# *DIPLOMAMUNKA*

Tóth Antal Lajos

Debrecen

2008

Debreceni Egyetem

Informatika Kar

# Alkalmazás fejlesztés JAVA nyelven

Témavezető:

Dr. Boda István

Egyetemi docens

Készítette:

Tóth Antal Lajos

Programtervező matematikus

Debrecen

2008

# Tartalomjegyzék


**BEVEZETÉS ... 1**

**1. FEJEZET ... 2**

Sakk program ... 2
Bábuk ... 4
Lépés ... 9
Adott állás megadása ... 11
PGN ... 13
Mentés ... 13
Megnyitás ... 16
Navigáló gombok ... 17
Automatikus lépés ... 19

**2. FEJEZET ... 20**

Konvertálás PGN formátumra és formátumról ... 20
PGN fájlba írás ... 25
PGN fájlból olvasás ... 29
Tábla leképezése, kirajzolása ... 33
Adott állás megadása ... 38
Sakk, matt, patt vizsgálat ... 40
Automatikus lépés – Minimax algoritmus alfa-béta vágással ... 42
Néhány főbb osztály ismertetése ... 48

**ÖSSZEGZÉS ... 51**

**IRODALOMJEGYZÉK ... 52**

# Ábrajegyzék

*kezdő állás* .......... 2
*sötét nyert* .......... 3
*sáncolás* .......... 4
*királynő lehetséges lépései* .......... 5
*bástya lehetséges lépései* .......... 6
*futó lehetséges lépései* .......... 7
*ló lehetséges lépései* .......... 8
*menetközbeni ütés (en passant)* .......... 9
*lépésajánlat* .......... 11
*adott állás megadása* .......... 12
*mentés* .......... 15
*mentés - fájlválasztó* .......... 16
*megnyitás (több játszmát tartalmazó PGN fájl)* .......... 17
*mozgató gombok* .......... 18
*alfa-béta vágás* .......... 44

# BEVEZETÉS

A Java nyelv lehetőségeit egy alkalmazáson keresztül szeretném bemutatni. Ez az alkalmazás egy játék: a sakk. A sakk - egyes felmérések alapján - Európában a legjobban elterjedt társasjáték. Max Weiss német sakkozó szerint a sakk a társasjátékok királya.

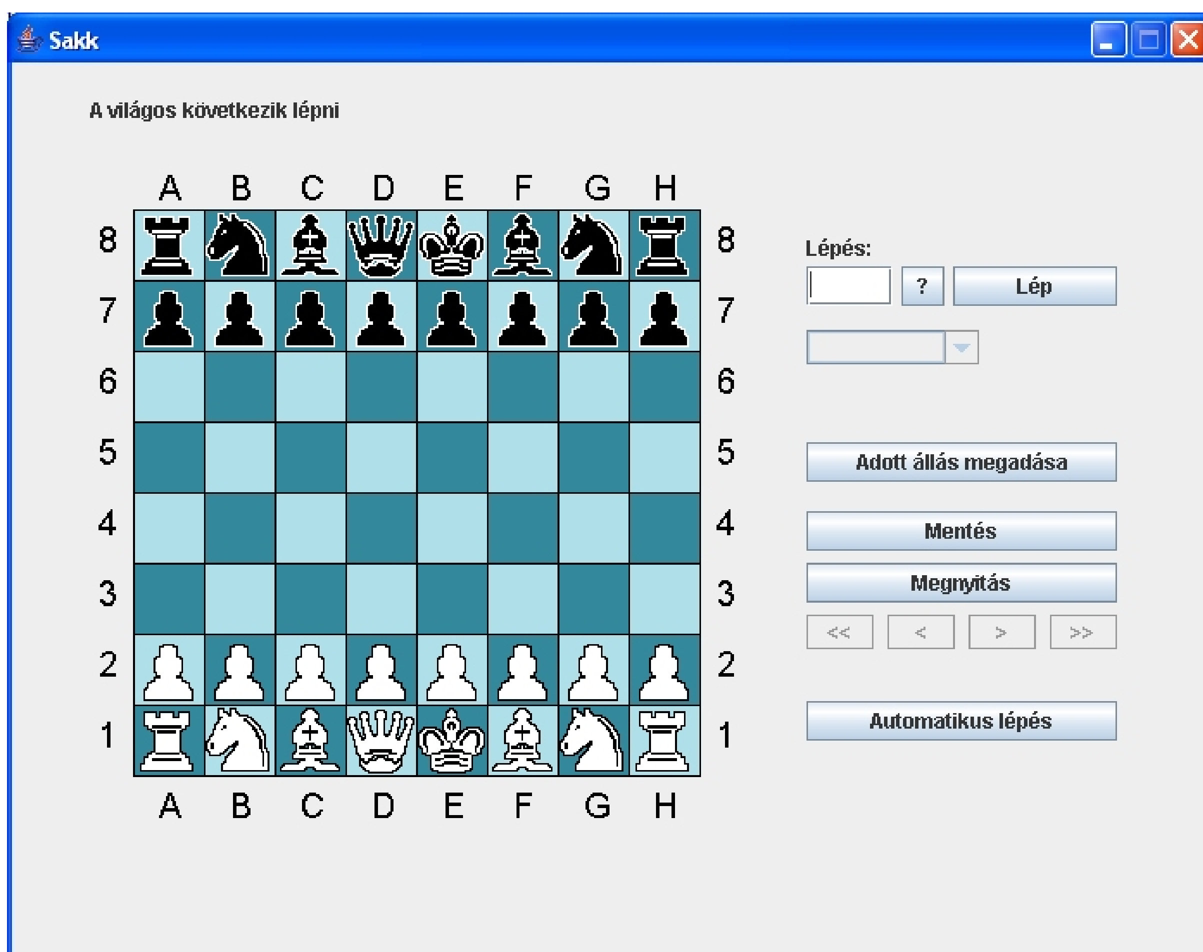
Az alkalmazás képes a szabályos sakkjátszmák lejátszását lehetővé tenni, játszmákat lementeni, beolvasni, így más sakk programokkal kommunikálni, képes lépésajánlatot tenni. A sakk egy kétszemélyes táblás játék. Kétszemélyes, tehát vagy mindkét személyt mi irányítjuk, vagy az egyiket vagy mindkettőt a gép. Ehhez nagyon nagy segítséget nyújtanak a mesterséges intelligenciában a lépésajánló módszerek. Mivel „Mesterséges Intelligencia 1." tárgyban tanultam ezeket a módszereket, és akkor ezeket Java nyelven valósítottam meg, így ezeket alkalmazni nem jelentett gondot. A program, mivel csak szabályos lépést enged megtenni, és lehetőségünk van egy nem túl erős ellenféllel (számítógép) szemben játszani, így a kezdő, sakkot nem ismerő játékosnak segít megtanulni a sakkot. A gép és ember közötti kommunikációt Javában a legegyszerűbb az indításkor megadni paramétereken keresztül, de mivel itt interaktívan, a játék közben is kell üzeneteket váltani, így grafikus felületen keresztül lehet kommunikálnunk a programmal. A tábla és bábuk megjelenítése is grafikusan történik. Ezt a java.awt és java.swing csomagbeli osztályok segítségével valósítom meg. Az egyes játszmák lementése fájlba és a beolvasása fájlból történik, így I/O műveleteket is kell kezelni, ezt java.io csomaggal végzem. Ezenkívül elengedhetetlen a java.util csomag néhány osztályának használata.

Látszik, hogy egy játék megvalósításához a Java nyelv több részét is használni kell, és mivel ez a sakk egy bonyolult játék, így nem csak a megjelenítésre, kommunikációra, I/O műveletekre kell nagy hangsúlyt fektetni, hanem magára a játékra, a játékszabályok korrekt és precíz megvalósítására is. Ezért találtam jónak a sakkot mint játékprogramot megvalósítani Java nyelven a diplomamunka keretein belül. A dolgozatot két részre osztottam: az első fejezet a programhoz egy felhasználói leírás, míg a második fejezet a továbbfejlesztéshez egy fejlesztői leírást tartalmaz.

1. FEJEZET

## Sakk program

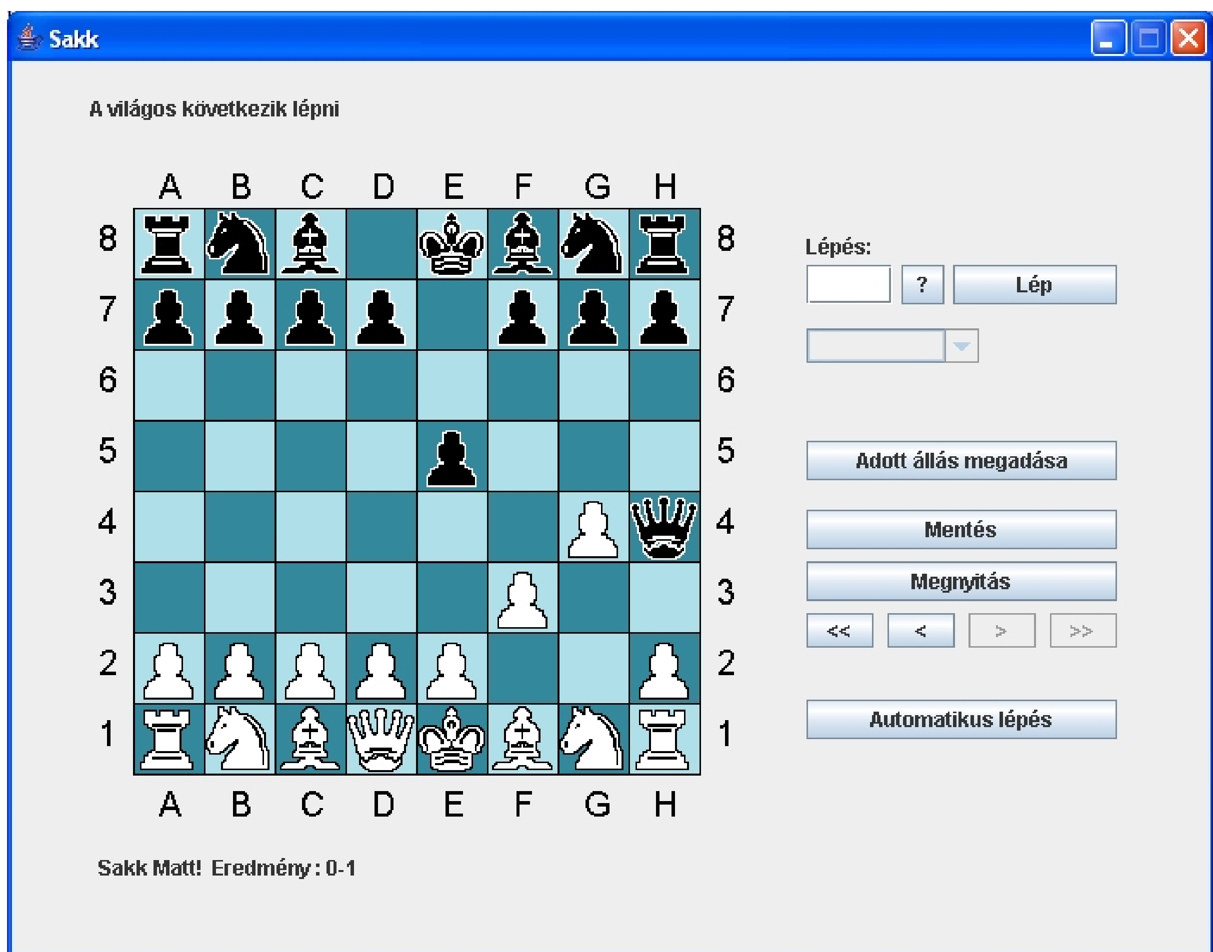
Mivel a program egy sakkjáték, ezért a sakkot mint játékot ismertetem[1]. A sakkot egy 8 x 8 táblán játszák, a táblán a mezők kétféle színűek lehetnek (világos, sötét), ezek felváltva vannak. A tábla oszlopai az angol ABC első 8 nagy betűjével (A-H) vannak megjelölve, a sorai 1-8-ig vannak számozva. A játékot ketten játszák, az egyik fehér (világos) bábukkal, a másik fekete (sötét) bábukkal rendelkezik. 6 fajta bábu létezik: király, királynő, bástya, futó, ló, paraszt. Egy játékos egy királlyal, egy királynővel, két bástyával, két futóval, két lóval és nyolc paraszttal rendelkezik. A játék elején ezek helye a táblán adott. A táblát úgy kell elhelyezni, hogy a fehérrel játszó játékosnak a jobb keze felőli sarkon található mező világos legyen. A játékot mindig a fehér kezdi.




*kezdő állás*

[1] http://www.jatek.hu/szabalyok/hlp_sakk.html

A játékban a felek egymás után következnek lépni. Egyetlen bábuval sem léphetünk le a tábláról, nem léphetünk olyan mezőre, melyen azonos színű másik bábu található, ha ellentétes színű bábu áll ott, akkor leütheti (kivéve a királyt), az odalépéssel egyidejűleg az ellenfél bábuját az adott mezőről eltávolítja. Egy bábu támad egy mezőt, ha oda szabályosan léphet, ha a támadott mezőn az ellenfél királya van, akkor sakkot adtunk. A játék célja, hogy az ellenfél királyát olyan helyzetbe hozzuk, hogy legalább egy bábuval támadva legyen a király (sakk), és ezt ne tudja hárítani, ekkor az ellenfél mattot kapott, és mi megnyertük a játékot. Ha nincs támadva az ellenfél királya, de csak olyan „szabályos” lépést tudna tenni az ellenfél, melyben a király támadásba kerül, ekkor pattal záródik a játék, és döntetlen az eredmény. Négyféle eredménye lehet a játéknak: fehér ad mattot (1-0), fekete ad mattot (0-1), patt (1/2-1/2), és közös megegyezéssel abbahagyják a játékot (*).

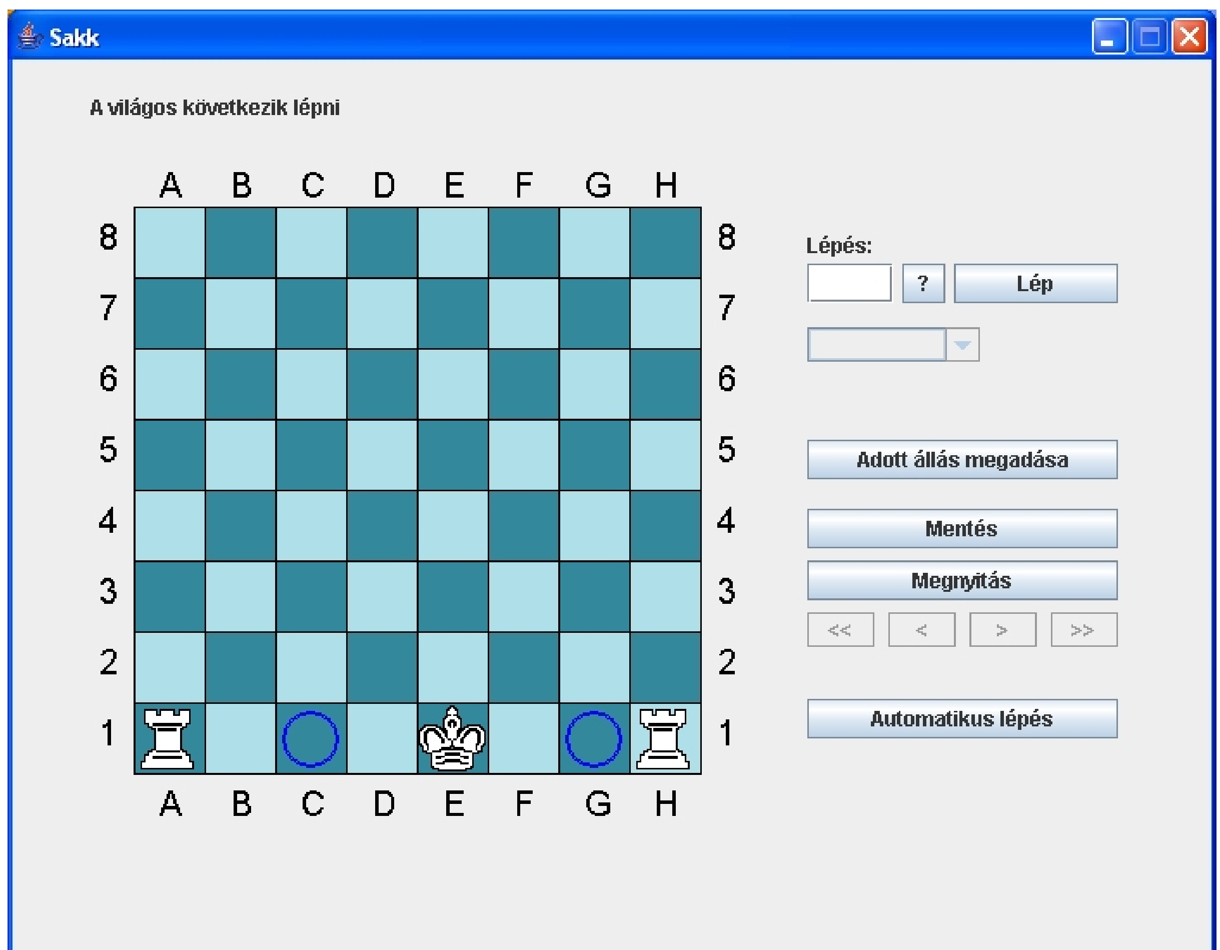



*sötét nyert*

## Bábuk

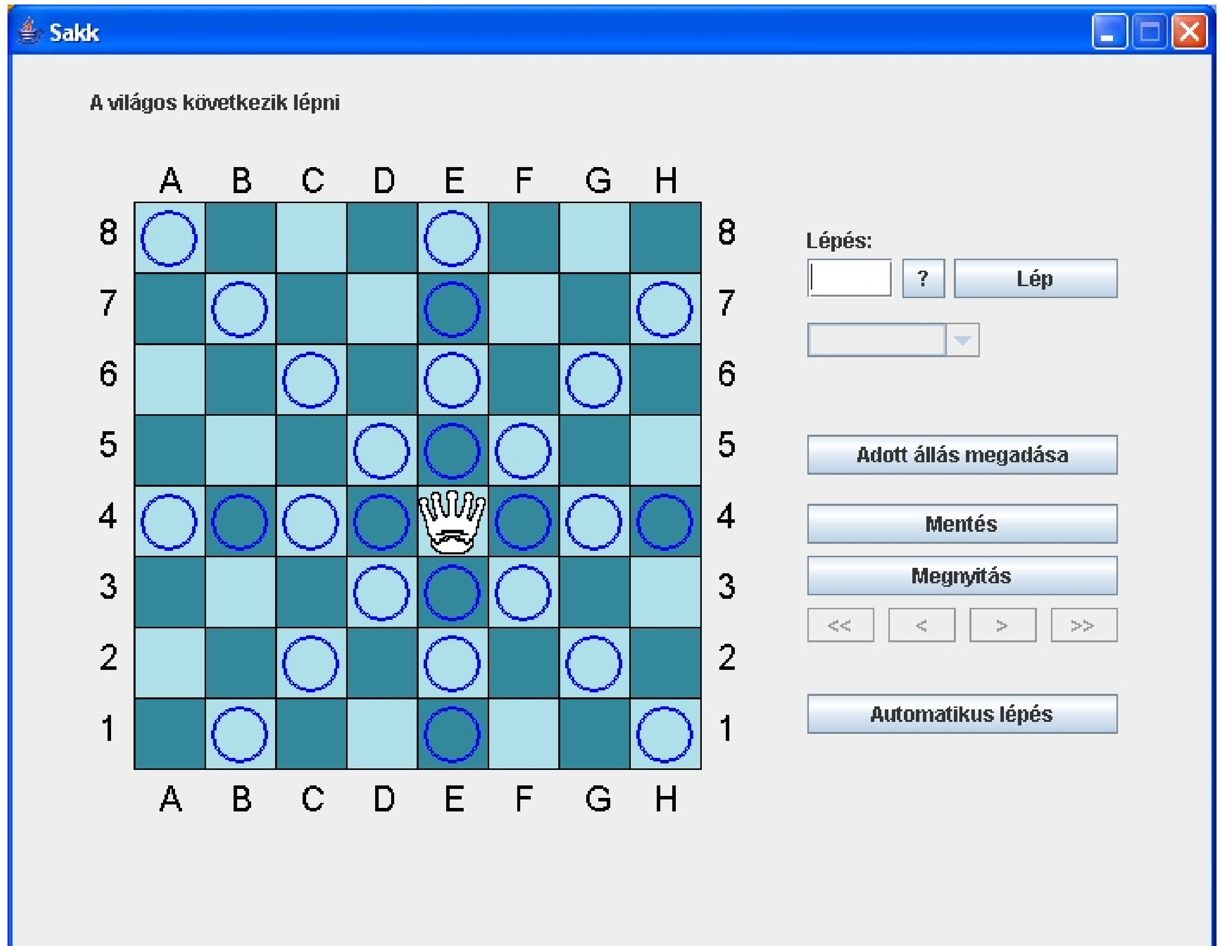
A bábuk részletes ismertetése:

