Sakk

❖ Table

- `public void setState(String FEN)`

  Egy String-et kap argumentumként ami egy sakktábla állapotát reprezentálja, és ezután felállítja a táblát ez alapján. A string Forsyth–Edwards Notation szerint tárolja a táblát, ezt a függvény nem ellenőrzi.

- `public String getState()`

  Egy String-et ad vissza, ami a pálya jelenlegi állását reprezentálja Forsyth–Edwards Notation szerint.

- `public Table(String fileName,JFrame jf,boolean bot)`

  A class egyik konstruktora, ami kap egy String-et, amit egy filenévként kezel, és abból tölti be a tábla állását, egy JFramet, amire a táblát kirajzolja, és egy booleant ami azt dönti el, hogy a játék egy bot ellen fog-e történni.

- `public Table(JFrame jf,boolean bot)`

  A class egyik konstruktora, ami kap egy JFramet, amire a táblát kirajzolja, és egy booleant ami azt dönti el, hogy a játék egy bot ellen fog-e történni. A tábla állását egy előre meghatározott fileból olvassa be.

- `public void kirajz(JFrame frame)`

  Újra rajzolja az egész frame tartalmát.

- `public void wasclicked(Piece temp)`

  Amikor egy bábura kattint a felhasználó, akkor ez a függvény hívódik meg, és kezeli, hogy az adott bábú lépett-e vagy csak ki lett jelölve.

- `int didBlackWin()`

  Ha a fehér bábukkal nem lehet legális lépést tenni, a fehér körében, megnézi, hogy patt, vagy a fekete bábukkal rendelkező játékos nyert.

- `int didWhiteWin()`

  Ha a fekete bábukkal nem lehet legális lépést tenni, a fekete körében, megnézi, hogy patt, vagy a fehér bábukkal rendelkező játékos nyert.

- `int gameHasEnded()`

  Megnézi, hogy a játék valamilyen szabály alapján véget ért-e

- `public boolean ILikeToMoveItMoveIt(Position from,Position to)`

  Két pozíciót kap, hogy honnan és hova lépnek a táblán, és ha a lépés legális, megteszi azt.

❖ Game

- `void saveTheGame()`
  Nyit egy új ablakot amivel a felhasználó elmentheti a tábla állását tetszőleges néven.

- `void loadGame()`
  Nyit egy új ablakot, amin a felhasználó kiválaszthatja melyik mentést szeretné betölteni (max 20 mentés)

- `void newGame()`
  Nyit egy új ablakot, amin a felhasználó meghatározhatja milyen beállításokkal szeretné az új játékot elkezdeni, majd elkezdheti azt.

- `public void menu()`
  Az ablak tetején lévő menüt hozza létre, és az abban lévő menüpontokat.

- `public static void main(String[] args)`
  A program main funkciója, itt kezdődik a futás.

❖ Piece extends JLabel

- `public char getchar()`
  Visszaadja a bábú betűs reprezentációját.

- `void moveTo(int x1,int y1) throws cannotMoveThere`
  A bábú kap egy x és egy y koordinátát ami a tábla egy mezőjét reprezentálja, és ha ez egy legális lépés, odalép.

- `public boolean givesCheckOn(int x1,int y1)`
  Visszaadja, hogy ha a megadott x és y koordinátákon állna az ellenfél királya sakkot adna-e neki.

- `public boolean KingWontBeInCheck(Position takenPiece)`
  Ha az adott bábú az argumentumban kapott pozícióra lépne, ez a függvény meghatározza, hogy a király nem lesz e sakkban, tehát hogy a lépés ilyen szempontból legális.

- `public boolean canMoveTo(int x1,int y1)`
  Meghatározza, hogy a bábú tud-e lépni a az adott x, y koordinátájú mezőkre.

- `public LinkedList<Position> getLegalMoves()`
  Egy láncolt listában visszaadja az összes legális pozíciót, ahova léphet az adott bábú

- `public Piece(int x1,int y1,boolean iw,String kepnev,Table t1)`
  Konstruktor, megkapja az x,y koordinátáit, ahol a bábú áll, hogy fehér-e a bábú, hogy mi a neve a kép filenak ami grafikusan reprezentálja a bábút.
- Illetve minden bábúban implementálva van a MouseListener egy implementációja, JavaMouseDeer néven.

- ## King extends Piece

  A Piece függvényein kívül implementálja:

  - `public boolean isInCheckOn(int x1,int y1)`

    Visszaadja, hogy a megadott x és y koordinátákon sakkban lenne-e.

