály osztálydiagramja a 19. ábrán látható.



19. ábra: Piece osztály osztálydiagramja

Adattagok:

private final Image img:

A bábuk megjelenítéséhez használt kép.

private final PieceType type:

A bábu típusát tárolja a PieceType enum szerint.

private boolean isWhite:

Itt tárolom el, hogy a bábu világos-e.

private boolean hasMoved:

Itt tárolom el, hogy a bábu lépett-e.

Függvények:

public Piece(boolean isWhite, Image img, PieceType type):

Az osztály konstruktora. A paraméterekkel beállítom az isWhite, img és type adattagokat.

public static ArrayList<Integer> getLinearMovePos (Piece[] board, int pos):

A Piece osztály statikus függvénye. A függvénnyel visszaadom a paraméterül kapott táblán a paraméterül kapott pozícióból lehetséges vízszintes és függőleges lépéseket egy Integer típusú listában.

public static ArrayList<Integer> getDiagonalMovePos(Piece[] board, int pos):

A Piece osztály statikus függvénye. A függvénnyel visszaadom a paraméterül kapott táblán a paraméterül kapott pozícióból lehetséges átlós lépéseket egy Integer típusú listában.

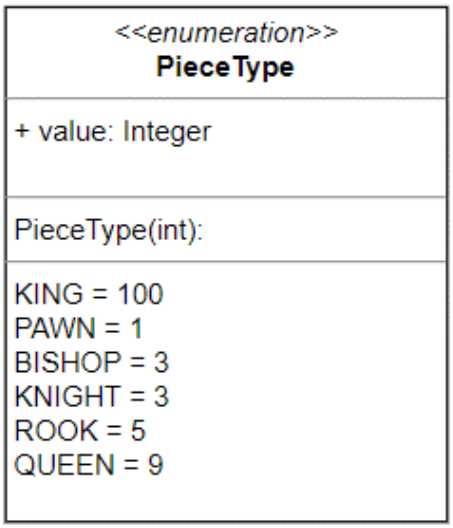
public abstract ArrayList<Integer> getMoves(Piece[] board, int pos):

A Piece osztály absztrakt függvénye. A származtatott osztályok lehetséges lépéseit adom vissza ezzel a függvénnyel.

3.2.8 PieceType enum

Ebben az enumban tárolom a bábuk lehetséges típusait és a hozzájuk tartozó értéket a számítógép által vezérelt játékos használatához.

Az osztály osztálydiagramja a 20. ábrán látható.



20. ábra: PieceType enum osztálydiagramja

Adattagok:

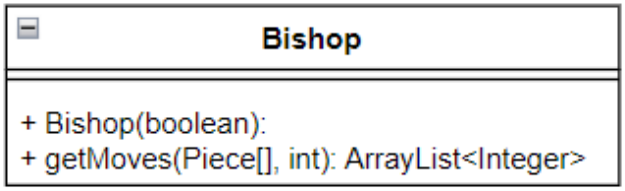
public final int value:

A típus értékét jelöli.

3.2.9 Bishop osztály

A Piece osztályból származó osztály, ezzel valósítom meg a futó bábut.

Az osztály osztálydiagramja a 21. ábrán látható.



21. ábra: Bishop osztály osztálydiagramja

Függvények:

public Bishop(boolean isWhite):

Az osztály konstruktora. Itt meghívom a Piece osztály konstruktorát a paraméterül kapott isWhite értékkel, a megfelelő színű bábu képével és a bábu típusával.

@Override

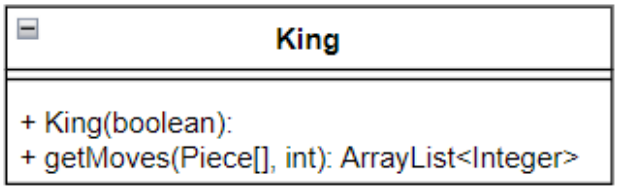
public ArrayList<Integer> getMoves(Piece[] board, int pos):

A Piece osztályból származó metódus. Mivel a futó csak átlósan mozoghat, ezért itt a Piece osztály statikus getDiagonalMovePos() függvényét hívom meg a megfelelő paraméterekkel és adom vissza.

3.2.10 King osztály

A Piece osztályból származó osztály, ezzel valósítom meg a király bábut.

Az osztály osztálydiagramja a 22. ábrán látható.