- Király: Angolul king. Kétfajta szabályos lépést tehet: léphet bármely szomszédos mezőjére, ha azt nem támadja az ellenfél, illetve azon megegyező színű bábu nem található, a másik fajta lépés a sáncolás. A sáncolás egy különleges lépés, a király vízszintesen két mezőt oldalra lép, az átlépett mezőre pedig a lépés irányának megfelelő sarokban található bástya kerül. Ennek a lépésnek a feltétele, hogy a király még nem lépett, a bástya sem, a király és a bástya közötti mezők üresek, valamint ahova lép és az „átlépett" mező az ellenfél által nincs támadva. A sáncolásnak két fajtája létezik, rövidsánc: amikor a király a hozzá közelebbi bástya irányában sáncol, és a hosszúsánc: a távolabbi bástya irányában történik a sáncolás. Nem lehet olyan lépést tenni, mellyel a király magát ütés alá helyezné. Lépésmegadásnál a király rövidítése: K.



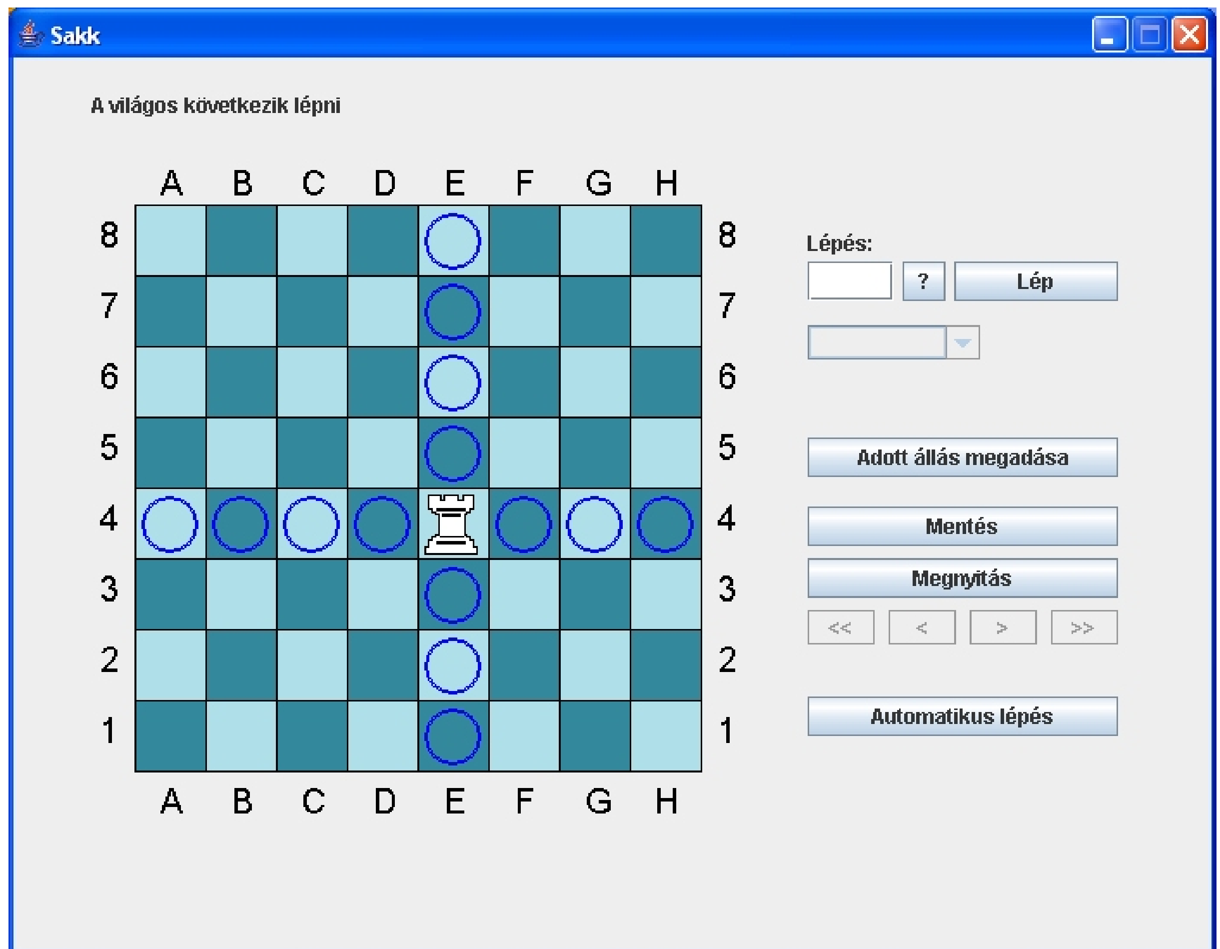

*sáncolás*

- Királynő (vezér): Angolul queen. A bábuk közül a legértékesebb. Vízszintesen, függőlegesen, átlósan bármennyit léphet bármelyik irányban, ha a mező, ahová lép, nem tartalmaz azonos színű másik bábut, és az átlépett mezők üresek. Lépésmegadásnál a rövidítése: Q.



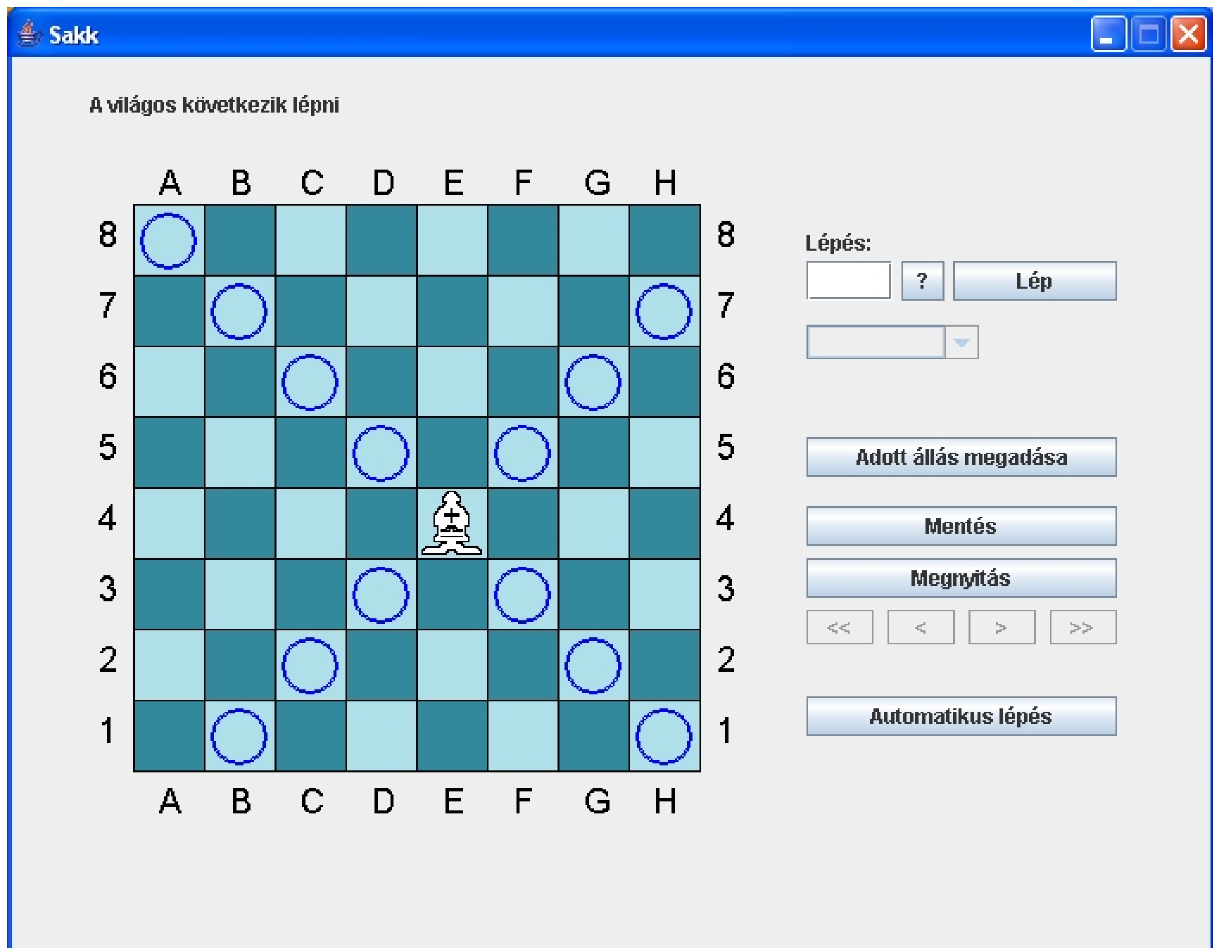

*királynő lehetséges lépései*

- Bástya: Angolul rook. Vízszintesen, függőlegesen léphet bármennyit bármelyik irányban, ha az átlépett mezők üresek, és a célmezőn megegyező színű bábu nincs. Lépésmegadásnál a rövidítése: R.



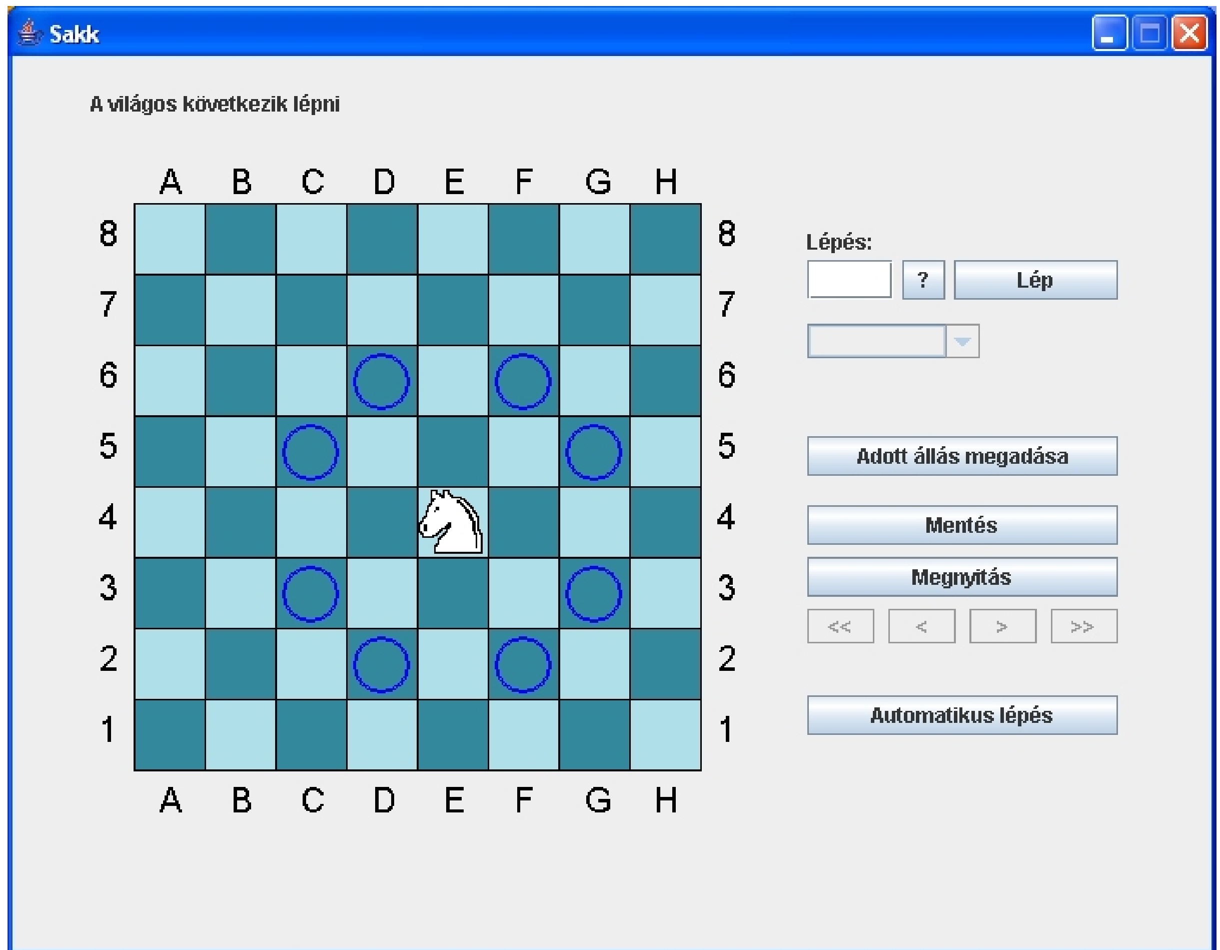

*bástya lehetséges lépései*

- Futó: Angolul bishop. Bármelyik átlós irányban léphet bármennyit, ha az átlépett mezők üresek, és a célmező saját bábuval nincs elfoglalva. Lépésmegadásnál a rövidítése: B.




*futó lehetséges lépései*

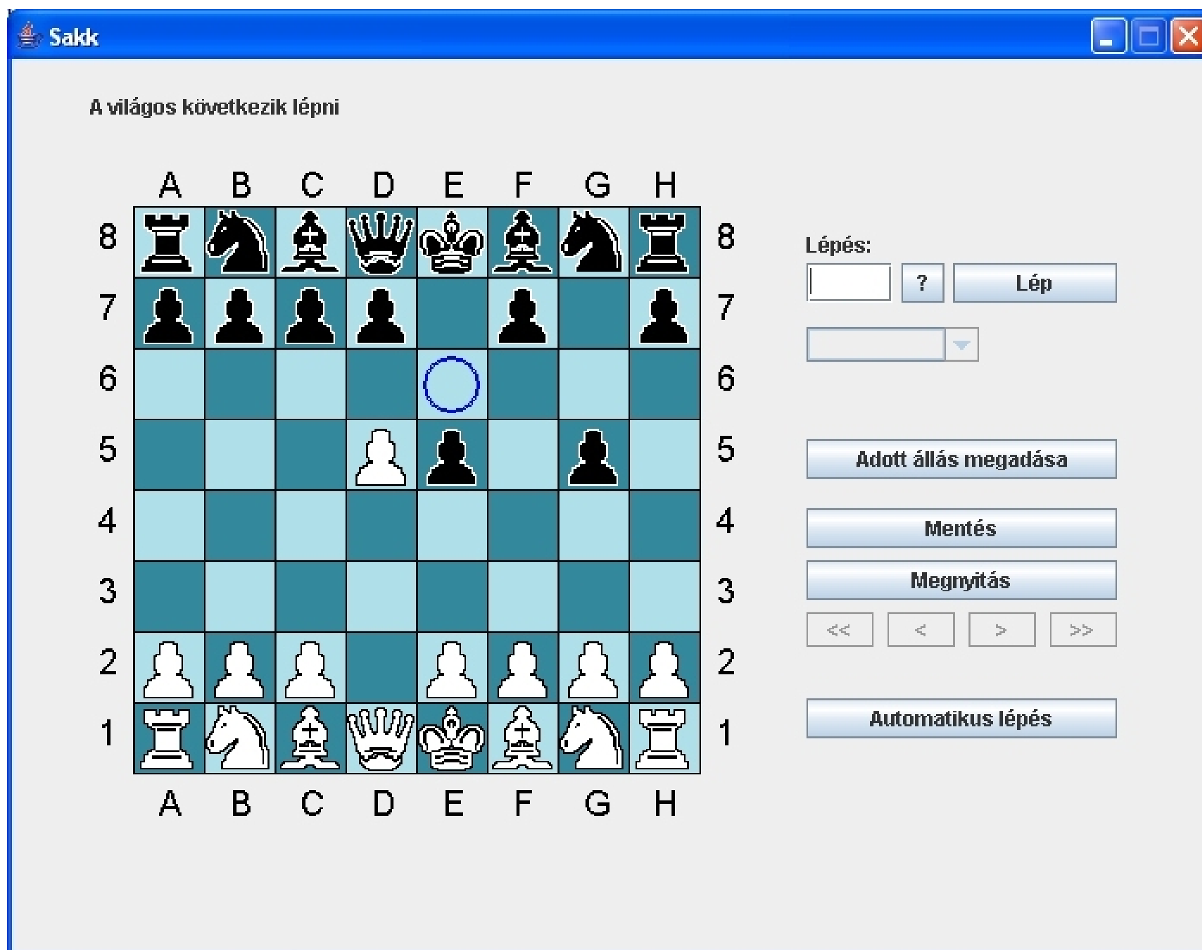
- Ló: Angolul knight. Azokra a legközelebbi mezőkre léphet, amelyek nem szerepelnek az átlójában, az adott sorában és adott oszlopában. Egy másik fajta megfogalmazásban a lépés leírható úgy is, hogy a vízszintes és függőleges irányok közül az egyik irányban kettőt lép, a másik irányban egyet, egy lépésben. A célmezőn megegyező színű bábu nem lehet, és az átlépett mezőkön állhat bábu. Lépésmegadásnál a rövidítése: N.