  - `public boolean isInCheck()`

    Visszaadja, hogy éppen sakkban van-e

- ## Pawn extends Piece
  - A Piece függvényein kívül semmit mást nem implementál.
- ## Rook extends Piece
  - A Piece függvényein kívül semmit mást nem implementál.
- ## Queen extends Piece
  - A Piece függvényein kívül semmit mást nem implementál.
- ## Bishop extends Piece
  - A Piece függvényein kívül semmit mást nem implementál.
- ## Knight extends Piece
  - A Piece függvényein kívül semmit mást nem implementál.
- ## Empty extends Piece
  - A Piece függvényein kívül semmit mást nem implementál.
  - Egy üres mező reprezentálására van.
- ## BasicBot

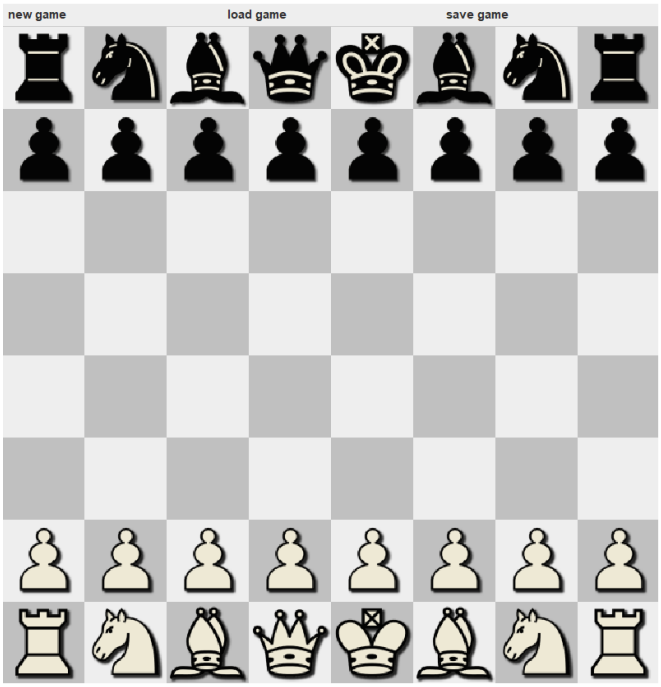
  - `public void makeBasicMove()`

    Egy szabályos lépést tesz a táblán. Randomra választja ki, melyik bábúval, és hova lép.
    Ha 100-szor próbálkozott kiválasztani egy random bábut amivel lépni akar, és egyiknek sem volt legális lépése, akkor abbahagyja a próbálkozást, ez nagyon kicsi valószínűséggel történik meg, ha még vannak lehetséges lépései az adott színnek.

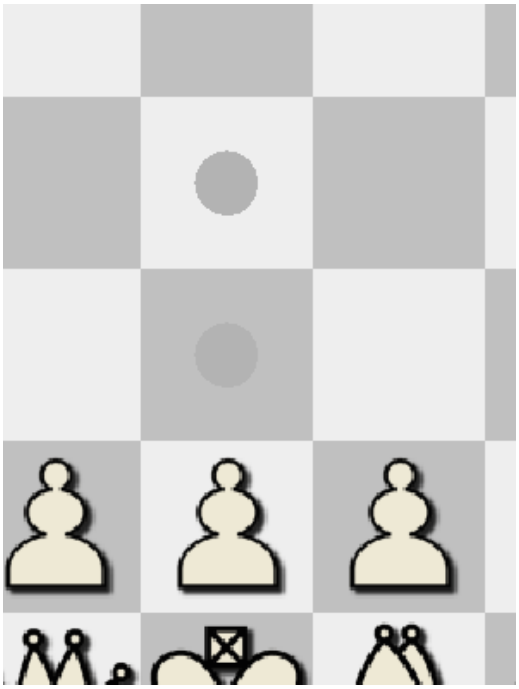
- ## Position
  - Egy x és egy y koordinátát tartalmaz amivel a pálya egyik mezőjét adja meg
- ## cannotMoveThere extends Throwable
  - Egy Throwable class amit a bábuk MoveTo függvényei dobnak, ha az adott lépés illegális.