22. ábra: King osztály osztálydiagramja

Függvények:

public King(boolean isWhite):

Az osztály konstruktora. Itt meghívom a Piece osztály konstruktorát a paraméterül kapott isWhite értékkel, a megfelelő színű bábu képével és a bábu típusával.

@Override

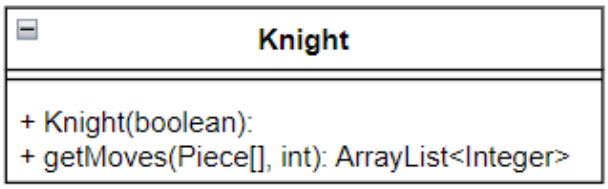
public ArrayList<Integer> getMoves(Piece[] board, int pos):

A Piece osztályból származó metódus. Visszaadom egy listában a szomszédos mezők koordinátáit, ahol nem áll a királlyal azonos színű bábu. Szabályos sáncolás esetén a lista tartalmazza még a királytól jobbra vagy balra található második mező koordinátáját.

3.2.11 Knight osztály

A Piece osztályból származó osztály, ezzel valósítom meg a huszár bábut.

Az osztály osztálydiagramja a 23. ábrán látható.



23. ábra: Knight osztály osztálydiagramja

Függvények:

public Knight(boolean isWhite):

Az osztály konstruktora. Itt meghívom a Piece osztály konstruktorát a paraméterül kapott isWhite értékkel, a megfelelő színű bábu képével és a bábu típusával.

@Override

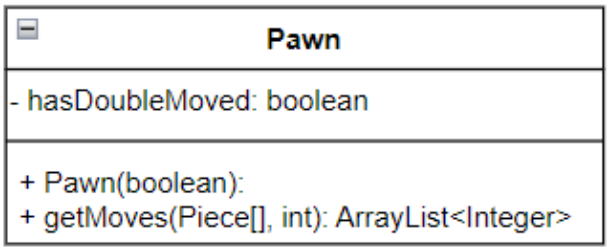
public ArrayList<Integer> getMoves(Piece[] board, int pos):

A Piece osztályból származó metódus. Visszaadom egy listában a táblán a paraméterül kapott pozícióból azokat a koordinátákat, ahova a huszár léphet és ahol nincs vele megegyező színű bábu.

3.2.12 Pawn osztály

A Piece osztályból származó osztály, ezzel valósítom meg a gyalog bábut.

Az osztály osztálydiagramja a 24. ábrán látható.



24. ábra: Pawn osztály osztálydiagramja

Adattagok:

private boolean hasDoubleMoved:

Azt jelölöm, hogy a gyalog az előző lépésben a kiindulási helyéről két mezőt lépett-e.

Függvények:

public Pawn(boolean isWhite):

Az osztály konstruktora. Itt meghívom a Piece osztály konstruktorát a paraméterül kapott isWhite értékkel, a megfelelő színű bábu képével és a bábu típusával.

@Override

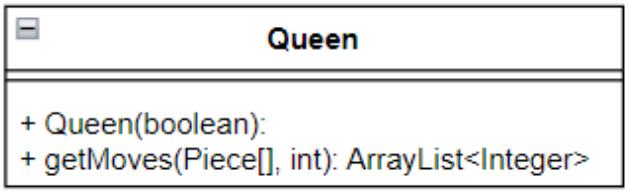
public ArrayList<Integer> getMoves(Piece[] board, int pos):

A Piece osztályból származó metódus, amivel egy listát adok vissza a lehetséges lépések koordinátáival. A lista tartalmazza a tábla paraméterül kapott koordinátáján lévő gyalog előtt lévő mező koordinátáját, ha az a mező üres. Ha a gyalog a kiindulási mezőn tartózkodik, akkor a lista tartalmazza a gyalog előtt kettő mezővel lévő mező koordinátáját is. Ha van a gyalog előtti átlós mezőn ellenfél bábuja, akkor annak a mezőnek a koordinátáját is tartalmazza a lista. Szabályos en-passant lépés esetén a lista tartalmazza a leütni kívánt ellenfél gyalogja mögötti mező koordinátáját is.

3.2.13 Queen osztály

A Piece osztályból származó osztály, ezzel valósítom meg a vezér bábut.

Az osztály osztálydiagramja a 25. ábrán látható.