*ló lehetséges lépései*

- Paraszt (gyalog): Meg kell különböztetni a lépést és az ütést. A paraszt az ellenfél irányában függőlegesen előre léphet egyet, első saját lépésében léphet kettőt, ha a célmező és az átlépett mező üresek. A gyalog által üthető mezők az átlósan közvetlen előtte álló mezők, ha ott az ellenfél bábuja található, illetve van egy másik fajta ütése a gyalognak: a menetközbeni ütés (en passant). Ha a gyalog által támadott mezőt az ellenfél gyalogja átlépi, akkor ütheti ezt a gyalogot úgy, mintha az csak egyet lépett volna. Ha a gyalog ütéssel vagy lépéssel eléri az utolsó sort, akkor a lépés részeként át

kell változnia azonos színű másik bábuvá, ez a bábu lehet királynő, bástya, futó vagy ló. Ezt gyalogátváltozásnak hívjuk, és azonnal érvénybe lép az új bábu.



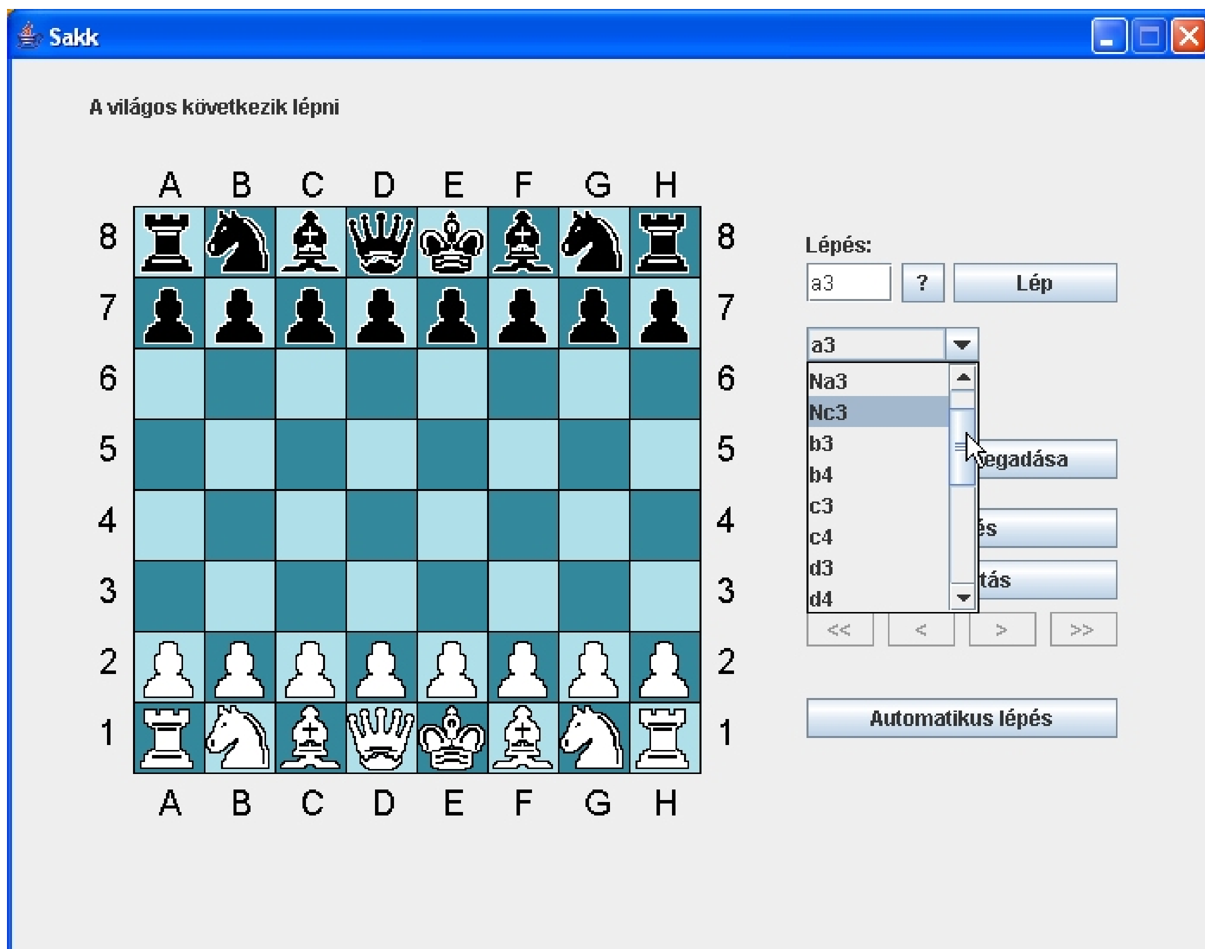

*menetközbeni ütés (en passant)*

## Lépés

Állásból állásba lépéssel jutunk. Minden lépésnek van rövidítése[2], ezt PGN lépésformátumnak nevezzük. A rövidítés a paraszt kivételével az adott bábu nevének rövidítésével kezdődik (K, Q, R, B, N). Ezután ha ahová lépni akarunk, oda megegyező színű és típusú bábu is léphetne, akkor az a sor (1-8) vagy oszlop (a-h) jelét kell írni, amely egyértelműen megkülönbözteti a két bábut. Ha ütés történik, akkor ezt „x" jelzi, ezután pedig a célmező oszlop (a-h) és sor (1-8) azonosítója kerül. Ha paraszttal lépünk, akkor csak a célmezőt kell írni, ha ütünk, akkor azt az oszlopot, melyből indultunk, majd „x", és végül a célmező azonosítója áll. Ha gyalogátváltozás történik, akkor a lépésrövidítés végére az átváltozott bábu rövidítése is odakerül. Amennyiben az adott lépésben sakkot adunk, + jel

[2] http://hu.wikipedia.org/wiki/A_sakkjatszmak_lejegyzese

kerül a lépésrövidítés végére, matt esetén #. Rövidsánc jele O-O, hosszúsánc jele O-O-O. A lépésrövidítés lehet, hogy szintaktikailag helyes, de lehet, hogy szemantikailag nem, lehet, hogy ugyanannak a lépésnek eltérő rövidítése van más-más állásokban. Egy lépés szabályos, ha annak a játékosnak a bábujával lépünk, mely lépni következik, és a lépés az adott bábunak megengedett. Lépésrövidítés példák: e4, Qxc4+, Ndxf4. A lépést megadhatjuk úgy, hogy a táblán két helyre kattintunk, először arra a mezőre, ahonnan lépni szeretnénk, ha ez mező nem üres, és annak a játékosnak a bábuja található rajta, amelyik lépni következik, akkor a mező körül egy piros négyzet jelenik meg, ezzel kijelölve az adott bábut. Ezután oda , ahová lépni kívánunk. Ha megengedett ez a lépés, akkor a bábu odalép, ha nem, akkor a lépést szabálytalannak tekinti, ezt a „Szabálytalan lépés" hibaüzenettel jelzi, és léphetünk újra. (A sakk szabályai között szerepel egy olyan kitétel is, hogy „fogott bábu lép", vagyis ha egy bábut megfogunk, és annak létezik szabályos lépése vagy lépései, akkor ezzel a bábuval muszáj lépni. A sakk eme szabályát a program nem vizsgálja.) Egy szabályos lépés után az ellenfél következik lépni kötelezően. Lépést megadhatunk a lépésrövidítés megadásával is. A programban ennek megadására egy szövegmező szerepel, itt kell megadni a lépésrövidítést. A program nemcsak a szigorú értelemben vett lépésrövidítést fogadja el. Ütésnél nem kötelező az x-et, a sakk vagy matt jelzésére szolgáló + , # jelet kiírni, gyalogátváltozásnál a bábu jelét, amire átváltozunk, megadhatjuk, de a program figyelmen kívül hagyja, és felajánlja a választást! Amint megadtuk a lépést, a LÉP gombra kattintva először szintaktikai, majd szemantikai ellenőrzést hajt végre a program. Amennyiben szintaktikailag nem szabályos a lépés, akkor „Érvénytelen lépésmegadás" hibaüzenetet kapunk, ha szintaktikailag megfelelő, de szemantikailag nem (pl. sakkba lépnénk), a „Szabálytalan lépésmegadás" hibaüzenettel jelzi ezt számunkra a program. Lehetőségünk van a szabályos lépések közül választani, erre a szövegmező és a LÉP gomb között található kérdőjel feliratú gombra kattintva nyílik lehetőségünk. Ekkor egy legördülő menüből választhatunk, a lépésrövidítésre kattintva a szövegmezőben ez megjelenik, melyet a LÉP gombra való kattintással fogadhatunk el.




*lépésajánlat*

## Adott állás megadása

Nemcsak kezdőállásból indulhat a játék, lehetőségünk van a 16-16 bábu táblán való tetszőleges elhelyezésére, valamint annak megadására, hogy az adott állásban ki következik lépni. Erre szolgál az „Adott állás megadása" feliratú gomb, melyre kattintva egy újabb ablak nyílik meg a tetszőleges állásmegadáshoz (ameddig ezt az ablakot be nem zártunk, az eredeti ablak inaktív marad). Itt rádiógombok segítségével lehetőségünk van kiválasztani, hogy világos vagy sötét játékos következzen lépni, illetve a 32 bábu képe látható, mindegyik mellett egy szövegmező, ahová megadhatjuk annak a mezőnek a nevét, melyen az adott bábut látni szeretnénk. A mezőnév megadásánál az oszlop betűjele (tetszőlegesen lehet kis- vagy nagybetű) és a sor sorszáma kell, hogy szerepeljen. Ha egy szöveges mezőt üresen hagyunk, az adott bábu nem fog szerepelni a táblán. Felmerülhet a kérdés, hogyha egy olyan játékállást szeretnénk megadni, amelyben például két világos királynő szerepel, azt hogy kell. Mivel a játék folyamán kialakulhat ilyen állás gyalogátváltozás révén, így úgy találtam célszerűnek,

hogy ezt a paraszt bábujánál lehessen megadni. A gyalog pozíciójának megadása után a szövegmezőbe kell beírni annak a bábunak a rövidítését (Q, R, N, B), amelyre a parasztot „cseréljük”. Pl.: a2Q. Az ablak megnyílásakor minden szöveges mező üres, és a világos játékos következik lépni. Lehetőségünk van a táblán jelenleg szereplő állás importálására a „Jelenlegi állás” feliratú gombbal. Ekkor a program automatikusan kitölti a szöveges mezőket, de így is lehetőségünk van módosítani rajta. Kérhetjük a kezdő állást is az „Alapállás” gombra kattintva, ekkor a kiinduló állásnak megfelelően tölti ki a szöveges mezőket a program, ha akarjuk ezen is módosíthatunk. A „Mégse” gombbal léphetünk vissza az állásmegadástól. A „Rendben” gombbal fogadtathatjuk el az állásmegadást, ekkor a gép ellenőrzi a szöveges mezőket. Amennyiben szabálytalan pozíciómegadás szerepelne, vagy egy mező nevét több helyre is megadtuk, „Hibás helyzetmegadás” hibaüzenettel jelzi ezt nekünk a program, és a kurzort az első hibás szövegmezőbe helyezi.

*adott állás megadása*

## PGN

A Portable Game Notation[3] (hordozható játék leírás) fájltípus szolgál a sakkjátszmák leírására, sakkprogramok közti adatcserére.A PGN állomány egy ASCII szöveges fájl, mely struktúrálva van a könnyebb írás-olvasás érdekében. A PGN fájl alapvetően két részből áll: STR (Seven Tag Roster) és a SAN (Standard Algebraic Notation). A STR hét címke sorrendi jegyzéke, ezek a cimkék a játék jellemzői. A SAN szabványos algebrai leírás, a lépések leírására szolgáló rész. A program is a PGN fájltípust használja, ilyen fájltípust tud megnyitni, illetve ilyen fájltípusba ment.

```
[Event "?"]
[Site "corr m (FS"]
[Date "????.??.??"]
[Round "?"]
[White "Wittmann W"]
[Black "Steinbrueck X"]
[Result "1-0"]
[ECO "A57/19"]

1.d4 Nf6 2.c4 c5 3.d5 b5 4.cxb5 a6
5.e3 g6 6.Nc3 Bg7 7.a4 O-O 8.Nf3 d6
9.Ra3 Bb7 10.Be2 Nbd7 11.O-O axb5 12.Bxb5 Ba6
13.Re1 Ra7 14.Bc6 Nb8 15.Nb5 Nxc6 16.dxc6 Ra8
17.e4 Ng4 18.h3 Ne5 19.Nxe5  1-0
```

*Egy PGN fájl*

## Mentés

A sakkjátszma bármely szakaszában lementhető az addigi játék PGN fájlba. Mentés gombra kattintva először egy ablak nyílik, ameddig ez nyitva van, addig a „fő” ablak inaktívvá válik. Ebben az ablakban a PGN fájlstruktúrának megfelelően néhány adat állítható be. Ezek az

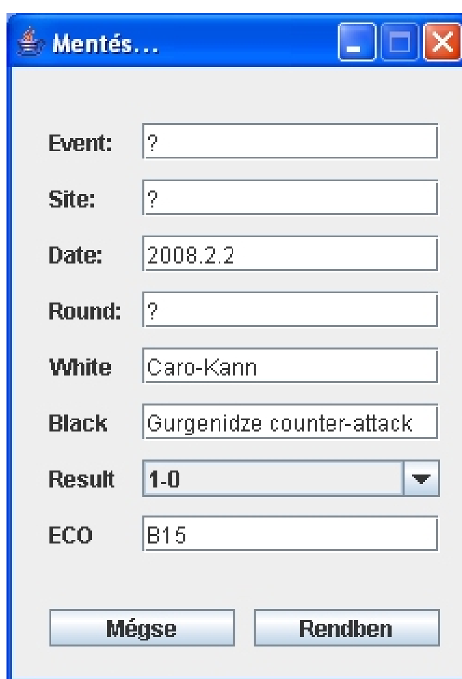
[3] http://geocities.com/CapeCanaveral/Launchpad/2640/pgn/pgn_spec/pgn_lidx.htm

adatok a strukturált PGN fájl első része és még egy adat: Event, Site, Date, Round, White, Black, Result, ECO.

- Az Event mező szolgál az esemény megnevezésére, mely meglehetősen jól leíró legyen. A következetes Event mező kitöltése segítség lehet adatbázisokban való kereséskor. Ha nem tudjuk az esemény nevét, akkor kérdőjelet kell beírni az Event mezőbe. Pl.: Moscow City Championship.
- A Site mezőben adhatjuk meg a város és a vidék nevét. A vidék nevének megadásához használjuk az IOC (International Olympic Committee) három betűs rövidítéseit, ha rendelkezésre állnak. Kérdőjelet kell megadni, ha nem kívánjuk kitölteni ezt a mezőt. Pl.: St. Petersburg RUS.
- A Date mezőben kell megadni a dátumot, amikor a játék elkezdődött, ez nem feltétlen egyezik meg az esemény dátumával. A dátumot ÉÉÉÉ.HH.NN ahol az ÉÉÉÉ az évet, HH a hónapot, NN a napot jelöli, és ezeket ponttal kell elválasztani. A gép automatikusan kitölti ezt a mezőt, de a változtatás lehetőségét meghagyja. Amennyiben nem ismert a dátum, úgy a nem ismert részekre kérdőjeleket kell rakni. Pl.: 1992.08.??
- Round mező szolgál a játékon belüli játszmaforduló jelölésére. Ha ismeretlen, kérdőjelet kell írni a mezőbe.
- White mezőben szerepel a világos bábukkal versenyző játékos vagy játékosok nevei. A vezetéknévvel kell kezdeni, utána ha rendelkezésre áll a utónév, akkor azt szóközzel elválasztva kell a vezetéknév után megadni. Amennyiben több versenyző is játszott a világos bábukkal, úgy a nevek közé kettőspontot kell tenni. Ha a játékos a számítógépprogram, akkor a program nevét kell megadni, lehetőség szerint verziószámmal. Ha nem ismert a játékos illetve a neve, akkor ez kérdőjel megadásával jelezhető. Pl.: Benko Pal.
- Black mezőben a sötét bábukkal játszó játékost vagy játékosokat kell megadni, a White mezőhöz hasonlóan.
- Result mezőben a játék eredménye szerepel. Mivel csak négy fajta végeredménye lehet egy sakkjátszmának, így ezek egy legördülő menüből választhatók ki. A lehetséges végeredmények: 1-0 világos nyert, 0-1 sötét nyert, 1/2-1/2 döntetlen, * a játék félbeszakadt vagy ismeretlen az eredmény. Ha a játék során valamelyik játékos mattot ad, vagy patt lesz a végeredmény, és így mentjük a játékot, akkor nincs

lehetőségünk a végeredmények közül választani, a program automatikusan kitölti ezt a mezőt. A nagy sakkjátékosok általában nem játszák végig a játékot, hanem hamarabb feladják, megnyerik, ezért matt nélkül is megadható a játszma végeredményeként 1-0, 0-1.

