Debreceni Egyetem
Informatika Kar

KÉTSZEMÉLYES JÁTÉKOK

Készítette:
Simay János
Programtervező matematikus

Témavezető:
Mecsei Zoltán
Egyetemi tanársegéd

Debrecen
2009

Tartalomjegyzék

1. Bevezetés

Elöljáróban szeretnék néhány szót szólni a témaválasztást és a választott területet illetően. Már gyerekkoromban érdeklődtem a stratégiai játékok iránt, így nem volt kérdés, hogy „Kétszemélyes játékok” címmel fogom írni diplomamunkámat. Az egyetemi évek alatt lehetőségem volt bepillantást nyerni a mesterséges intelligencia világába, így sikerült közelebb kerülnöm az érdekelt területhez. A kétszemélyes játékok gyűjtőnevén belül számos játék ismert, melyek közül egy megfelelően bonyolultat választottam vizsgálódásom tárgyának, a sakkot.

A sakknak különféle változatai ismertek, melyek közül a továbbiakban az Európában legnépszerűbb modern sakkot vesszük alapul, melynek szabályai a következő web címen olvashatók : http://hu.wikipedia.org/wiki/Sakk.

Az elemzés során négy meghatározó egységben fogom górcső alá venni a sakkot, mint mesterséges intelligenciai vizsgálódásom tárgyát. Először egy lehetséges állapottér reprezentációt adok a játék reprezentálására, majd gépi játékos készítéséhez alkalmazható lépésajánló mehanizmusokat mutatok be. Ezt követően a játék heurisztikájának megkonstruálásához használható meggondolásokba és szempontokba adok betekintést a teljesség igénye nélkül. A heurisztika tulajdonképpen egy adott állást adott játékos szempontjából számszerűsítő algoritmus. Végül pedig az egzakt java implementációmat érintő legfontosabb tudnivalókat ismertetem.

A sakk bonyolultsága szükségessé tette, hogy az állapottér reprezentáció tárgyalása során bevezessek néhány logikai értéket visszaadó függvényt a lépések szabályosságának vizsgálatához a „véget nem érő” logikai kifejezések elkerülése végett. Ezen függvényeket egy általánosan értelmezhető pszeudo nyelven írtam meg. A sakk heurisztikájának megkonstruálásához meggondolások és szempontok tömkelege áll rendelkezésre, azonban teljesítménybeli okokból ezek együttes figyelembevétele nem ajánlott, mivel így a lépésajánló algoritmus képtelen lesz nagyobb mélységekben kivárható időben kalkulálni.

2. Állapottér reprezentáció

2.1. Alkalmazott jelölések

A továbbiakban a következő megfeleltetéseket használom:

V - univerzális kvantor, E - egzisztenciális kvantor

! - negáció

&& - konjunkció, || - diszjunkció

>> - implikáció

null - üreshalmaz

2.2. Kezdőállapot

A sakk állapottér reprezentációjának leírását a kezdő(k) állapot megadásával és magyarázatával kezdem:

T (a sakktáblát reprezentáló mátrix)

13	14	15	12	11	15	14	13
16	16	16	16	16	16	16	16
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
6	6	6	6	6	6	6	6
3	4	5	2	1	5	4	3

L_lista = null (az eddig megtett lépéseket tároló lista)

L_idx = 0 (azt az L_lista indexet jelzi,ahová a következő lépést be kell szúrni)

Köv = személy (a lépni következő játékost mutatja)

V_s_k = TRUE (világos sáncolhat -e király szárnyon / kezdetben igen /)

V_s_v = TRUE (világos sáncolhat -e vezér szárnyon / kezdetben igen /)

S_s_k = TRUE (sötét sáncolhat -e király szárnyon / kezdetben igen /)

S_s_v = TRUE (sötét sáncolhat -e vezér szárnyon / kezdetben igen /)

2.3. Alkalmazandó függvények

Az állapottér reprezentáció átláthatósága érdekében be kell vezetnünk különböző vizsgálatokat végző, logikai értéket visszaadó, pszeudo kódú függvényeket, melyekre a későbbiekben hivatkozni fogunk.