25. ábra: Queen osztály osztálydiagramja

Függvények:

public Queen(boolean isWhite):

Az osztály konstruktora. Itt meghívom a Piece osztály konstruktorát a paraméterül kapott isWhite értékkel, a megfelelő színű bábu képével és a bábu típusával.

@Override

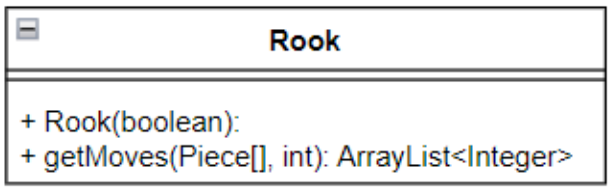
public ArrayList<Integer> getMoves(Piece[] board, int pos):

A Piece osztályból származó metódus. Visszaadok egy listát, ami tartalmazza a Piece osztály getDiagonalMovePos() függvényéből kapott átlós és a getLinearMovePos() függvényből kapott egyenes koordinátákat.

3.2.14 Rook osztály

A Piece osztályból származó osztály, ezzel valósítom meg a bástya bábut.

Az osztály osztálydiagramja a 26. ábrán látható.



26. ábra: Rook osztály osztálydiagramja

Függvények:

public Rook(boolean isWhite):

Az osztály konstruktora. Itt meghívom a Piece osztály konstruktorát a paraméterül kapott isWhite értékkel, a megfelelő színű bábu képével és a bábu típusával.