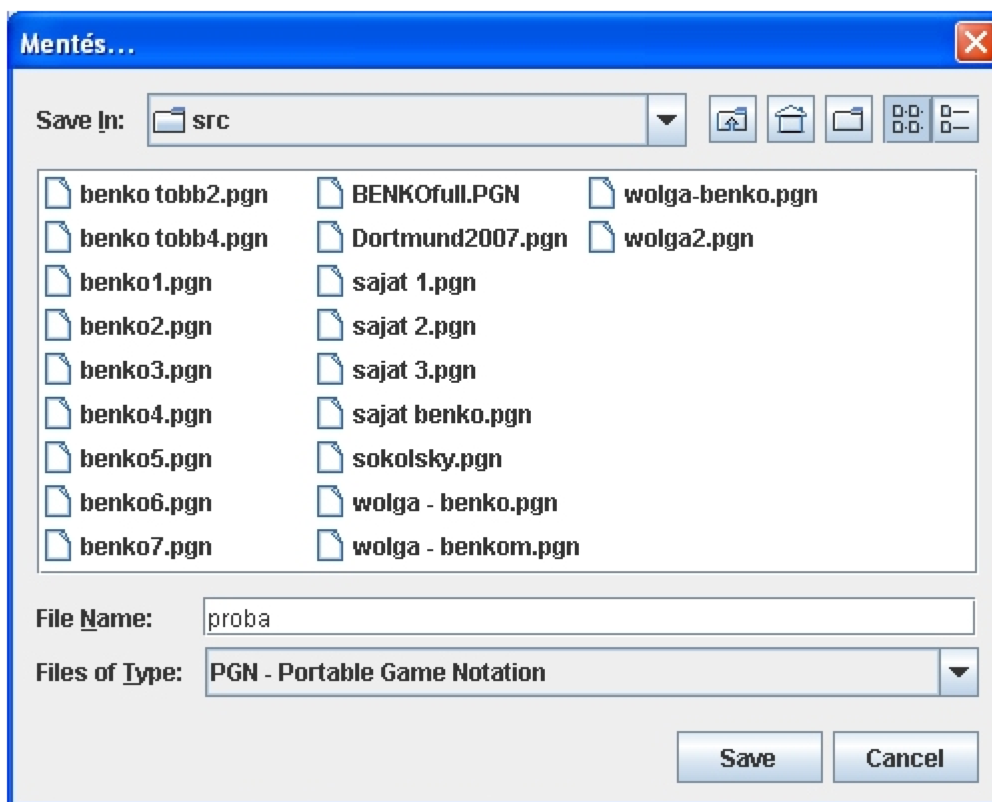
- ECO mezőben játéknyitás leírást lehet megadni[4]. Az ECO a Encyclopedia of Chess Openings szavak rövidítése, egy rögzített lista, melyben a játszmák első pár lépéseihez van azonosító rendelve. Az azonosító egy három betűs szó, melynek szerkezete úgy néz ki, hogy először egy betű van (A-E), utána két számjegy (0-9). A program ezt az ECO mezőt automatikusan kitölti, valamint a White, Black mezőbe beírja, hogy kiknek a nevéhez fűződik ez a játszmanyitás. Pl.: B15.




*mentés*

Miután kitöltöttük a mezőket, és a Rendben gombra kattintottunk, megjelenik egy másik ablak, melyben kiválaszthatjuk a fájl nevét és elérési útvonalát. A fájl típusa PGN típusra van állítva, és pgn kiterjesztéssel lesz a fájlnév ellátva.

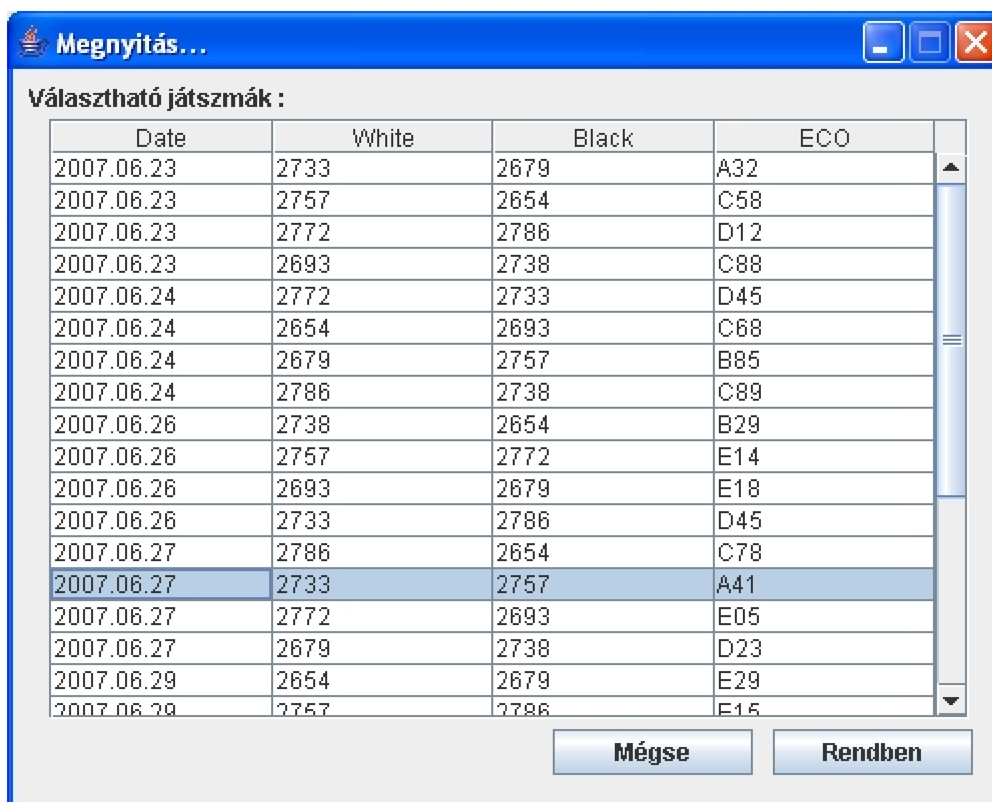
[4] http://observer.homestead.com/openings.html




*mentés - fájlválasztó*

## Megnyitás

A program képes PGN fájlokban tárolt játékok megnyitására, betöltésére. A PGN fájlokat alapvetően kétféle csoportba oszthatjuk: egy vagy több játszmát tartalmazó PGN fájl. Ezeket nem tudjuk megkülönböztetni „ránézésre”. A játék bármely részén lehet játékot megnyitni, ilyenkor az eddigi játékunk elvész. Amikor a Megnyitás gombra kattintunk, megjelenik a mentésnél már látott fájlkiválasztó ablak. Itt alapesetben csak pgn kiterjesztésű PGN fájlokat tudunk megnyitni. Megnyitás után, ha egy játszmát tartalmazó PGN állomány volt, akkor megnyitja a program. Ha több játékot is tartalmaz a PGN fájl, akkor megjelenik egy ablakban egy táblázat, melynek négy oszlopa és annyi sora van, ahány játékot tartalmaz a fájl. Ezek az oszlopok a PGN fájl címkejegyzékéből valók, ezek szinte egyedileg azonosítanak minden egyes játszmát. Az oszlopok: Date (a játék dátuma), White (világos játékos neve), Black (fekete játékos neve), ECO (játéknyitás osztály). A táblázatból kiválasztjuk azt a játszmát, melyet meg szeretnénk nyitni, ezt megtehetjük úgy, hogy kiválasztás után a Rendben gombra kattintunk, vagy az adott játszmát azonosító soron kettőt kattintunk.



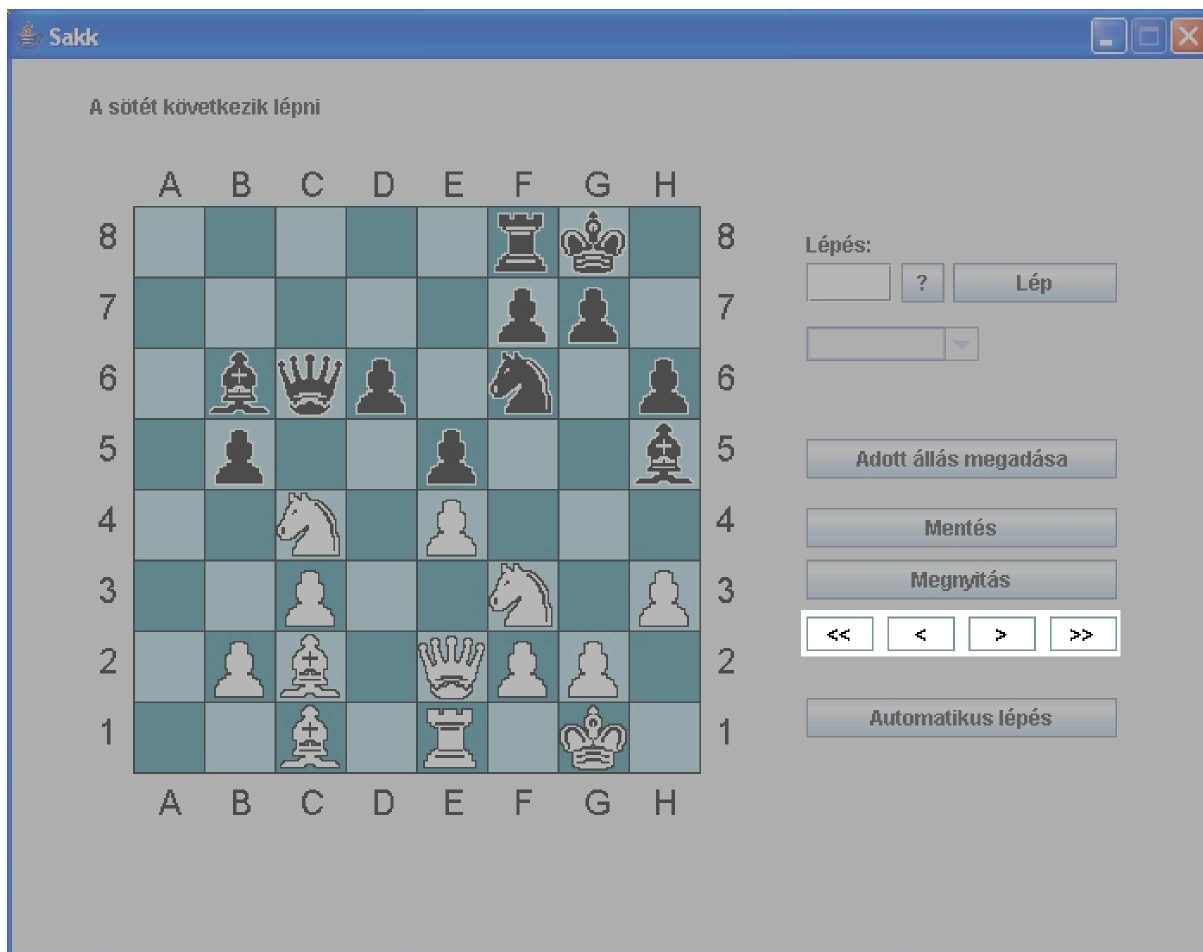

| Date | White | Black | ECO |
|---|---|---|---|
| 2007.06.23 | 2733 | 2679 | A32 |
| 2007.06.23 | 2757 | 2654 | C58 |
| 2007.06.23 | 2772 | 2786 | D12 |
| 2007.06.23 | 2693 | 2738 | C88 |
| 2007.06.24 | 2772 | 2733 | D45 |
| 2007.06.24 | 2654 | 2693 | C68 |
| 2007.06.24 | 2679 | 2757 | B85 |
| 2007.06.24 | 2786 | 2738 | C89 |
| 2007.06.26 | 2738 | 2654 | B29 |
| 2007.06.26 | 2757 | 2772 | E14 |
| 2007.06.26 | 2693 | 2679 | E18 |
| 2007.06.26 | 2733 | 2786 | D45 |
| 2007.06.27 | 2786 | 2654 | C78 |
| 2007.06.27 | 2733 | 2757 | A41 |
| 2007.06.27 | 2772 | 2693 | E05 |
| 2007.06.27 | 2679 | 2738 | D23 |
| 2007.06.29 | 2654 | 2679 | E29 |

*megnyitás (több játszmát tartalmazó PGN fájl)*

Felmerülhet a kérdés, hogy mi történik hibás PGN állomány esetén. Amennyiben a PGN fájl első részében a hét címke sorrendi jegyzékében van a hiba, úgy az a címke nem töltődik ki. Ha a játékleírás-részben van szintaktikai vagy szemantikai hiba, ezt a program úgy kezeli, mintha csak eddig tartana a játszma, vagyis a hibás lejegyzést és a fájlban utána következő részt figyelmen kívül hagyja.

## Navigáló gombok

A Megnyitás gomb alatt található négy navigáló gomb. Ezek lehetnek aktívak illetve inaktívak.




*mozgató gombok*

A játék során a lépések sorozata felfogható egy kétirányban láncolt lista elemeiként. Ezen elemek közötti navigálásra szolgálnak a gombok. Játék megnyitásnál ezen gombok segítségével „nézhetjük” végig a játszmát. Ezek révén nyílik lehetőségünk a játék során visszalépni egy vagy több lépést. Ez kicsit ellentmond a sakk szabályai közül a „fogott bábu lép, letett bábu marad” elvnek. Sakkban eredetileg nincs visszalépés, így ez az elv tiszteletben tartása ránk van bízva. A négy gomb jelentése balról jobbra: az első (<<) a láncolt lista legelejére lép, a második (<) a láncolt listában egyet visszalép, a harmadik (>) a láncolt listában egyet előre lép, a negyedik (>>) a láncolt lista legvégére lép. A „<<” gomb akkor aktív, ha nem a láncolt lista legelső elemére mutat a képzeletbeli mutatónk, vagyis van hová visszalépnünk. A gomb megnyomásával egyidejűleg ez a gomb és a „<” gomb inaktívvá válik, a „>”, „>>” gombok pedig aktívvá. Ezzel a gombbal tudunk a játék „legelejére” lépni. A „<” gombot megnyomni akkor van lehetőségünk (aktív), ha a láncolt listában a mutatónk által jelzett elem előtt található még elem. A gomb megnyomásakor a „>” és a „>>” gomb

aktív lesz. Amennyiben a gomb megnyomása után a mutatónk a láncolt lista legelső elemére mutat, ez a gomb és a „<<” gomb inaktív lesz. Ezzel a gombbal tudunk egyet visszalépni a játékban. A „>” gombbal tudunk egyet előre lépni a játékban, ha aktív a gomb. Ez a gomb akkor aktív, ha a mutató nem a láncolt lista legutolsó elemére mutat. A „>” gomb megnyomásakor a „<<” és a „<” gomb aktív lesz. Ha a gomb megnyomása után a lista legutolsó elemére mutat a mutató, akkor ez a gomb és a „>>” gombok inaktívvá válnak. A „>>” gomb a lista legvégére való ugrásra szolgál. Akkor aktív, ha nem a láncolt lista legvégén vagyunk, a gomb megnyomásakor a „<<”, „<” aktívvá válnak, a „>”, „>>” gombok pedig inaktívvá. Mostmár tudjuk, értjük a gombok jelentését, szerepük főleg az elmentett játékok betöltése után van, ezen gombok segítségével „játszható újra” a játék. A „…letett bábu marad” elv betartása úgy oldható meg, ha a játékban visszaléptünk egy vagy több lépést, utána előrelépünk addig, ahol a játékot abbahagytuk. Kezdő játékosoknak nagy segítség lehet, hogy vissza tudja nézni, hogy hogyan, milyen lépések során jutott el az adott állásig, egyes lépéseinek milyen következményei voltak.

## Automatikus lépés

Ez a gomb a navigáló gombok alatt található. A programmal játszhat két ember, ember gép ellen és gép gép ellen. Ennek a gombnak a segítségével tud a program a gép nevében lépést tenni. A gomb megnyomásakor a program tesz egy általa jónak ítélt lépést. Ez a program bármely részén megtehető, a játék bármely állásában. Tehát ha a gép ellen szeretnénk játszani, akkor miután tettünk egy lépést, erre a gombra kattintva a gép is lép egyet, ami után szintén mi következünk. Kérhetjük, hogy mi helyettünk is lépjen, amennyiben ezt rossz lépésnek ítélnénk, a navigáló gombok segítségével lehetőségünk van ezt a lépést visszavonni, és másik lépést tenni. Ha a játék során matthoz jutunk, akkor a gomb megnyomásakor nem tesz a lépést a gép, mivel matt esetén nincs szabályos lépés. Az algoritmus, ami alapján a program lépést ajánl, később lesz részletesen ismertetve.

Végigértünk a program bemutatásán, láttuk, hogyan használható a program, megismertük az egyes gombok jelentését, a hibaüzeneteket kiváltó eseményeket, és hogy mit jeleznek ezek az üzenetek.

# 2. FEJEZET

## Konvertálás PGN formátumra és formátumról

A program írásakor, és különböző programozási szempontokból célszerűbbnek láttam egy lépést nem a PGN formátummal leírni, hanem hogy honnan hova lépünk. Tehát négy koordinátával leírható egy lépés, honnan x, y koordináta, hová x és y koordináta, ezek a programban `nx,ny,ax,ay` változókkal vannak jelölve („n” az a honnan, az „a” a hová utolsó betűjéből származik, így könnyen megfejthető a rövidítés). Ezzel a lépésleírással az a probléma, hogy mivel program a felhasználóval PGN formátummal kommunikál, így meg kell valósítani a konverziót oda és vissza is. A négy koordináta számtípusú(0-7), az oszlop- és sorkoordinátát is át kell konvertálni, mert egyrészt az oszlopkoordináta a PGN formátumban betűkkel van jelölve (A-H), a sorkoordináta pedig 1-8 ig, ezeknek a konverziója nagyon egyszerű. Mivel már leírtam az előző fejezetben, hogy hogyan is épül fel a PGN lépés formátum, így erre most nem térek ki.

A lépés-koordinátákról a PGN formátumra konvertálás talán az egyszerűbb feladat. Nem szabad elfelejtkezni arról, hogy egy adott lépés mindig egy adott álláshoz tartozik, vagyis ennek az állásnak az ismerete is szükséges a konverzióhoz. Tehát `nx,ny,ax,ay` koordinátákból kell PGN formátumot készíteni. Ezt az `Operator` osztály `alakit` metódusa végzi, melynek a paramétere az állás, és egy sztringet ad visszatérési értékként. Mivel ez a sztring több részből áll, ezért egy `StringBuffert`[5] létrehozok, és ebbe „dobom bele” az információkat, és a végén kiolvasom. A függvény elöször lekéri az adott állásban az `nx,ny` koordinátán szereplő bábut. Mivel csak szabályos lépésekkel foglalkozunk, így itt a hatféle bábu lehet, ennek megfelelően mindegyiknél mást-mást kell tennünk!