2.3.1. Támadott függvény

Az első függvény adott játékos és pozíció esetén meghatározza, hogy a pozíció támadva van e a paraméterként átadott játékos ellenfele által:

```
function boolean támadott ( T,játékos,pos_s,pos_o ){
   // a huszár mozgását reprezentáló mátrix
   H = {-2,-2,-1, 1, 2, 2, 1,-1
          -1, 1, 2, 2, 1,-1,-2,-2}
   // a vezér és a futó mozgását reprezentáló mátrix
   VF = {-1,-1, 1, 1
            -1, 1, 1,-1}
   // a vezér és a bástya mozgását reprezentáló mátrix
   VB = {-1, 0, 1, 0
              0, 1, 0,-1}
   // a világos gyalog köördináta növekményei
   GY = { 1, 1
             -1, 1}
   // a világos oldal támadását figyeljük
   offset = 0
   if ( játékos == személy ){
    // a sötét gyalog koordináta növekményei
    GY[0,0] = -1
    GY[0,1] = -1
```

```
  GY[1,0] = -1
  GY[1,1] =  1
  // a sötét oldal támadását figyeljük
  offset = 10
}
// támadja e gyalog az adott mezőt
for ( j = 0;j <= 1;j++ ){
   if ( pos_s + P[0,j] >= 0 && pos_s + P[0,j] <= 7
   && pos_o + P[1,j] >= 0 && pos_o + P[1,j] <= 7 )
        if ( T[pos_s + P[0,j],pos_o + P[1,j]] == 6 + offset ) return TRUE
}
// támadja e huszár az adott mezőt
for ( j = 0;j <= 7;j++ ){
   if ( pos_s + H[0,j] >= 0 && pos_s + H[0,j] <= 7
   && pos_o + H[1,j] >= 0 && pos_o + H[1,j] <= 7 )
        if ( T[pos_s + H[0,j],pos_o + H[1,j]] == 4 + offset ) retrun TRUE
}
// támadja e futó vagy vezér futó vonalon az adott mezőt
i = 1
j = 0
while ( j <= 3 ){
   if ( pos_s + ( i * VF[0,j] ) >= 0 && pos_s + ( i * VF[0,j] ) <= 7
   && pos_o + ( i * VF[1,j] ) >= 0 && pos_o + ( i * VF[1,j] ) <= 7 ){
       if ( T[pos_s + ( i * VF[0,j] ),pos_o + ( i * VF[1,j] )] == 5 + offset
       || T[pos_s + ( i * VF[0,j] ),pos_o + ( i * VF[1,j] )] == 2 + offset )
          return TRUE
       if (T[pos_s + ( i * VF[0,j] ),pos_o + ( i * VF[1,j] )] != 0 ){
          i = 1
          j++
       }else i++
   }else{
      i = 1
```

```
            j++
        }
    }
    // támadja e bástya vagy vezér bástya vonalon az adott mezőt
    i = 1
    j = 0
    while ( j <= 3 ){
        if ( pos_s + ( i * VB[0,j] ) >= 0 && pos_s + ( i * VB[0,j] ) <= 7
        && pos_o + ( i * VB[1,j] ) >= 0 && pos_o + ( i * VB[1,j] ) <= 7) {
            if ( T[pos_s + ( i * VB[0,j] ),pos_o + ( i * VB[1,j] )] == 3 + offset
            || T[pos_s + ( i * VB[0,j] ),pos_o + ( i * VB[1,j] )] == 2 + offset)
              return TRUE
            if ( T[pos_s + ( i * VB[0,j] ),pos_o + ( i * VB[1,j] )] != 0 ){
              i = 1
              j++
            }else i++
        }else{
            i = 1
            j++
        }
    }
    return FALSE
}
```

2.3.2. Sakk függvény

A következő függvény adott játékos esetén meghatározza, hogy sakkban van e:

```
function boolean sakk ( T,játékos ){
    if ( játékos == gép ) offset = 0
    else offset = 10
    for ( i = 0;i <= 7;i++ )
```

```
      for ( j = 0;j <= 7;j++ )
         if ( T[i,j] == 11 - offset ){
           if ( tamadott( T,játékos,i,j ) ) return TRUE
           else return FALSE
         }
}
```

2.3.3. Léphető függvény

Az utolsó bevezetésre kerülő függvény megmutatja, hogy egy adott játékos az adott lépést meglépheti e:

```
function boolean léphető ( T,játékos,pos_s1,pos_o1,pos_s2,pos_o2){
   if ( játékos == gép ) offset = 0
   else offset = 10
   en_passant = FALSE

    if ( abs(pos_s1 - pos_s2) == 1 && abs(pos_o1 - pos_o2) == 1
    && T[pos_s1,pos_o1] == (16 - offset) && T[pos_s2,pos_o2] == 0 ){
        T[pos_s1,pos_o2] = 0
        en_passant = TRUE
    }
    érték = T[pos_s2,pos_o2]
    T[pos_s2,pos_o2] = T[pos_s1,pos_o1]
    T[pos_s1,pos_o1] = 0
    bool = sakk( T,játékos )
    if ( en_passant == TRUE ) T[pos_s1,pos_o2] = 6 + offset
    T[pos_s1,pos_o1] = T[pos_s2,pos_o2]
    T[pos_s2,pos_o2] = érték
    return !bool
}
```

2.4. Operátorok

A következőkben sorra vesszük az állapotváltozásokat leíró operátorokat.