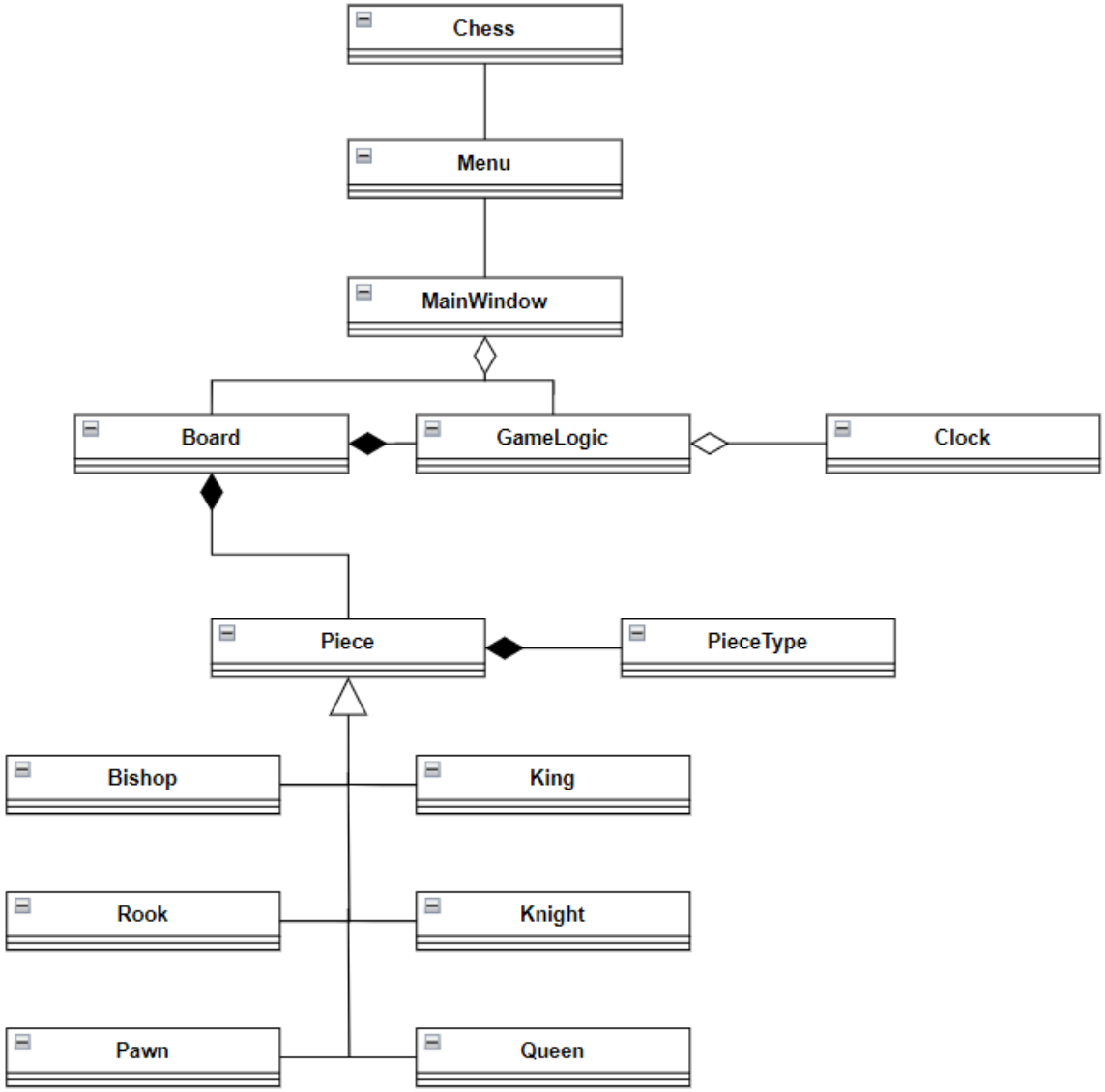
@Override

public ArrayList<Integer> getMoves(Piece[] board, int pos):

A Piece osztályból származó metódus. Visszaadok egy listát, ami tartalmazza a Piece osztály getLinearMovePos() függvényéből kapott egyenes koordinátákat.

3.3 Osztályok kapcsolata

A programomban lévő osztályok kapcsolatát a 27. ábra mutatja be.



27. ábra: Osztályok kapcsolata

4 Tesztelés

A teszteléshez fekete doboz és fehér doboz tesztelési módszereket használtam. A fehér doboz tesztelést JUnit 5.8.1 keretrendszerrel valósítottam meg.

4.1 Fekete doboz tesztek

Program indítása, leállítása

Ellenőriztem, hogy a felhasználói dokumentációban leírtak alapján a program megfelelően elindul és bezárul.

Játék újraindítása menüből

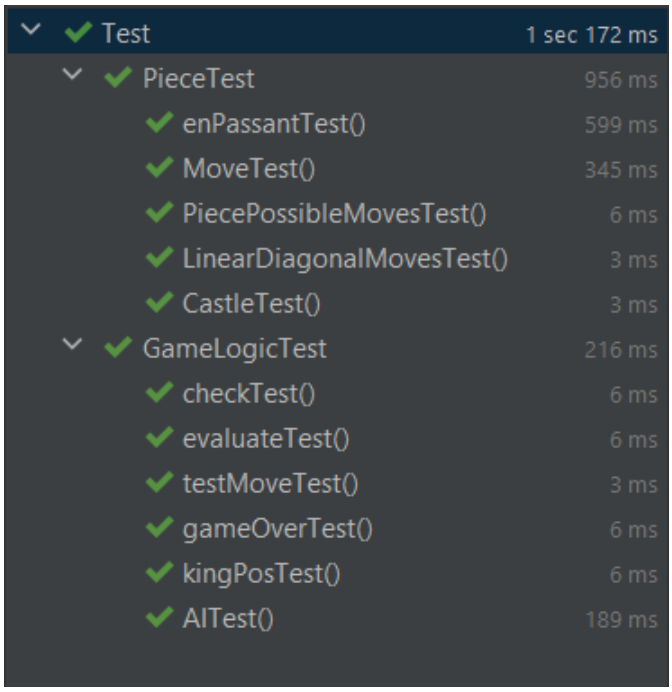
Ellenőriztem, hogy a restart menüpontra kattintva a játék minden esetben megfelelően újraindul.

Játék vége

Ellenőriztem, hogy sakk-matt, patthelyzet vagy idő lejárta esetén a játéknak vége, és a program megfelelően jelzi a győztes játékost.

4.2 Fehér doboz tesztek

A sikeresen lefutó JUnit tesztek a 28. ábrán láthatóak.



28. ábra: JUnit tesztek eredménye

4.2.1 Bábuk lépéseinek tesztelése

Tesztfájl: PieceTest.java

Ebben a fájlban azt vizsgáltam, hogy a bábuk lépéseihez használt függvények megfelelő értékkel térnek-e vissza.