Ha paraszt található, és ütés történik ebben a lépésben, vagyis az `nx` és `ax` koordináta között a különbség egy, akkor le kell írnunk azt az oszlopkoordinátát, amiből idejutottunk, vagyis `nx`-et átkonvertálva, azután egy „x”-et teszünk, ezzel jelezve az ütést, majd pedig a hová koordinátái átkonvertálva kerülnek leírásra. Ha nem volt ütés, akkor csak a hová koordinátákat kell leírni. Mivel a parasztlépésnek van egy olyan speciális esete, hogy elér az ellenfél alapvonalához, ekkor átváltozik, ha ilyen lépésünk van, akkor a lépésleírás végére nagybetűvel oda kell írni az átváltozott bábu rövidítését (N,B,R,Q). Pl.: e4, dxc5, h8Q, fxg8B

[5] http://java.sun.com/j2se/1.5.0/docs/api/

Ha ló található itt, akkor a leírás egy N betűvel kezdődik, jelezve a bábu fajtáját! Ezután meg kell vizsgálnunk, hogy ahova lépett, oda vele megegyező színű másik lóbábu is tudott volna-e lépni, ha igen, akkor az egyértelműség miatt jelölni kell, hogy melyik bábu lépett, ezt a bábu oszlopkoordinátájának megadásával tesszük, ha ez is egyezne, akkor a sorkoordinátát adjuk meg. Ha egyértelmű a hová koordinátákból is, hogy melyik lóbábu lépett, akkor erről nem kell információt leírni. Ha hová koordinátákon a lépés előtt bábu található, vagyis ütés van, akkor ezt a PGN lépésformátumban egy „x" leírásával jelezzük. Ezután leírjuk az átkonvertált hová koordinátákat `ax`, majd `ay` sorrendben. Pl.: Ne4, Ndb5, Nxe6, N3xe4

Ha futóbábu van a honnan koordinátákon, akkor B betűvel kezdődik a lépésleírás, majd hasonlóan a lóhoz, itt is megvizsgáljuk, hogy egyértelmű-e a hová koordinátákból, hogy honnan melyik bábu lépett. Felmerülhet a kérdés, hogy ezt miért kell vizsgálni, hiszen két futóbábu található színenként a táblán, és ezek ellenkező alapszínű mezőkön tudnak mozogni, így biztos, hogy nem tudnak a „másik" mezőjére lépni. Ez igaz is lenne, ha két dolog nem lenne a programban, az adott állásmegadás és a paraszt átváltozás, ugyanis mindkettőnek lehet az a következménye, hogy két megegyező színű futó található a táblán, melyek azonos színű mezőkön mozognak. Tehát ezt a vizsgálatot el kell végezni, és többértelműség esetén a megfelelő információkat le kell írni a B betű után. Ha ütés történik a lépésben, akkor ez az x leírásával jelezhető, ezután pedig a hová koordinátákat kell átkonvertálva leírni. Pl: Bc2, Bce3, Bxd4, B6xe4

Ha bástyával lépünk, akkor a bástyabábunak megfelelő rövidítés leírásával kell kezdeni a lépésmegadást, ez a betű az R. Ezután megvizsgáljuk, hogy másik bástya is léphetett volna-e ugyanoda, ha igen, akkor a megfelelő bástya oszlop- vagy sorkoordináját le kell írni, ezzel azonosítva a bábut. Ezután ütés esetén x, majd a hová koordinátát írjuk le a konverzió után. Pl.: Rc1, Rae1, Rxa4, R1xb5

Ha királynővel lépünk, akkor Q betűvel indul a lépésleírás. Ezután hasonlóan a többi bábuhoz, az egyértelműség vizsgálata következik, majd az ütésvizsgálat, majd a hová információ leírása. Pl.: Qg4, Qdh5, Qxd2, Qexb4

Ha pedig királybábuval lépünk, akkor a bábu kezdőbetűjének leírása előtt meg kell vizsgálni, hogy nem sáncolás történik-e, ugyanis ennek más a leírása. Kétféle sánc lehet: kissánc vagy nagysánc. Ha sáncolás történik, akkor el kell dönteni, hogy kissánc vagy nagysánc, ezt az `ax` koordináta viszgálatával történik, ha `ax=6`, akkor kissánc, ha `ax=2`, akkor nagysánc. A kissánc jele: `0-0`, a nagysáncé: `0-0-0`. Ha nincs sánc, akkor a királybábu kezdőbetűjével

kell kezdeni, ami a K. Itt nem kell egyértelműséget vizsgálni, mivel csak egy király lehet a táblán. Tehát az ütést x betűvel jelölhetjük, majd pedig az átkonvertált hová koordinákat kell leírni.
Mindegyik bábunál - kivéve a királynál - a lépésleírás még egy plusz információt tartalmazhat az eddigiek után: sakk esetén +, matt esetén # jel szerepel a lépésleírás végén. Így történik a konvertálás PGN formátumra, ezt főleg mentésnél használjuk fel. A konvertálás másik irányát is meg kell valósítani, mert pl. fájl megnyitáskor vagy lépésmegadáskor PGN formátumot kell a programnak értelmezni és átkonvertálni négy koordinátára.
Ezt a konverziót egy ellenőrzés előzi meg, egy szintaktikai ellenőrzés, majd konvertálás után egy szemantikai, ha megfelel, akkor egy helyes átkonvertált lépésformátummal rendelkezünk.
A szintaktikai ellenőrzést az `Operator` osztály `ellenoriz` metódusa végzi, mely eldönti egy paraméterben megkapott sztringről, hogy szintaktikailag helyes PGN lépésleírás-e, ezt a logikai típusú visszatérési értékkel jelzi. A függvény használ két „segédfüggvényt", amely egy adott karaterről eldönti, hogy az érvényes oszlop- illetve sorazonosító-e.
Az `ellenoriz` függvény először megvizsgálja a sztring hosszát, ha kettőnél rövidebb, akkor hamis visszatérési értéket kapunk.
Ezután megvizsgálja az utolsó karaktert, ha ez '+' vagy '#', akkor ezt levágja a sztring végéről, és tovább folyik a vizsgálat.
Ha a sztring első karaktere 0, akkor a sztring vagy 0-0 vagy 0-0-0 lehet, ezekben az esetekben igaz visszatérési értéket ad, ellenkező esetben hamist.
A király-, királynő-, bástya-, futóbábuk lépésmegadása nagyon hasonló, csak az első karakterben térhetnek el a szintaktikailag helyes formátumok.
A függvény megvizsgálja a lépésmegadás első karakterét, ha ez K, Q, R, B, akkor megnézi, hogyha 3 hosszú a sztring, akkor az utolsó két karakter egy szabályos oszlop- és egy sorazonosító kell, hogy legyen ilyen sorrendben, ha így van, akkor igaz visszatérési értéket ad, ellenkező esetben hamist, mivel szabálytalan a formátum. Ha a sztring hossza 4, akkor a második karater x, a harmadik az oszlop-, a negyedik a sorazonosító kell, hogy legyen, ellenkező esetben helytelen a lépésmegadás. Öt hosszúságú sztring esetén a második karakter egy szabályos oszlop- vagy sorazonosító kell, hogy legyen, a harmadik x, a negyedik oszlop-, az ötödik pozícióban sorazonosító szereplése esetén helyes, egyébként helytelen a formátum.
Ha a sztring első karaktere kisbetű, akkor parasztbábu megadás lehet. Megvizsgálja, hogyha az utolsó karakter nagybetű, akkor ez csak K, Q, R, B lehet szabályos formátum esetén.

Amennyiben így van, ezt az utolsó karaktert a további vizsgálat idejére levágja. Ha kettő hosszúságú a sztring, akkor egy szabályos oszlop- és sorformátumból kell állnia. Ha négy hosszúságú, akkor az első karakter egy oszlopazonosító, a második x, a harmadik oszlop-, a negyedik sorazonosító kell, hogy legyen szabályos lépésmegadás esetén. Minden egyéb esetben a függvény hamis visszatérési értéket ad.

Amennyiben szintaktikailag helyes egy formátum, akkor kerülhet feldolgozásra. A feldolgozást az `Operator` osztály `alakit` eljárása végzi, amely két paraméterrel rendelkezik, egy sztring és egy állapot. Azért szükséges az állapot is, mert a lépés mindig egy adott állapothoz kötődik, teljesen más lépést jelenthet egy adott lépésmegadás más állapotban. Az eljárás célja, hogy egy PGN formátumot átalakítson négy koordinátára, ezek a koordináták az `Operator` osztály adattagjai. Az osztály rendelkezik logikai adattagokkal, ezek a lépésről adnak bizonyos információkat. Ezek az adattagok: `sanc`: sáncolás esetén igaz, `anp`: en passant esetén igaz, `csere`: parasztcserekor igaz, `sakk`: sakk adáskor igaz, `matt`: matt esetén igaz, `patt`: patt helyzet esetén igaz. Ezen adattagok értéke alapesetben hamis. Valamint rendelkezik egy `Babu` típusú `cser` adattaggal, amiben parasztcsere esetén tároljuk, hogy milyen bábura történt az átváltozás.

Ha a paraméterben kapott sztring első karaktere 0, akkor sáncolás történik az adott lépésben. Meg kell keresni az adott állásban következő játékos bábui közül a királyt, az `nx` koordináta ennek az oszlopazonosítója lesz, a `ny` a sorazonosítója, mivel a sáncoláskor a sorazonosító nem változik („vízszintesen" történik a sáncolás), ezért az `ay` koordináta is a király sorazonosítóját kapja értékül. Ha a sztring három hosszú, akkor rövid sánc, tehát `ax` adattag értéke 6 lesz, ellenkező esetben hosszú sánc és `ax` értéke 2. A `sanc` adattag értékét igazra állítjuk.

Ha a sztring kisbetűvel kezdődik, akkor ebben a lépésben paraszt lép. Megvizsgálja, hogy a sztring tartalmazza-e a + vagy a # jelet, ha igen, akkor az előbbi esetén a `sakk`, a második esetén a `matt` adattag értékét állítja igazra, és az átalakítás idejére levágja a végéről ezt a jelet. Ezek után megnézi, hogy a sztring utolsó karaktere nagybetű-e. Ha igen, akkor a `csere` adattagot igazra állítja, és ennek a betűnek megfelelően beállítja a `cser` adattagot is, és levágja a sztring végéről ezt a karaktert. Ezek után, ha a sztring kettő hosszú, akkor az első karakter oszlopazonosító lesz, a második pedig sorazonosító. Az oszlopazonosító átalakított értékét kapja az `nx` és az `ax`. A sorazonosítójét `ay`. Majd az eljárás keres egy olyan

parasztbábut, amelyik nx oszlopban található, és egy szabályos lépésben ax, ay koordinátájú mezőbe tud lépni. Ennek a bábunak a sorazonosítóját kapja ny értékül. Négy hosszúságú sztring esetén az első karakter az nx koordináta lesz, a harmadik az ax, a negyedik pedig az ay. Majd keresünk egy olyan parasztbábut, mely ax, ay mezőbe tud lépni az nx oszlopból, ennek sorazonosítója lesz ny.

Ha a sztring nem kisbetűvel kezdődik, és nem 0-val, akkor ötfajta nagybetűvel kezdődhet a sztring, mivel szabályos lépésmegadásról vagy szó. Ha N betűvel kezdődik, akkor ez egy ló lépés lesz. Megvizsgálja a függvény, hogy a sztring tartalmaz-e + vagy # jelet. + jel esetén a sakk, # jel esetén pedig a matt adattagok értékét igazra kell állítani, majd a további vizsgálódás idejére levágjuk ezt a sztring végéről. Ha a sztring tartalmaz x jelet, akkor ütés van. Ha ez az x jel a sztringben a második pozícióban van, akkor a sztring 3. karaktere lesz ax értéke, a 4. ay. Majd keresünk egy olyan lóbábut, mely egy szabályos lépésben ax,ay mezőbe tud lépni. nx, ny koordináták ennek a bábunak a koordinátái lesznek. Ha nem a második karakter az x, hanem a 3. karakter, akkor a 4. karakter lesz ax értéke, az 5. ay. Azért a 3. pozícióban van az x karakter mert ax, ay mezőbe több ló is tud egy szabályos lépésben jutni, vagyis a megkülönböztető sor- vagy oszlopkoordináta van a 2. pozícióban. Meg kell vizsgálni, hogy a 2. karakter betű vagy szám. Ha szám, akkor ez lesz ny , ha betű, akkor nx koordináta lesz. ny ismerete esetén keresni kell egy olyan bábut, mely ax, ay mezőbe tud lépni egy szabályos lépésben ny sorból, nx esetén nx oszlopból. Ennek a bábunak a koordinátáiból megállapítható a hiányzó nx/ny koordináta. Ha a sztring nem tartalmaz x jelet, ax a sztring 2. karaktere lesz, ay a 3., majd keresni kell egy olyan lóbábut, mely egy szabályos lépésben ax, ay koordinátájú mezőbe tud jutni, és ennek a bábunak a koordinátái lesznek nx és ny koordináták.

Ha a sztring B-vel kezdődik, akkor futó lépés lesz, R esetén bástya, K esetén király Q esetén pedig királynő lépés lesz, és az átalakítás megegyezik a ló lépésnél látott módszerrel.

## PGN fájlba írás

Itt szeretném egyrészt bemutatni, hogy hogyan valósítottam meg a fájlkezelést. A programban fájlba írás akkor történik, ha egy játékot le akarunk menteni. Ezt a MENTÉS gomb megnyomásával érhetjük el, ekkor egy újabb ablak nyílik, ahol különböző játékkal kapcsolatos információkat adhatunk meg. Itt két mező kitöltése az érdekes: az eredmény mező és az ECO mező. Az eredmény mező négyféle értéket vehet fel, ezeket egy `javax.swing.JComboBox`-ból választhatjuk ki. Ha patt vagy matt végeredménynél végzünk mentést, akkor nincs lehetőségünk választani, a program automatikusan beállítja ezt. A patt, matt eredményt egy globális logikai változó segítségével tudjuk meg. Az ECO mező egy leírás[6] a játéknyitásról, vagyis hogy a játék elején milyen lépések voltak. Abban az esetben, ha tetszőleges állásból (állásmegadás segítségével) indítottuk a játékot, akkor nem tudunk játéknyitás osztályról beszélni. Az ECO osztályok egy fájlban vannak letárolva. Ebben a fájlban szerepel az összes osztály és a hozzá tartozó lépések, valamint a világos ill. sötéttel játszó játékos vagy játékosok neve. Az osztályokhoz különböző hosszúságú lépéssorozatok vannak megadva. Abba az ECO osztályba fog tartozni a játszma, amelyik egyrészt illeszkedik a játszma első lépéseihez, másrészt az illeszkedők közül a leghosszabb. Ezt az ECO fájlt a program során csak egyszer dolgozzuk fel, utána megőrizzük ezt. Az ECO fájlban egy osztálybejegyzés az `Eco` osztály példányával van reprezentálva. Ennek az osztálynak négy adattagja van: `osztaly, white, black, minta.` Mindegyik `String` típusú, ezekben tároljuk az adott osztályról az információkat. Ezeket a példányokat egy `java.util.Vector`-ban[7] tároljuk.

```
[Site "A57"]
[White "Benko gambit"]
[Black "Zaitsev system"]

1. d4 Nf6 2. c4 c5 3. d5 b5 4. cxb5 a6 5. Nc3
```

*ECO fájlból részlet*

[6] http://observer.homestead.com/openings.html
[7] http://java.sun.com/j2se/1.5.0/docs/api/

Az ECO fájl beolvasásakor soronként olvassuk be az állományt. Egy osztály beolvasása után ezt a sztringet átadjuk az `Eco` osztály `feldolgoz` metódusának. Ez a metódus beállítja a példány adattagjait. Ezután ezt az Eco példányt a vektorhoz fűzzük. Ha elkészült a vektor, akkor ez a program futása során végig élni fog, célszerűbb egy ekkora vektort megtartani, mint minden mentésnél egy körülbelül 12000 soros állományt feldolgozni. A mentés folyamán, amikor ECO osztályt kell választanunk, meg kell keresnünk a vektorban a legjobban illeszkedő ECO osztályt. Ehhez a mi játszmánkat is le kell képezni az ECO osztályban megadott mintára, hogy egyezőséget tudjunk vizsgálni. Felmerülhet a kérdés, hogy miért nem az osztályban megadott mintát konvertáljuk a játszma leírásunkhoz. Egyrészt azért, mert a játszmák nagy részében az adott játszma lépéseinek hossza rövidebb, mint az összes ECO osztály mintája, másrész pedig az ECO osztályban megadott formátum a szabványos formátum, és mentésnél is ezt kell használni. Ha átkonvertáltuk a játszmánkat, akkor végig kell nézni az egész vektort. Azért kell az egész vektort, mert nem csak egy illeszkedőt keresünk, hanem a legjobban illeszkedőt. Ha megtaláltuk az illeszkedő ECO osztályt, akkor ennek az osztálynak a nevét írjuk a Mentés ablak ECO mezőjébe, valamint kitöltjük a sötéttel illetve a világossal jásztó játékos nevét is, ezek az információk az adott `Eco` példány `black` és `white` adattagjából kiolvashatók. Ezen információk megadása, beállítása után tudunk ténylegesen menteni, a Mentés ablakban szereplő Rendben gomb megnyomásával.

```
    javax.swing.filechooser.FileFilter szuro = new
javax.swing.filechooser.FileFilter(){
            public boolean accept(File fajl){
                String name = fajl.getName();
                if (fajl.isDirectory()) return true;
                if (name != null){
                    name = name.toLowerCase();
                    if (name.lastIndexOf(".pgn") != -1 )
return true;
                }
                return false;
            }
            public String getDescription(){
```

```
                return "PGN - Portable Game Notation";
            }
        };
```

Ekkor egy új `Jdialog` -ként egy új `JFileChooser` nyílik meg[8]. Így tudjuk majd kiválasztani a helyét, és megadni a fájl nevét, amibe írni akarunk. Ennek a fájlválasztónak a címkéjét a `setDialogTitle` metódussal tudjuk megadni. Egy szűrőt is alkalmaztam, hogy csak a pgn kiterjesztésű fájlok jelenjenek meg az ablakban, a könyvtárokon kívül. Ehhez egy `javax.swing.filechooser.FileFilter`-t kell létrehozni, és ebben beállítani az `accept` metódusával. Ez a metódus akkor tér vissza igaz értékkel, ha paraméterben kapott `java.io.File` vagy könyvtár vagy pgn kiterjesztésű fájl. A fájlleírást a `getDescription` metódussal tudjuk beállítani.

```
        jFileChooser2.setFileFilter(szuro);
        jFileChooser2.setDialogTitle("Mentés...");
        try{
        jFileChooser2.setCurrentDirectory(new File(new
File(".").getCanonicalPath()));
            }
        catch(IOException ioe){}
```