Felhasználói dokumentáció

A program megnyitásakor a következő látható az ablakban:



Bármelyik bábóra kattintással a táblán megjelenik, hogy az a bábú hova tud lépni:

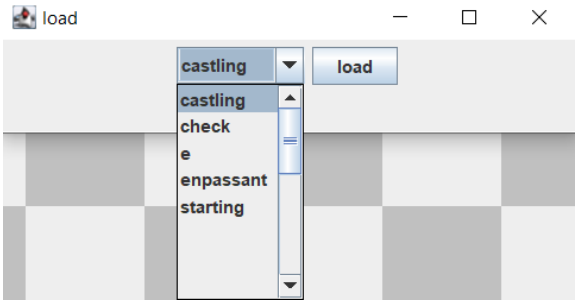


Az ablak tetején megjelenő menü opciói:

- a new game opcióra kattintva az alábbi ablak jelenik meg:

A bal oldali kettő leugró füllel lehet beállítani, hogy bot ellen vagy ember-ember ellen akar játszani, és, hogy melyik színnel, ez csak akkor számít igazából, bot elleni játékról van szó.

- A load game opcióra kattintva a következő ablak jelenik meg:



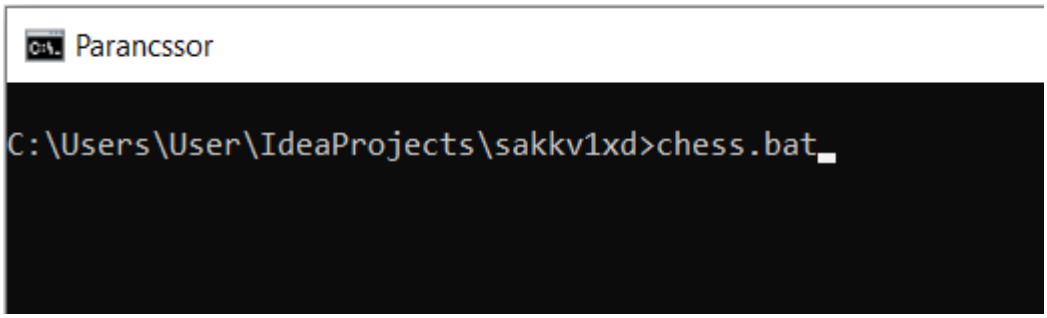
Itt a leugró ablakból kiválasztható, hogy melyik mentést töltse be a program.

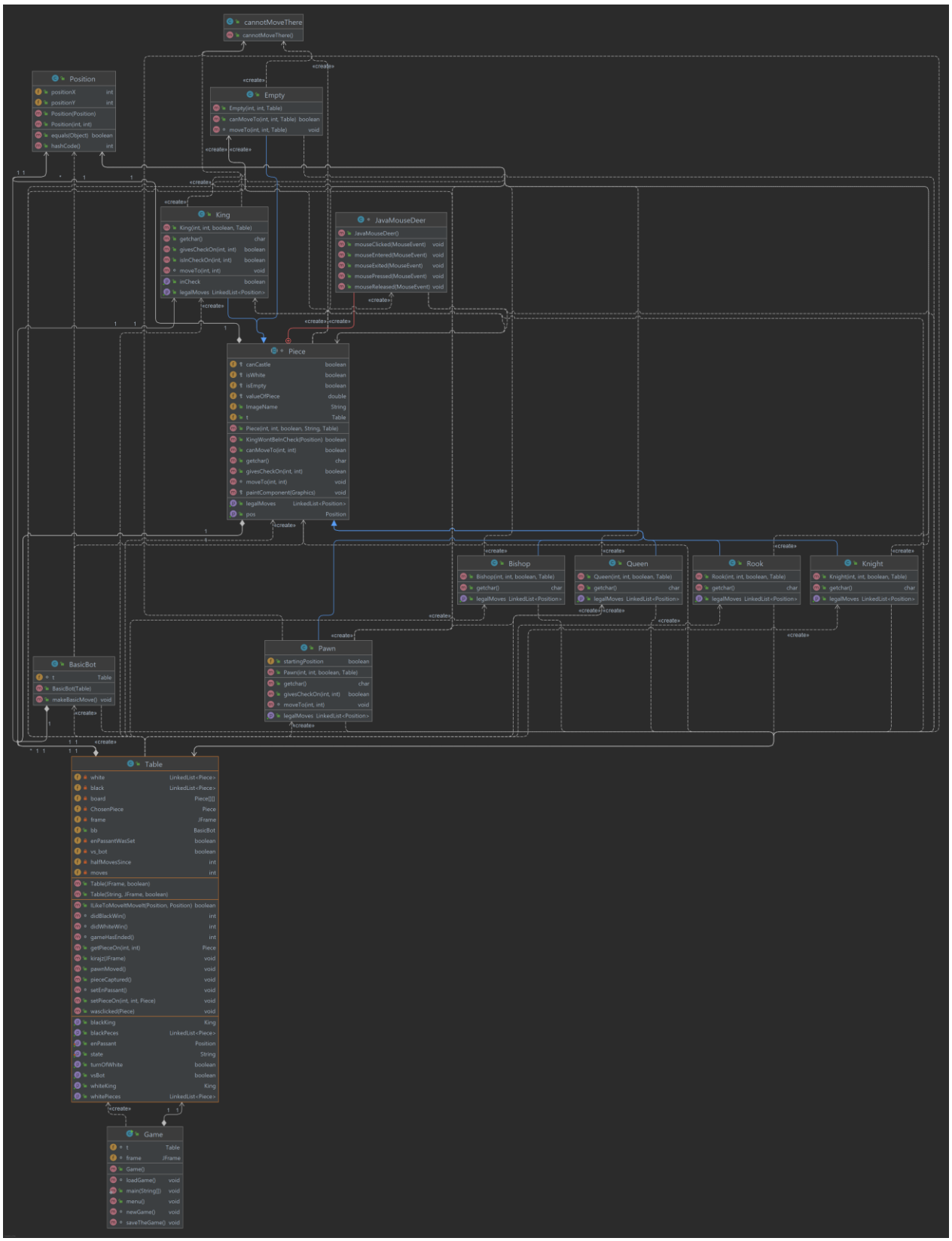
- A save game funkció a következő ablakot hozza be:

Ahol a fehér mezőbe (akol most mentes1 áll) lehet beírni milyen néven szeretnénk, hogy a program mentse a jelenlegi táblaállást.

A játék maga egy sakk játék amiben csak legális lépéseket lehet tenni, ebbe beletartozik a sáncolás, az enpassant. A játék véget érhet, ha az egyik szín a saját körében nem tud legális lépést tenni, vagy az 50 lépés szabály szerint.

- A játékot a chess.bat file futtatásával lehet elindítani.




cannotMoveThere
cannotMoveThere()
Position
positionX int
positionY int
Position(Position)
Position(int, int)
equals(Object) boolean
hashCode() int
Empty
Empty(int, int, Table)
canMoveTo(int, int, Table) boolean
moveTo(int, int, Table) void
King
King(int, int, boolean, Table)
getchar() char
givesCheckOn(int, int) boolean
isInCheckOn(int, int) boolean
moveTo(int, int) void
inCheck boolean
legalMoves LinkedList<Position>
JavaMouseDeer
JavaMouseDeer()
mouseClicked(MouseEvent) void
mouseEntered(MouseEvent) void
mouseExited(MouseEvent) void
mousePressed(MouseEvent) void
mouseReleased(MouseEvent) void
Piece
canCastle boolean
isWhite boolean
isEmpty boolean
valueOfPiece double
ImageName String
t Table
Piece(int, int, boolean, String, Table)
KingWontBeInCheck(Position) boolean
canMoveTo(int, int) boolean
getchar() char
givesCheckOn(int, int) boolean
moveTo(int, int) void
paintComponent(Graphics) void
legalMoves LinkedList<Position>
pos Position
Bishop
Bishop(int, int, boolean, Table)
getchar() char
legalMoves LinkedList<Position>
Queen
Queen(int, int, boolean, Table)
getchar() char
legalMoves LinkedList<Position>
Rook
Rook(int, int, boolean, Table)
getchar() char
legalMoves LinkedList<Position>
Knight
Knight(int, int, boolean, Table)
getchar() char
legalMoves LinkedList<Position>
Pawn
startingPosition boolean
Pawn(int, int, boolean, Table)
getchar() char
givesCheckOn(int, int) boolean
moveTo(int, int) void
legalMoves LinkedList<Position>
BasicBot
t Table
BasicBot(Table)
makeBasicMove() void
Table
white LinkedList<Piece>
black LinkedList<Piece>
board Piece[][]
ChosenPiece Piece
frame JFrame
bb BasicBot
enPassantWasSet boolean
vs_bot boolean
halfMovesSince int
moves int
Table(JFrame, boolean)
Table(String, JFrame, boolean)
ILikeToMoveItMoveIt(Position, Position) boolean
didBlackWin() int
didWhiteWin() int
gameHasEnded() int
getPieceOn(int, int) Piece
kiraj2(JFrame) void
pawnMoved() void
pieceCaptured() void
setEnPassant() void
setPieceOn(int, int, Piece) void
wasclicked(Piece) void
blackKing King
blackPeces LinkedList<Piece>
enPassant Position
state String
turnOfWhite boolean
vsBot boolean
whiteKing King
whitePieces LinkedList<Piece>
Game
t Table
frame JFrame
Game()
loadGame() void
main(String[]) void
menu() void
newGame() void
saveTheGame() void
«create»