Függvények:

public void PiecePossibleMovesTest():

A tesztelés folyamán megvizsgálom, hogy egy üres táblán elhelyezett bábu getMoves() függvénye megfelelő értékkel tért-e vissza. A teszt során megvizsgáltam mind a 6 különböző bábutípus lehetséges lépéseit.

public void CastleTest():

Ebben a függvényben a sáncolás lépést tesztelem. Megvizsgáltam, hogy a világos és sötét játékos is mindkét oldalra tud sáncolni és megfelelő helyre kerülnek a bábuk.

public void enPassantTest():

Ebben a fügvényben az en-passant lépést tesztelem. Megvizsgáltam, hogy a világos és sötét játékos is képes végrehajtani a lépést és a bábuk a megfelelő helyre kerülnek.

public void LinearDiagonalMovesTest():

Megvizsgáltam egy üres táblán elhelyezett vezérrel, hogy a Piece osztály statikus metódusai megfelelő értékekkel térnek-e vissza.

4.2.2 GameLogic osztály függvényeinek tesztelése

Tesztfájl: GameLogicTest.java

Ebben a fájlban azt vizsgáltam, hogy a GameLogic osztály metódusai megfelelően működnek, az elvárt értékkel térnek-e vissza.

Függvények:

public void MoveTest():

Megvizsgáltam, hogy egy bábuval való lépés esetén megfelelően változik a táblán elhelyezkedő bábuk koordinátája.

public void testMoveTest():

Ellenőriztem, hogy szabályos, illetve szabálytalan lépés esetén a megfelelő értéket adta-e vissza a függvény.

public void kingPosTest():

Ellenőriztem, hogy a világos és sötét játékos esetén is a megfelelő értéket adta vissza a kingPos() függvény a kezdőpozícióból és lépés után is.

public void checkTest():

Ellenőriztem, hogy sakk és sakk-matt esetén helyesen működik-e az isInCheck() függvény.

public void evaluateTest():

Ellenőriztem, hogy alaphelyzetben és egy bábu leütése után is megfelelő értéket ad vissza az evaulate() függvény.

public void AITest():

Ellenőriztem, hogy a táblán egy egyszerű felállásban a minimax() függvény a megfelelő értékkel, vagyis a legjobb lépéssel tér-e vissza.

public void gameOverTest():

Ellenőriztem, hogy különböző pozíciókban, a gameOver() és az isInCheck() függvény megfelelő értékekkel tér-e vissza.

5 Összefoglalás

Szakdolgozatom célja egy olyan játékszoftver megírása volt, ami felhasználói és fejlesztői szempontból is megfelelő.

A dokumentációban bemutatom felhasználói és fejlesztői oldalról a program működését.

Felhasználói szempontból a játék szórakoztató, tanulságos élményt nyújt. A két játékmód lehetőséget ad, hogy egy másik játékossal vagy a számítógép által vezérelt játékos ellen is kipróbálhassuk tudásunkat és fejleszthessük gondolkodásmódunkat.

A fejlesztő egy jól strukturált objektum orientált programkóddal találkozhat, amivel fejlesztheti szakmai tudását.

A szoftver létrehozásakor az egyetemi tanulmányaim során megismert és elsajátított ismeretek alkalmazására törekedtem, figyelve arra, hogy a programom megfeleljen a szakdolgozat követelményeinek.

A program fejlesztése sok izgalmas kihívás elé állított és örömmel töltött el egy-egy probléma megoldása. Úgy érzem, hogy szakmailag sokat fejlődtem, amit későbbi munkáim során alkalmazni tudok.

5.1 Továbbfejlesztési lehetőségek

5.1.1 Heurisztika továbbfejlesztése

A program jelenleg a minimax algoritmust használja a legjobb lépés kiválasztásához. A későbbiekben szeretnék többféle algoritmust is implementálni, illetve az evaluate függvényt továbbfejleszteni.

5.1.2 Felhasználói felület továbbfejlesztése

A programom egyszerű grafikus felületét a későbbiekben tervezem tovább fejleszteni.

5.1.3 Több sakkvariáns implementálása

A hagyományos sakkon kívül szeretnék implementálni többféle sakkvariánst a jövőben.

6 Irodalomjegyzék

[1] https://hu.wikipedia.org/wiki/Sakk (2023.05.27.)

[2] https://www.chess.com/terms/fen-chess (2023.05.27.)

[3] https://www.sakk.hu/help/ (2023.05.27.)

[4] https://www.chessprogramming.org/Main_Page (2023.05.27.)