Ezt a szűrőt a fájlválasztóhoz kell rendelnünk a `jFileChooser_nev.setFileFilter (szuro_nev);` utasítással. A fájlválasztónak megadható, hogy a könyvtárstruktúra mely pontját mutassa kiinduláskor. Ez a `setCurrentDirectory` metódussal állítható be. A fájlválasztón történő eseményeket kezelnünk kell. Meg kell adni mindkét gomb megnyomása esetén, hogy mi történjen. Azt, hogy melyik gombot nyomtuk meg, az esemény `getActionCommand` metódusával kérhető le.

```
if(evt.getActionCommand().equals("CancelSelection"))
      jDialog1.hide();
```

[8] Nyékyné Gaizler Judit: Java2 utikalauz programozóknak 1.3 I.,II. kötet, ELTE TTK Hallgatói Alapítvány, Budapest, 2000, ISBN: 963-463-364-1

Cancel gomb megnyomása esetén az ablakot be kell zárnunk, ezt a `hide()` metódussal tesszük meg.

```
if(evt.getActionCommand().equals("ApproveSelection")) {...}
```

Save gomb megnyomásakor is be kell zárnunk az ablakot a `hide()` utasítással, a `getSelectedFile()` metódussal meg tudjuk kapni a kiválasztott fájlt. Egyszerű sztring műveletekkel megvizsgáljuk, hogy tartalmazza-e a kiterjesztést (.pgn), ha nem, akkor a kiterjesztést a fájlnévhez fűzzük. A fájlba írást `java.io.PrintWriter` segítségével végezzük. Ha megvan a fájlnév elérési útvonallal, akkor ezt a `PrintWriter`-t példányosítjuk `new PrintWriter(new FileWriter(fajl_nev));` utasítással.

```
out=new PrintWriter(new FileWriter(sb.toString()));
```

Ekkor egy out csatornánk van, melybe írhatunk. Ezt megtehetjük pl. a `println()` utasítással, mellyel sorokat írhatunk a fájlba. Kezelnünk kell a fájlkezelésből adódható kivételeket. Ezért ezeket az utasításokat kivételkezelővel kell ellátnunk, try / catch / finally blokkba kell raknunk. `catch`-csel az I/O hibákat kezeljük, a `finally` részben pedig egy újabb try/catch blokkban lezárjuk a fájlt, de itt már nemcsak az I/O hibákat kezeljük.

```
try{
      out=new PrintWriter(new FileWriter(sb.toString()));
      out.println("[Event \""+event+"\"]");
      out.println("[Site \""+site+"\"]");
      out.println("[Date \""+date+"\"]");
      out.println("[Round \""+round+"\"]");
      out.println("[White \""+white+"\"]");
      out.println("[Black \""+black+"\"]");
      out.println("[Result \""+result+"\"]");
      }
   catch(IOException ioe){}
finally{try{out.close();}catch(Exception e){}}
```

Tehát a fájlba írás három fő részből áll: fájl megnyitása, fájlba írás, fájl lezárása.

## PGN fájlból olvasás

A lementett játékok PGN fájlban vannak tárolva. Kétféle PGN fájl létezik, az egyik típusnál egy fájl egy játékleírást tartalmaz, a másiknál egy fájlban több játékleírás található. Mindkét típusú fájlnál a beolvasás ugyanúgy történik, csak a feldolgozásban tér el a kettő. A beolvasni kívánt fájlt egy fájlválasztó segítségével választhatjuk ki, hasonlóan, mint fájlíráskor. Fájlt beolvasni a Megnyitás gomb segítségével lehet, ekkor egy fájlválasztó nyílik meg, egy új `Jdialog` -ként egy új `JFileChooser` . Ezt a fájlválasztót a fájlíráskor használt fájlválasztóhoz hasonlóan kell beállítani. A fájlválasztóban alapesetben csak PGN fájlok jelennek meg. Fájlkiválasztás után az Open gombra kattintva tudjuk megnyitni a fájlt.

```
      String sg;
      try{
         BufferedReader in=new BufferedReader(new
FileReader(jFileChooser1.getSelectedFile().toString()));
        while((sg=in.readLine())!=null) {...}
        }
        catch(IOException ioe){}
        finally{
                 try{in.close();}catch(Exception e){}
               }
```

A fájlt egy `BufferedReader`-ként nyitjuk meg. A fájl megnyitást és beolvasást kivételkezelővel kell ellátni, vagyis try/catch/finally blokkba kell helyezni. A fájlt soronként olvassuk be egy while ciklus segítségével. Ez a ciklus akkor áll meg, ha már nincs mit beolvasni, vagyis a beolvasott adat `null`. A beolvasott sort egy `String`-ként kezeli, ha ez 0 hosszúságú vagy `%` jellel vagy `;` -vel kezdődik, akkor ezt a sort figyelmen kívül kell hagyni, mivel ezek a szabványos PGN állományban a megjegyzést jelölik. A PGN fájlban minden leírás EVENT sorrendi címkével kezdődik. Tehát beolvasáskor vizsgálom a sorokat, ha nem az első EVENT bejegyzést találom, akkor az eddig beolvasott rész egy külön játszmát fog alkotni, vagyis más szavakkal az állományt EVENT kezdetű sorok mentén játszmákra bontom. Ezeknek a játszmáknak a tárolására szolgál a `Jatszma` osztály. Ez az osztály tizenegy `String` adattagot tartalmaz, ezek a játszma jellemzőinek tárolására szolgálnak.

Ebből a tizenegy adattagból tíz sorrendi címke, a fennmaradó egy adattag pedig a lépéssorozatot tárolja. Több játszma esetén a választásra egy újabb ablak nyílik, ebben az ablakban egy táblázat található. A táblázat sorai egy-egy játszmát jelképeznek, oszlopai az adott játszmáról adnak információkat. A táblázatot `java.swing.JTable` osztállyal oldottam meg. Ennek a táblázatnak a megfelelő testre szabását egy `TableModel` interfész implementálásával végeztem.

```
Vector v;
TableModel dataModel = new AbstractTableModel(){
        public int getRowCount() {
           return v.size();
        }
        public int getColumnCount() { return 4;}
        public String getColumnName(int columnIndex){
        if (columnIndex==0) return new String("Date");
        if (columnIndex==1) return new String("White");
        if (columnIndex==2) return new String("Black");
        if (columnIndex==3) return new String("ECO");
        return null;}

        public Object getValueAt(int row, int col){
        if(col==0)return ((Jatszma)v.get(row)).date;
        if(col==1)return ((Jatszma)v.get(row)).white;
        if(col==2)return ((Jatszma)v.get(row)).black;
        if(col==3)return ((Jatszma)v.get(row)).eco;
        return null;

        }
    };
```

A sorok számát a `getRowCount` metódus definiálásával tudjuk beállítani, az oszlopok számát pedig a `getColumnCount` –tal. Az oszlopneveket a `getColumnName` metódussal tudjuk megadni. A táblázat celláiban szereplő adatok a `getValueAt`

metódussal állíthatók be. Az interfész implementálása után ezt a táblázathoz kell rendelni a `jTable1.setModel(dataModel);` paranccsal.

Ha több játszma található a PGN fájlban, választás után teljesen egyenértékű azzal, mintha csak egy játszmát tartalmazna, vagyis csakis egy `Jatszma` példányt kell feldolgozni. A feldolgozást, mint műveletet két fő lépésre lehet bontani. Az egyik az, amikor a megkapott `String` halmazt kell leképezni egy `Jatszma` példányra, a másik pedig mikor a sztringet kell értelmezni, a sztringként tárolt lépéssorozatot felbontani lépések sorozatára. A leképezést a `Jatszma` osztály konstruktora végzi. Ez vizsgálja a sorok elejét, ha talál sorrendi címkét, akkor a sor további részét a megfelelő adattagban eltárolja. Az adott játszma feldolgozásának azon részét, amikor értelmezni kell a beolvasott adatot, a `Sakk` osztály `feldolgoz` metódusa végzi. Ez az eljárás paraméterként egy `Jatszma` példányt kap.

```
public void feldolgoz(Jatszma j){...}
```

A `Jatszma` objektum adattagjainak értékét, kivéve a lépéssorozatot, átadja a `Sakk` osztály megfelelő adattagjainak. A lépéssorozatot, mely egy sztring, `java.util.StringTokenizer` segítségével sorvégjelek mentén sztringekre bontom. Majd ezeket a sztringeket szóköz mentén újabb sztringekre vágom, melyeket egy vektorban fűzök fel.

```
java.util.Vector vs=new java.util.Vector();
StringTokenizer stok=new StringTokenizer(j.jatsz,"\n",false);
  while(stok.hasMoreTokens())
     {sg=stok.nextToken();
      st=new StringTokenizer(sg," ",false);
        while(st.hasMoreTokens()){
          stt=st.nextToken();
            vs.add(stt);}}
```

Ennek a vektornak az elemei egy-egy PGN lépésmegadást tartalmaznak, ennek kétféle ellenőrzését el kell végezni. A szintaktikai ellenőrzést az `Operator` osztály `ellenoriz` metódusa végzi. Ha szintaktikailag helyes a lépésmegadás, akkor szemantikailag is ellenőrizni kell. Ehhez úgymond végig kell játszani a játszmát, vagyis a játék alapállás állapotára - kivéve ha a játék nem alapállásból indul, akkor a megfelelő kezdőállapotra - alkalmazni kell az első lépést, ha megengedett a lépés, akkor az előálló állapotra meg kell próbálni alkalmazni a következő lépést, és így tovább egészen a vektor utolsó eleméig, vagy amíg hibát nem

találunk. Ha szintaktikailag vagy szemantikailag hibásnak ítélünk egy lépésmegadást, akkor a lépésbeolvasás, feldolgozás itt megáll, a vektor további elemei nem kerülnek feldolgozásra. A PGN fájl első részében nem tudunk hibát keresni, mivel nem kötelező minden címkét megadni, és a címke adatrészére sincs semmiféle kritérium, így a fájl második felében lehetséges csak a hibakeresés. Megnyitott fájl feldolgozása után a lépések sorozata egy vektorban található. Ezeken a lépéseken a navigáló gombokkal tudunk végighaladni.

## Tábla leképezése, kirajzolása

A sakktáblán 64 mező található, bármelyik mezőben állhat bármelyik játékos bármelyik bábuja, természetesen egyszerre csak egy. Kétféle leképezési mód a legkézenfekvőbb. Az egyik, hogy az összesen lehetséges 32 bábut tároljuk, és azt az információt tároljuk róluk, hogy hol található a táblán. Ez azért jó tárolási mód, mert a játék vége felé, amikor már (nagy valószínűséggel) kevesebb bábu található a táblán, akkor kevesebb adattal kell dolgozni, viszont az adatokkal való dolgozás bonyolult. Gondoljunk például arra, hogy amikor egy futóval lépünk, akkor meg kell vizsgálni, hogy az átlépett mezőkön nem található-e bábu. Ehhez legrosszabb esetben az összes bábut meg kell vizsgálni, hogy hol találhatók, és megnézni, hogy az adott koordináta az átlépett mezőkhöz tartozik-e.

A másik egyszerű tárolási mód az, hogy a táblát, mint egy nyolcszor nyolcas mátrixot kezeljük, és ezen tároljuk a bábukat. Ennél a megoldásnál mindig 64 elemmel kell dolgozni, de ha az előbbi példára gondolunk, akkor a mezők vizsgálata sokkal egyszerűbb. Tehát nagyobb adattal kell dolgozni, de könnyebben. A program ezt az utóbbi leképezést valósítja meg. A mátrix elemei `Babu` típusú objektumok.

```
Babu[][] t=new Babu [8][8];
```

A `Babu` osztály leszármazottjai a hattípusú bábu osztályai. Ha az adott pozícióban található bábu, akkor a megfelelő bábu osztálynak a példánya található a mátrixban, ha üres a mező, akkor `null`.

A tábla megjelenítésénél is jó ez a fajta ábrázolási mód, mivel végigmegyünk a táblán, ha valamelyik mezőn találunk bábut, akkor azt kirajzoljuk. A tábla kirajzolás egy vászon segítségével történik. Ehhez egy új osztályt kell létrehozni a `java.awt.Canvas` osztály leszármazottjaként.

```
public class STabla extends Canvas{…}
```

Ennek az osztálynak egy példányát rárakjuk egy panelra, ahová rajzolni akarunk.

```
STabla tabla = new STabla();
JPanel1.add(tabla);
```

Az `STabla` osztálynak a `paint` metódusa szolgál a rajzolásra. Ez egy felüldefiniált metódus. `public void paint(java.awt.Graphics g) {…}`

A rajzolást 3 részre bontottam: táblarajzolásra, táblafelirat rajzolására, és a bábuk rajzolására. Rajzolásnál a `Graphics` osztály példányát is át kell adni.

A táblafelirat rajzolása `tf` metódussal:

```
public void tf(java.awt.Graphics g){
     int bx=5;
     int by=5;
     g.setColor(new Color(0,0,0));
     g.setFont(new Font("sajat",4,20));
      g.drawString("A",ox+m/2-bx,oy-by);
      g.drawString("B",ox+m+m/2-bx,oy-by);
      g.drawString("C",ox+2*m+m/2-bx,oy-by);
}
```

A szín beállítása a `g.setColor(new Color(0,0,0));` utasítással történik, a `setColor` metódusnak paraméterként egy `java.awt.Color` példányt kell adni, mely egy RGB szín reprezentálása. Majd beállítható a megfelelő betűtípus a `g.setFont(new Font("sajat",4,20));` paranccsal. Táblafelirat betűt a `g.drawString ("A",ox+m/2-bx,oy-by);` utasítással íratjuk ki, ez a metódus három paraméterrel rendelkezik: a kiírandó szöveg, x koordináta és y koordináta.

A tábla kirajzolása a `tabla` metódussal történik.

```
   public void tabla( java.awt.Graphics g){
     Color c1=new Color(175,223,233);
     Color c2=new Color(51,137,155);
     g.setColor(new Color(0,0,0));
     int tm=8*m;/*tabla méret*/
     g.drawRect(ox,oy,tm,tm);
     for(int i=0;i<7;i++)
     {g.drawLine(ox+(i+1)*m,oy,ox+(i+1)*m,oy+tm);
      g.drawLine(ox,oy+(i+1)*m,ox+tm,oy+(i+1)*m);
     }
     g.setColor(c1);
     for(int i=0;i<8;i++)
         for(int j=0;j<8;j++)
         {if ((i+j)%2==0)g.setColor(c1); else g.setColor(c2);
          g.fillRect(i*m+ox+1,j*m+oy+1,m-1,m-1);
         }
```

```
}
```

Itt beállítok két színt (c1 és c2), ezek lesznek a mezők alapszínei. Majd megrajzolom a tábla határvonalait, mint négyzetet a `g.drawRect(ox,oy,tm,tm);` utasítással, melynek négy paramétere van: a kiinduló x és y koordináta és szélesség, illetve a magasság. Majd a táblát berácsozom, tehát hét vonalat húzok vízszintesen, és hetet függőlegesen a `g.drawLine(ox+(i+1)*m,oy,ox+(i+1)*m,oy+tm);` paranccsal, amelynek négy paramétere a kezdőpont x és y koordinátái, és a végpont x és y koordinátái. Ekkor már megrajzoltuk a sakktábla vonalait, már csak ki kell színezni a megfelelő színnel. A megfelelő szín kiválasztása úgy történik, hogyha a sor- és oszlopkoordináta összege páros, akkor az egyik színnel, ha páratlan, akkor a másik színnel lesz kitöltve az adott mező. A sötétebb és világosabb háttérszínek egymást felváltva követik, és adott, hogy melyikkel kell kezdeni. A kitöltésre a `g.fillRect(i*m+ox+1,j*m+oy+1,m-1,m-1);` utasítás használható.

A bábuk kirajzolását a `public void rajzol(java.awt.Graphics g){…}` metódus végzi. A hattípusú bábuhoz egy-egy JPEG kép tartozik. Ezeket a képeket beolvasom, és ennek megfelelően rajzolom ki a képet. A mátrixból kiolvasható, hogy az adott mezőn található-e bábu, ha igen, milyen típusú, és az ennek megfelelően kiválasztott képet kell rárajzolni a mezőre. A JPEG kép beolvasásához egy BufferedImage –t hozok létre. Ezekbe olvasom be a képeket.

```
BufferedImage bim1 =(JPEGCodec.createJPEGDecoder(new
FileInputStream("bastya.jpg")).decodeAsBufferedImage());
```

Ennek a BufferedImage példánynak a pontjait egyenként el lehet érni, és lekérdezni az itt található képpont színét.

```
t=bim1.getRGB(i,j);
```

A kép két színt tartalmaz, ha a pont nem háttérszínű, akkor ebben a pozícióban rajzolunk egy megfelelő színű pontot a mezőre. Azért nem a konkrét képet rajzoljuk ki, mert akkor a két játékosnak megfelelően mindegyik képből kettő kellene. Így viszont a megfelelő színnel lehet kirajzolni. Azért nem a program forrásszövegében tárolom a képet, mert így könnyebben lecserélhető. Azért, hogy a kirajzolt kép jobban látható legyen az adott mezőn, a kép körvonalát az ellenkező színnel rajzoljuk meg. Tehát kirajzoláskor a beolvasott képet vizsgálom, és ha valamelyik szomszédos képpont ellenkező színű (vagyis az adott pont a körvonal része), akkor ezt a pontot ellenkező színnel rajzolom meg. Képpont kirajzolása úgy

történik, hogy egy olyan vonalat rajzolunk ki, amelynek kezdő- és végpontja ugyanaz a pont. Az így rajzolt pontnak a színe is a `setColor()` metódussal állítható be.

```
g.setColor(new Color(0,0,0));
g.drawLine(x+i,y+j,x+i,y+j);
```

A `Canvas` osztály rendelkezik `repaint` metódussal, aminek megadható egy tartomány, hogy a vászon melyik részét frissítse, vagyis pl. egy bábu lépésekor nem kell az egész sakktáblát újrarajzolni, elég csak azt a mezőt, ahonnan ellépett, és azt a mezőt, ahová lépett. Ezzel egyrészt időt és energiát takarítunk meg, másrészt a kép „vibrálása" ezzel megelőzhető. A `repaint` metódus négy paraméterrel rendelkezik, ezek a tartomány két csúcsának két-két koordinátái.

A játék során egy lépés megadható úgyis, hogy a táblán két mezőre kattintunk. Egyszer arra a mezőre, ahol az a bábu található, amelyikkel lépni akarunk, majd arra a mezőre, ahová lépni szeretnénk. Ebből a program felépít egy lépést, ellenőrzi azt, majd ha szabályos, akkor alkalmazza a lépést, és az új állapotot kirajzolja. Tehát a táblán való kattintást a programnak kezelnie kell, ezt a

```
public void STablaMouseClicked(java.awt.event.MouseEvent evt)
{…}
```

metódus végzi. Az `evt` paramétertől lekérdezhető az a pont, ahová kattintottunk, és ebből kiszámolható, hogy ez melyik mezőnek megfelelő, illetve hogy egyáltalán a táblán kattintottunk-e.

```
Point p=evt.getPoint();
```

Meg kell különböztetni a két kattintást, ugyanis ha először kattintunk, és olyan mezőre kattintottunk, melyen bábu található (ez könnyen eldönthető, hisz a kattintás pozícióját átalakítjuk oszlop- és sorkoordinátára, majd a kétdimenziós mátrixban megnézzük, hogy adott pozícióban van-e bábu, és annak a játékosnak a bábuja van-e, amelyik lépni következik), akkor a mező körül egy piros négyzet rajzolódik ki, ezzel kijelölve az adott bábut. A második kattintás esetén, át kell alakítani a kattintás pozícióját oszlop- és sorkoordinátára, az így előálló lépést megvizsgálni, hogy alkalmazható-e, ha szabályos a lépés, akkor az előálló állapotnak megfelelően újra kell rajzolni a tábla megfelelő részeit, és a piros négyzetet le kell törölni. Szabálytalan lépés esetén is el kell távolítani a kijelölő négyzetet, de a táblán újrarajzolás nem történik. Tehát mindkét esetben át kell alakítani a kattintás pozícióját koordinátákra. Külön kell kezelni az x és y koordinátákat. Mindkettőnél ugyanúgy lehet az

átalakítást elvégezni, x koordináta esetén az adott pozícióból ki kell vonni azt a távolságot, amely vízszintesen van a vászon széle és a tábla széle között, majd el kell osztani egy mező szélességével. Ha az így kapott szám 0 és 7 közé esik, akkor a táblán történt a kattintás (természetesen, ha az y koordináta is megfelel ennek a feltételnek). Ez a szám könnyen átalakítható oszlop- és sorkoordinátára. Annak eldöntése, hogy először vagy másodjára történt kattintás, egy logikai változóval követhető. A beszúrt kódrészletekből látható, hogy sehol sem konkrét koordinátákat adtam meg a pozíciók megadásánál, hanem paraméteresen, hogy esetlegesen később más méretben, más helyen történő rajzolás lehetősége adott legyen.

## Adott állás megadása

Egy játékot nemcsak az alapállásról lehet kezdeni. A játék bármely pontján lehet adott állásról indítani a játékot. Ilyenkor mindig új játszma indul. Adott állásmegadásnál lehetőségünk van a táblán a jelenleg található állás importálására, vagy választhatjuk az alapállást is, illetve ezek választása után/helyett tetszőlegesen változtathatók a beviteli mezők tartalmai. Az állásmegadás elfogadása esetén ellenőrizni kell a bevitt adatokat, illetve le kell tárolni az állást, amiről indul a játék, és mentés esetén pedig ezt az állást PGN szabványnak megfelelően a fájlban is le kell tárolni. A beviteli mezők `JTextField` objektumok, melyeket a

```
javax.swing.JTextField  jTextField1 = new javax.swing.JTextField();
```

paranccsal tudunk létrehozni. A szövegmezőbe írni

```
jTextField1.setText("a1");
```

paranccsal lehetséges. A szövegmező tartalmának lekérdezése

```
jTextField1.getText();
```

parancs segítségével történik.

A jelenlegi állás importálása úgy történik, hogy az összes beviteli mezőt töröljük. Majd végigmegyünk a táblán, és ha találunk egy bábut, akkor a típusának megfelelő első üres beviteli mezőbe beírjuk a tábla mező koordinátáit. Ha több azonos típusú bábu található a táblán, mint ahány hely van az adott bábutípusnak (pl.: 3 azonos színű ló található a táblán, mert parasztátváltozásnál ló lett választva), ilyenkor egy parasztbábuhoz kell beírni a mező koordinátáját a megfelelő bábu rövidítésével kiegészítve! Ezt a programnak fel kell ismernie, ezért kell a legelején kitörölni a beviteli mezőket, hogy a mező kitöltésével jelölhető legyen a bábuk száma. Mivel parasztátváltozás során jutottunk több azonos típusú bábuhoz, így ennek a parasztbábunál történő megadása nem veszi el a helyet a parasztbábuktól, illetve az is teljesül, hogy azonos színű bábuból maximum 16 helyezhető el a táblán tetszőlegesen.

Az alapállás választásánál egy előre megadott, letárolt állással tölti ki a program a mezőket, ez az az állás, amelyikkel indul a program.

Miután beállítottuk a kívánt állást, a Rendben gombbal elfogadtathatjuk azt. Ilyenkor egy ellenőrzés történik. A nem paraszt típusú bábunál a beviteli mezőben kettő hosszúságú sztring lehet, melynek első karaktere egy betű 'a' és 'h' között, a második pedig 1 és 8 közötti szám lehet. Parasztbábunál szereplő beviteli mezőben a sztring vagy az előbb említett formájú lehet, vagy három hosszúságú: az első egy betű 'a' és 'h' között, a második 1 és 8 közötti szám, a harmadik pedig 'Q', 'N', 'R', 'B' betűk valamelyike lehet. Ez a mező szintaktikai

ellenőrzése, ezen túl van egy szemantikai ellenőrzés, amely abból áll, hogy egy táblamező koordinátája nem adható meg kettő vagy több helyre. Hibás helyzetmegadás esetén egy hibaüzenet jelzi ezt a felhasználó felé, és az első hibásan kitöltött beviteli mezőbe lesz a kurzor. Ezt a `jTextField9.requestFocus(true);` paranccsal tudjuk elérni.

Amennyiben helyesen megadtunk egy számunkra megfelelő állást, a program ezt, mint kezdőállapotot eltárolja.

Ha egy olyan játékot akarunk menteni, amelyet nem alapállásból indítottunk, akkor ezt mentésnél jelezni kell. Ha nem jeleznénk, és következő beolvasásnál úgy értelmeznénk a lépéseket, mintha alapállásról indulnánk, lehet, hogy már az első lépésmegadás szemantikailag hibás lenne, értelmezhetetlen arra az állásra. A PGN fájl az adott állás jelzésére és tárolására két sorrendi címkével rendelkezik. Ezek a SetUp és a FEN címkék. A SetUp címke, ha szerepel és 1 az értéke, az azt jelzi, hogy nem alapállásból induló játék van lementve, és csak ekkor lehet FEN címke, amiben pedig az adott állás leírása található. A FEN a Forsyth-Edwards Notation szavak rövidítéséből származik. A FEN címke 6 részből áll, de számunkra ebből csak az első két rész a lényeges, a részeket szóköz határolja. A címke első része szolgál az állapot letárolására. Ez 8 részből áll, mindegyik rész egy-egy sort jelképez, ezek a sorokat jelképező karaktersorozatok perjellel vannak elválasztva, a nyolcas sortól indul, és sorban halad az egyes sorig. A sorok leírása úgy történik, hogy a mezőket vizsgálja sorba az ’A’ oszloptól a ’H’ oszlopig. Amennyiben talál egy bábut, akkor annak a rövidítését leírja, ha a fehérrel játszó játékoshoz tartozik, nagybetűvel, ha a feketéhez, akkor kisbetűvel. Ha üres mezőt talál, ezt megjegyzi, és megy a soron következő mezőre; ha olyan mezőt talál, amelyiken bábu van, vagy a sor végére ér, akkor megszámolja, hogy idáig folytonosan hány darab üres mezőt talált, és ezt egy számmal jelzi. Pl. a táblán két bábu található, a fehér király a ’a2’-es mezőn, a fekete király pedig a ’f6’-os mezőn. Ekkor a FEN címke első része a következőképpen néz ki:

8/8/5k2/8/8/8/K7/8 . A címke második részében azt jelezzük, hogy az adott állásban ki következik lépni, ha a fehérrel játszó játékos, akkor ’w’-t, ha a fekete, akkor ’b’ –t írunk ebbe a részbe. A címke többi 4 részébe nem írunk adatot. Pl.: egy szabályos FEN címke:

[FEN "8/8/5k2/8/8/8/K7/8 w - - - -"]

Természetesen adott állásról induló játék mentésekor nincs értelme játéknyitási osztályról (ECO) beszélni, így ezt a program nem is vizsgálja, és mentésnél is az ECO mező, címke üresen marad.

## Sakk, matt, patt vizsgálat

A sakk és matt értelmezéséről az első fejezetben már volt szó. A játék fontos részei és célja a sakk-, illetve a főcélja a matt-adás az ellenfélnek. A játékban ezek az események csak lépés után következhetnek be, ezért ezeknek a vizsgálatát is ekkor kell, hogy elvégezze e program. Minden lépés után automatikusan el kell végezni az ellenőrzést. A sakk vizsgálatát a `Sakk` osztály `sakk` metódusa végzi, amely logikai visszatérési értékkel rendelkezik. Nem biztos, hogy a lépett bábuval adunk sakkot, így nem elég ennek a bábunak vizsgálni a lehetséges ütésmezőit, hanem az adott játékos összes bábujának összes lehetséges lépését vizsgálni kell. Abban az állásban, amelyikben sakk adódik, azt az aktuálisan lépni következő játékos kapja, vagyis a lépett bábuval rendelkező játékos. Tehát a metódus először a lépni következő játékos ellenfelének bábuiról készít egy 8x8-as bitmátrixot, melyben eggyel van jelölve egy mező, ha oda egy szabályos lépésben bábu tud lépni, nullával ellenkező esetben. Ennek a mátrixnak a kitöltése úgy történik, hogy a program végighalad az adott sakktáblán, ha valamelyik mezőn talál bábut, megvizsgálja, hogy a vizsgált játékosé-e, aztán megvizsgálja, hogy ez a bábu mely mezőkre léphet, hogy ne kelljen az összes 64 mezőt megvizsgálni, ezért mindegyik bábutípushoz meg van adva az összes lehetséges lépés helye. A mátrix kitöltése után a program megkeresi a lépni következő játékos királybábuját, és megnézi, hogy a mátrixban azon a mezőn, ahol a király található, nulla vagy egyes szerepel. Egy esetén a visszatérési érték igaz, egyébként hamis.

A matt vizsgálata hasonló a sakk vizsgálatához, mivel a matt egy különleges sakk. Olyan állapot, melyben sakkot kapott a lépni következő játékos, de ezt elhárítani nem tudja. A vizsgálatot a programban a `Sakk` osztály `matt` metódusa végzi, amely szintén logikai visszatérési értékű. A függvény csak azt vizsgálja, hogy a lépni következő játékosnak van-e olyan lépése, amely megengedett. Ugyanis csak olyan lépés megengedett egy játékos számára, melynek hatásaként előálló állapotban a király bábuja nincs ütés alatt, vagyis nem kerül vagy marad sakkban. A függvénynek sorba kell venni a lépni következő játékos bábuit, és meg kell nézni, hogy van-e olyan lehetséges lépése, amely megengedett, hogy ne kelljen minden mezőt megvizsgálni lehetséges lépésként, ezért felhasználja az előbb említett lépéshely megadást a bábukhoz.

A patt vizsgálatához nem kell külön függvény, mivel a patt egy olyan végállapot, melyben a lépni következő játékosnak nincs megengedett lépése, és nem kapott sakkot az előző

lépésben. A matt függvény pontosan azt vizsgálja, hogy az adott játékosnak van-e szabályos lépése, vagyis erre a vizsgálatra már létezik függvény.

A lépésmegadás után a következő folyamatú vizsgálat zajlódik le, mely négy eredménnyel zárulhat: nincs semmi „különleges" hatása a lépésnek, sakk, matt, patt. Meghívja a sakk és matt függvényt. Ha mindkettő hamis, akkor mivel nincs hatása a lépésnek a vizsgálat szempontjából, így nem íratunk ki semmit. Ha a sakknak igaz, de a mattnak hamis a visszatérési értéke, akkor sakkot adott a játékos, és erről egy üzenetben tájékoztatja a program. Ha a sakk és matt is igaz visszatérési értékű, akkor matt eseményről kap tájékoztatást a játékos. Ha a matt igaz, és a sakk hamis akkor patt helyzetről beszélünk, és erről tájékoztat a program.

## Automatikus lépés – Minimax algoritmus alfa-béta vágással

A sakk egy kétszemélyes, véges, teljes információjú, determinisztikus, zérusösszegű, stratégiai játék. Kétszemélyes, vagyis a játékosok száma kettő, véges, tehát a játszmák véges sok lépés után véget érnek, másrészt minden állásban véges sok lehetséges lépése van a játékosnak. Teljes információjú alatt azt értjük, hogy ismerünk a játékkal kapcsolatban minden információt. Determinisztikus, vagyis a véletlennek nincs szerepe. Zérusösszegű, tehát a játékosok nyerési és vesztési esélyei egyenlőek. Mivel a sakk egy ilyen játék, így alkamazható a minimax algoritmus.[9] A minimax algoritmushoz kell egy heurisztika. A heurisztika egy függvény, ami egy adott álláshoz egy számértéket rendel. A két játékos legyen „A", „B". Ha az „A" játékosnak kedvez az adott állás, akkor egy pozitív számot adjon vissza a függvény, minél kedvezőbb az állás, annál nagyobb legyen ez a szám. Matt esetén pedig nagyon nagy legyen a szám, lehetőleg olyan, amilyen nagyot egyébként nem ad vissza a függvény. „B" játékos esetén negatív előjellel szerepeljen az érték. Semleges állás, vagyis egyik játékosnak sem kedvező helyzet esetén 0-t adjon vissza a függvény. Az algoritmus hatékonysága nagyban függ a heurisztikától, ugyanakkor meg kell jegyezni, hogy tökéletes heurisztika nem létezik. A programban használt heurisztikus függvény a következőképpen ad vissza értéket. A bábukhoz értéket rendel: királynő: 1000, bástya: 500, futó, ló: 300, paraszt:100. Az adott állásban a függvény összeadja a bábuk értékeit, mégpedig úgy, hogy „A" játékos bábuértékei pozitív előjellel szerepelnek az összegben, „B" játékos esetén pedig negatív előjellel. Ezután megszámolja, hogy „A" játékos a sakktáblán hány darab mezőt „birtokol", és „B" játékos mennyit. „A" játékos esetén pozitív előjellel, „B" esetén negatív előjellel kerül hozzáadásra ez a szám. Egy játékos a táblán birtokol egy mezőt, ha rendelkezik olyan bábuval, mely egy szabályos lépéssel ebbe a mezőbe tud jutni. Matt esetén pedig 15000-et vagy –15000-et ad vissza a függvény a játékosnak megfelelően. A heurisztika egyébként nem tud ilyen nagy értéket visszaadni, mert pl. „A" játékosnak legkedvezőbb esetben van kilenc királynője, kettő bástyája, kettő futója, kettő ló, 9*1000+2*500+2*300+2*300=11600, az ellenfél pedig csak a királyával rendelkezik, ekkor látszik, hogy hiába birtokolnánk az összes, 64 db mezőt, akkor is csak 11664 lenne a függvény visszatérési értéke, vagyis kisebb, mint matt helyzet esetén.

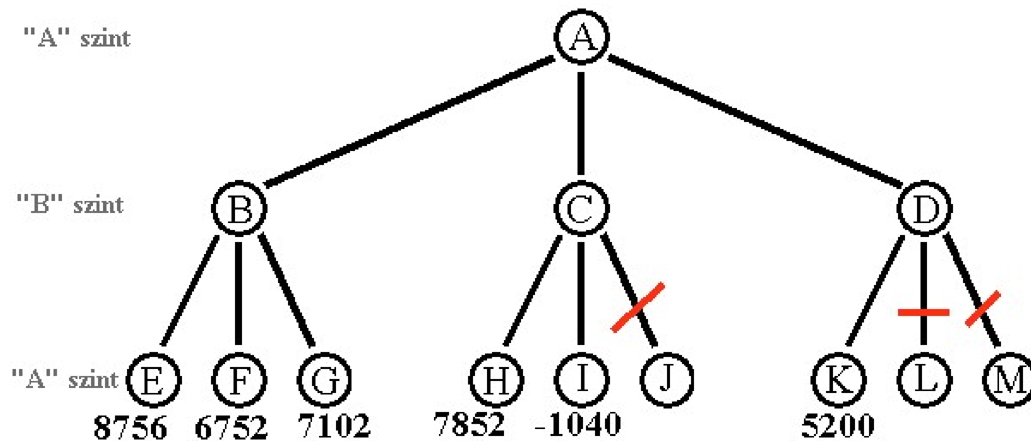
A játék minden állásából egy játékfa építhető. A fa elemei játékállások, egy faelemnek a gyerekei az adott állásból egy szabályos lépés eredményeként előálló állások. A fa szintjein a

[9] Dr. Szalay Tibor: A mesterséges intelligencia alapjai, Gábor Dénes Főiskola, Budapest, 2002

definíciónak megfelelően szintenként másik játékos állásai szerepelnek. A fa szintjeit feloszthatjuk „A” és „B” szintekre, melyek egymás után felváltva következnek.
A minimax algoritmus az adott állásból egy adott mélységig előállítja a játékfát. Ennek a fának a levél elemein kiszámolja a heurisztikáját. Majd szintenként visszafelé haladva minden elemnek megállapítunk egy értéket, „A” szinten az elem gyerekei értékeinek a maximumát, „B” szinten negatívumát számoljuk. Mikor elérjük a fa gyökerét, akkor olyan lépést ajánlunk, melyen keresztül olyan állásba jutunk, mely értéke megegyezik a fa gyökerének értékével. Ha több ilyen lépés is van, akkor a program véletlenszerűen választ, azért, hogy ugyanolyan állás esetén se legyen unalmas a játék, ne mindig ugyanazt a lépést ajánlja. A minimax algoritmus hatékonyságának két feltétele van: minél jobb heurisztika és minél mélyebben előállítani a részjátékfát. Mivel a fa elemeinek száma szintenként exponenciálisan növekedik, és ezeket elő kell állítani, így a program időigénye minden szinttel exponenciálisan nő. Ezért jól kell a mélységet és a heurisztikát is megválasztani. Ahhoz, hogy minél jobban, minél okosabban játsszon a program, két dolgot lehet tenni: jobb heurisztikát találni, vagy pedig mélyebben előállítani a játékfát. Heurisztikát lehet keresgélni, próbálgatni, de nem lehet előre tudni, csak sejteni, hogy egy adott heurisztika mennyire teszi „okossá” a program lépését. Tehát ebben az irányban nincs kijelölt út. A másik módszer a játékfát mélyebben előállítani. Egy adott állásban átlagosan 40 szabályosan megtehető lépés van. Ha a program két lépéssel gondolkodik előre, a játékfát kettő mélyen állítja elő, akkor 40 x 40=1600 állás jön létre. Ezen állásokban tegyük fel, 32 bábu található, akkor ha csak a bábukat számoljuk, 51200 példányt kell előállítani. Három mélység esetén 2048000 példány. Ekkor csak a bábuk példányait számoltuk, pedig ezenkívül a játékfa előállításakor mást is kell példányosítani, de ezen is nagyon jól látszik az exponenciális növekedés. Ez a program JAVA nyelven íródott, és tudjuk, hogy ebben a nyelvben a példányosítás eléggé időigényes művelet. Látszik, hogy nagyon ügyesen kell megválasztani a mélységet, mert ettől függ a program gyorsasága és okossága is, melyek egymással fordítottan arányosak. Felmerülhet a kérdés, hogy muszáj ennyit példányosítani? Nem lehetne valahogy ezen csökkenteni? Nem lehetne valahogy azt megoldani, hogy ne kelljen minden állást előállítani? Ezekre a kérdésre a válasz az alfa-béta vágás. Az alfa-béta vágást[10] a minimax algoritmusra lehet alkalmazni, nem a sakkjáték sajátosságait használja föl, nem ezek ismeretével végzi a vágást, hanem a heurisztika segítségével. Itt is látszik, mennyire fontos a jó heurisztika. Lehetnek olyan játékfák,

[10]Dombi József, Feteke István, Gyimóthy Tibor, Sántáné Tóth Edit: Mesterséges intelligencia, Budapest, 1999, ISBN: 963-9078-99-9

melyekben az alfa-béta vágásnak köszönhetően nem kell minden játékállást előállítani, mert a játékfa azon részletétől független a lépésajánlat.




*alfa-béta vágás*

A példán kettő mélység lett előállítva a fából. „A" játékosnak szeretnénk lépést ajánlani. A fa gyökerének három gyereke van, ezek közül először a „B" jelzésű elemet kezeljük. Ennek a gyerekei „E", „F", „G", mivel ezek a fa levélelemei, ezért ezeknek az elemeknek kiszámoljuk a heurisztikáját. A „B" elemnek a heurisztikáját nem a heurisztikus függvénnyel számoljuk, hanem mivel „B" szinten található a fában, ezért a gyerekeinek értékei közül a legkisebbet vesszük. Tehát a „B" elem heurisztikája 6752. Ezután a „C" elemet dolgozzuk fel. A „H" gyermekével kezdünk, kiszámoljuk a heurisztikát: 7852. Mivel „C" heurisztikájánál minimalizálunk, ezért ha ez az elem lenne a legkisebb, akkor ezt kapná „C". Kiszámoljuk „I" heurisztikáját -1040. Tudjuk, hogy „C" értéke a gyerekek értékének a minimuma lesz, tehát biztos, hogy –1040-től nem lesz nagyobb „C" heurisztikája. „A" elemnek az értékét maximalizálással számoljuk, és „B" értéke 6752, „C" értéke nem nagyobb, mint –1040, ezért felesleges „C" további gyerekeit előállítani, mert azok heurisztikus értékétől független „A" értéke. Tehát „J"-t nem kell előállítani, nem kell kiszámolni a heurisztikáját, a fában vágás történik. „C" feldolgozása után „D" következik, gyermekei közül a „K" elemnek kiszámoljuk a heurisztikáját: 5200. Tudjuk, hogy „D" elem „B" játékos szintjén helyezkedik el, tehát

minimalizálással választunk heurisztikát, „D” értéke 5200-nál nagyobb nem lehet, vagyis „A” heurisztikájának nem „D” értéke lesz választva, mert van vele azonos szinten nagyobb értékű elem. Tehát „D” további gyermekeit nem kell előállítani, vágás történik. „A” heurisztikus értéke 6752 lesz, és „B” gyermekével reprezentált állásba való lépés lesz javasolva. A példán a 13 elemből 3 elemet kivágtunk, ennyivel kevesebb példányosítás történne.
A minimax algoritmus a `Minimax` osztályban van megvalósítva.

```
    Sakk s;
    int maxmelyseg=2;
    int limitmax=20000;  //nem használt limit
    Operator javasoltop;

Operator kereses(Allapot a){
     for(int i=0;i<s.operatorok.size();i++)
      ((Operator)s.operatorok.get(i)).h=100000;
       javasoltop=null;
     kiterjeszt(a,0,20000);
     return javasoltop;}

     int kiterjeszt(Allapot a, int melyseg,int limit){
      Operator o;
      Operator o2;
      if(melyseg==maxmelyseg||a.celfelt()){return a.h();}
      int minimax;
      int aminimax; //olyan érték mely az állapotnak a legrosszabb lenne
      if(a.kijon.equals("A"))aminimax=-15000;
      else aminimax=15000;
      minimax=aminimax;
      for(int i=0;i<s.operatorok.size();i++){
       o2=(Operator)s.operatorok.get(i);
       o=new Operator(o2.nx,o2.ny,o2.ax,o2.ay);
       if (o.elofelt(a))
        {
         Allapot uj=o.alkalmaz(a);
         if (minimax==aminimax)limitmax=20000;else limitmax=minimax;
         int h=kiterjeszt(uj,melyseg+1,limitmax);
         if (a.kijon.equals("A"))minimax=Math.max(minimax,h);
         else minimax=Math.min(minimax,h);
```

```
      if(limit!=20000)
        {if(a.kijon.equals("A")&&limit<minimax)return minimax;
         if(a.kijon.equals("B")&&limit>minimax)return minimax;}
      if (h==minimax&&melyseg==0){o.h=h;javasoltop=o;a.ao.add(o);}
    }
    }
  return minimax;
 }
```

A minimax algoritmus által javasolt operátort a `kereses` függvény adja meg, aminek átadjuk az adott állapotot paraméterként. Majd meghívjuk a `kiterjeszt` függvényt, ami az adott állapotot egy fa elemeként kezelve előállítja annak gyermekeit, ha szükséges. Ezt a függvényt rekurzívan fogjuk meghívni. A függvénynek 3 paramétere van: az adott állapot, a mélység, amely azt jelzi, hogy a fában hányadik szinten található az állapot, ezzel állítható meg a rekurzió, a harmadik paraméter egy limit, melyet az alfa-béta vágáshoz használunk. A függvényben megvizsgáljuk, hogy az adott állapot hányadik szinten van, illetve, hogy ez az állapot célállapot-e, vagyis olyan állapot, melyben a játék végálláshoz érkezett, tehát nem lehet neki a fában gyereke. Ha a szint megegyezik a maximummélységgel, vagy célállapot, akkor az adott állapotnak a heurisztikáját nem a minimax algoritmussal, hanem a heurisztikus függvénnyel számoljuk ki. Létrehozunk egy `aminimax` nevű változót, amely azt a célt szolgálja, hogy egy olyan kezdőértéket adjunk a `minimax` változónak, mely 'A' játékos esetén olyan kicsi, 'B' játékos esetén pedig olyan nagy, amit egyébként nem tudna felvenni. A `minimax` megkapja `aminimax` értékét. Ezután az alkalmazható operátorok segítségével előállítjuk az adott állapot egy gyerekét, megvizsgáljuk a `minimax` értéket, és ennek megfelelően vagy a `limitmax`-szal vagy a `minimax` értékével hívjuk meg az állapotra a `kiterjeszt` függvényt. A függvény által visszaadott érték és `minimax` értéke közül 'A' játékos esetén a nagyobb, 'B' játékos esetén a kisebb lesz `minimax` értéke. Majd megvizsgálja `limit` értékét, ha ez nem egyenlő a kezdőértékével, vagyis a `kiterjeszt` függvény egy adott heurisztikus értékkel lett meghívva. Tehát például, ha 'B' játékos szintjéről hívtuk meg a kiterjeszt függvényt egy konkrét `limit` értékkel, és az egyik gyereke a `limit`-nél kisebb értékkel tér vissza, akkor a többi gyereket nem állítjuk elő, mert ez az állapot ('B' szintjén van) minimalizálva fog választani a értékek közül, vagyis a `limit`től kisebb értékkel tér vissza a `kiterjeszt` függvénye, és ennek a szülő állapota 'A' szinten

van, tehát maximalizálni fogunk, ezért a vizsgált elem többi gyerekét felesleges előállítani, mivel azoktól független lesz a lépésajánlat. Tehát 'A' játékos esetén `limit<minimax` , 'B' játékos esetén pedig `limit>minimax` vizsgálattal vágunk (alfa-béta vágás).

Ha `h` értéke egyenlő `minimax` értékével, és 0. szinten vagyunk, akkor az adott `operator`-t beállítjuk `javasolt operator`-nak, és az állapot egy `ao` vektorába rakja a `javasolt operator`-t, miután beállította az operátor heurisztikus értékét, amelyet az operátor hatásaként előállt állapot heurisztikus értékéből származtatunk. Azért kell vektorban összegyűjteni a javasolt operátorokat (az algoritmus alatt több javasolt operátor lehet, lehet olyan is, mely az algoritmus későbbi szakaszában már nem lehet javasolt operátor, ezért kell a heurisztikát is letárolni), hogy több azonos heurisztikájú javasolt operátor esetén véletlenszerűen tudjunk választani.

Végül `minimax` értékét adjuk visszatérési értékként.

## Néhány főbb osztály ismertetése

Itt néhány olyan osztályt szeretnék ismertetni, melyek a programnak a fő vázát képezik. Ez az ismertetés egyrészt a jobb átláthatóságot szolgálja, másrészt pedig a program továbbfejlesztésénél lehet hasznos, hogy a program főbb, szignifikánsabb részei hol találhatók.

Ezeknek az osztályoknak adattagjai, metódusai már részben voltak említve, illetve azokra a részekre kívánok kitérni, melyek a program szempontjából fontos feladatot végeznek, nem ismertetem minden osztály minden adattagját, metódusát.

`Operator` osztály: ez az osztály reprezentálja a lépést, mely állapotból állapotba visz. Az operátorok száma véges (minden mezőből minden mezőbe, de szűkíthető). Adattagok:

`boolean sakk=false;` Operátor hatásaként előálló állapot sakk-e

`boolean matt=false;` Operátor hatásaként előálló állapot matt-e

`boolean patt=false;` Operátor hatásaként előálló állapot patt-e

`int h=0;` Operátor hatásaként előálló állapot heurisztikája

`boolean csere=false;` Operátornak következménye-e parasztcsere

`boolean anp=false;` Operátor en passant lépés-e

`boolean sanc=false;` Operátor sánc lépés-e

Ezek az adattagok az állapottól függnek.

Metódusok:

`boolean elofelt(Allapot a){…}` előfeltétel függvény, mely egy operátor egy állapotra való alkalmazhatóságát vizsgálja

`public Allapot alkalmaz(Allapot a){…}`alkalmaz függvény, mely egy operátort egy állapotra alkalmazva előálló állapotot adja vissza

`Babu` osztály: A hatfajta bábutípus szülő osztálya. Rendelkezik azokkal a metódusokkal, amelyekkel az alosztályok is, csak itt nincsen definiálva a törzsük. De ennek a módszernek az a nagy előnye, hogy nem kell tudnom, hogy az adott mezőn milyen bábu található, meg tudok rá hívni pl. egy előfeltétel függvényt, vagy le tudom kérdezni a nevét a bábunak. Adattagok:

`boolean lm;` az adott bábu lépett már? A sáncvizsgálatnál van fontos szerepe.

`int ertek;` az adott bábu heurisztikus értéke

`String kie;` az adott bábu melyik játékoshoz tartozik { A , B }

Metódusok:

`public boolean elofelt(Allapot a,Operator l){…}` Adott állapotra az adott operátor alkalmazható-e. Annak a bábunak az előfeltétele fog meghívódni, mely a kiindulási mezőn található, mivel más típusú bábunak más lépés megengedett.

`public void uthet(Allapot a,int nx,int ny,int [][] u){…}` A mártixba (u) jelzi egyesekkel, hogy az adott állapotban (a) az adott pozícióról (x,y) mik a lehetséges lépéshelyek.

`public boolean neve(String s){…}` A bábu típusa állapítható meg.

<u>`Allapot` osztály:</u> Egy adott állást reprezentáló osztály. Adattagok:

`Vector ao=new Vector();` Adott állapotra alkalmazható operátorok

`Allapot r;` amelyik állapotból l lépésen keresztül ide jutott

`Operator l=new Operator();` amely lépésen keresztül idejutott

`String kijon;` ki következik lépni {A,B}

`Babu[][] t=new Babu [8][8];` bábu mátrix a bábuk tárolására

Metódusok:

`boolean celfelt(){…}`Minimax algoritmushoz szükséges célfeltétel.

`int h(){…}` Az adott állapot heurisztikus értékét visszaadó függvény

`public boolean matt(){…}` Adott állás matt-e?

`public boolean sakk(int [][] u){…}` Adott állás sakk-e?

<u>`Jatszma` osztály</u>: Egy játszmát reprezentáló osztály. Adattagok:

`String event="";` A játszma EVENT mezője

`String site="";` A játszma SITE mezője

`String date="";` A játszma DATE mezője

`String round="";` A játszma ROUND mezője

`String white="";` A játszma WHITEmezője

`String black="";` A játszma BLACK mezője

`String result="";` A játszma RESULT mezője

`String eco="";` A játszma ECO mezője

`String setup="";` A játszma SetUp mezője

`String fen="";` A játszma FEN mezője

`String jatsz;` A játszma lépéseit tartalmazza

Metódus:

`public void atalakit(String s) {…}` Egy adott sztringet feldolgoz, átalakít egy `Jatszma` példánynak.

`Minimax` osztály: Minimax algoritmus osztálya. Adattagok:

`int maxmelyseg=2;` fa előállítása az adott mélységig

`int limitmax=20000;` egy nem használt limit

`Operator javasoltop;` javasolt lépés

Metódusok:

`Operator kereses(Allapot a){…}` Adott állapothoz lépést ajánl

`int kiterjeszt(Allapot a, int melyseg,int limit){…}` Adott állapotot faelemként kezelve kiterjeszti. Ez egy rekurzívan meghívott függvény.

# ÖSSZEGZÉS

A dolgozatnak két fő célja volt: egyrészt maga a program bemutatása és a főbb részek JAVA nyelven történt megvalósításának áttekintése. A program maga a játék, a sakkjáték elég bonyolult, és elég nehéz játék, célok között szerepelt az is, hogy kezdő, vagy sakkot nem ismerő játékosoknak segítsen elsajátítani a sakkjáték rejtelmeit. Illetve, hogy érdekes, szórakoztató, és elég tudással rendelkezzen ahhoz, hogy akár egy ember is tudja játszani a kétszemélyes játékot! Ezt sikerült megvalósítani, és természetesen maradtak továbbfejlesztési lehetőségek. A játék könnyen kezelhetőségére is nagy hangsúlyt helyeztem, igyekeztem minden üzenetet könnyen értelmezhetően megírni, hogy segítség lehessen a felhasználónak. A grafikus megoldások felhasználó-baráttá tették a programot. A JAVA nyelv előnyei között szerepel a jól használható API, melyre többször is hivatkoztam, és a programírásánál is nagy segítség volt. El kell ismerni, hogy a JAVA nyelv hátrányaként emlegetett lassúság, mely a példányosításból fakad, ebben a program is bebizonyosodott a lépésajánló algoritmusnál. Természetesen lehet tovább optimalizálni a programot, olyan módosításokat alkalmazni, melyek inkább fordításkor, mint futtatáskor igényelnek több időt. A program továbbfejlesztési lehetőségei között említeném az olyan dolgokat, melyeket nem tudtam megvalósítani, de hasznos kiegészítései lehetnek a programnak:

- A heurisztika továbbfejlesztése, játéknyitási minták, melyekkel a program komolyabb ellenfél lehet.
- A lépésajánló algoritmus erősségének állíthatósága.
- A grafikus dolgok a program nagyon kis változtatásával változtathatóak pl.: a bábuk képei külön fájlban találhatók. Különböző stílusok választhatósága, sajátok készítése.
- A vakok, gyengén látók számára hangkiegészítés, mely az adott lépést vagy az adott állást felolvasná.
- A játékot nem ismerők, vagy kevésbé ismerők számára lehetséges lépések grafikus bemutatása akár egy adott állásban is.
- A „fogott bábú lép” elv tiszteletben tartása, ami nagy hiányossága a programnak. Ha egy bábut „megfogunk”, és annak létezik szabályos lépése az adott állapotban, akkor ezen lépések közül kelljen választani.

Remélem, sikerült olyan programot készítenem, amely sok sakkot szerető ember számára ad lehetőséget a kellemes és hasznos időtöltésre!

# IRODALOMJEGYZÉK


1. Nyékyné Gaizler Judit: Java2 útikalauz programozóknak 1.3 I.,II. kötet, ELTE TTK Hallgatói Alapítvány, Budapest, 2000, ISBN: 963-463-364-1
2. Dr. Szalay Tibor: A mesterséges intelligencia alapjai, Gábor Dénes Főiskola, Budapest, 2002
3. Dombi József, Feteke István, Gyimóthy Tibor, Sántáné Tóth Edit: Mesterséges intelligencia, Budapest, 1999, ISBN: 963-9078-99-9
4. Kovács P. Attila: Sakkprogramozásról mindenkinek, Novotrade Rt., Budapest 1987, ISBN: 963-024-691-0
5. http://java.sun.com/j2se/1.5.0/docs/api/
6. http://hu.wikipedia.org/wiki/A_sakkjatszmak_lejegyzese
7. http://geocities.com/CapeCanaveral/Launchpad/2640/pgn/pgn_spec/pgn_lidx.htm
8. http://observer.homestead.com/openings.html
9. http://www.jatek.hu/szabalyok/hlp_sakk.html