2.4.1. Gyalog lép egyet :

alkalmazási előfeltétel:

((köv == személy) && EiEj(T[i,j] == 6 && T[i-1,j] == 0 &&
léphető(T,személy,i,j,i-1,j))) ||
((köv == gép) && EiEj(T[i,j] == 16 && T[i+1,j] == 0 &&
léphető(T,gép,i,j,i+1,j)))

alkalmazás hatása:

ha köv == személy és az [s,o] pozíción álló gyalogra teljesül a feltétel:

$$T'[i,j] = \begin{cases} 0, \text{ ha } i == s \quad \&\& \; j == o \\ 6, \text{ ha } i == s - 1 \;\&\&\; j == o \;\&\&\; s \mathrel{!}= 1 \\ 2, \text{ ha } i == s - 1 \;\&\&\; j == o \;\&\&\; s == 1 \\ T[i,j], \text{ egyébként} \end{cases}$$

L_lista' = L_lista

L_lista'(L_idx) = (s,o,s-1,o)

L_idx' = L_idx + 1

köv' = gép

V_s_k' = V_s_k

V_s_v' = V_s_v

S_s_k' = S_s_k

S_s_v' = S_s_v

ha köv == gép és az [s,o] pozíción álló gyalogra teljesül a feltétel:

$$T'[i,j] = \begin{cases} 0, \text{ ha } i == s \quad \&\& \; j == o \\ 16, \text{ ha } i == s + 1 \;\&\&\; j == o \;\&\&\; s \mathrel{!}= 6 \\ 12, \text{ ha } i == s + 1 \;\&\&\; j == o \;\&\&\; s == 6 \\ T[i,j], \text{ egyébként} \end{cases}$$

L_lista’ = L_lista

L_lista’(L_idx) = (s,o,s+1,o)

L_idx’ = L_idx + 1

köv’ = személy

V_s_k’ = V_s_k

V_s_v’ = V_s_v

S_s_k’ = S_s_k

S_s_v’ = S_s_v

2.4.2. Gyalog lép kettőt :

alkalmazási előfeltétel:

((köv == személy) && EiEj(i == 6 && T[i,j] == 6 && T[i-1,j] == 0 &&
T[i-2,j] == 0 && léphető(T,személy,i,j,i-2,j))) ||
((köv == gép) && EiEj(i == 1 && T[i,j] == 16 && T[i+1,j] == 0 &&
T[i+2,j] == 0 && léphető(T,gép,i,j,i+2,j)))

alkalmazás hatása:

ha köv == személy és az [s,o] pozíción álló gyalogra teljesül a feltétel:

$$T'[i,j] = \begin{cases} 0, \text{ ha } i == s \quad \&\& \; j == o \\ 6, \text{ ha } i == s - 2 \;\&\&\; j == o \\ T[i,j], \text{ egyébként} \end{cases}$$

L_lista’ = L_lista

L_lista’(L_idx) = (s,o,s-2,o)

L_idx’ = L_idx + 1

köv’ = gép

V_s_k’ = V_s_k

V_s_v’ = V_s_v

S_s_k’ = S_s_k

S_s_v’ = S_s_v

ha köv == gép és az [s,o] pozíción álló gyalogra teljesül a feltétel:

$$T'[i,j] = \begin{cases} 0, \text{ ha } i == s \quad \&\& \; j == o \\ 16, \text{ ha } i == s + 2 \;\&\&\; j == o \\ T[i,j], \text{ egyébként} \end{cases}$$

L_lista' = L_lista

L_lista'(L_idx) = (s,o,s+2,o)

L_idx' = L_idx + 1

köv' = személy

V_s_k' = V_s_k

V_s_v' = V_s_v

S_s_k' = S_s_k

S_s_v' = S_s_v

2.4.3. Gyalog lép enpassant :

alkalmazási előfeltétel:

((köv == személy) && EiEj(i == 3 && T[i,j] == 6 &&
((T[i-1,j-1] == 0 && T[i,j-1] == 16 && L_lista(L_idx-1) == (i-2,j-1,i,j-1) &&
léphető(T,személy,i,j,i-1,j-1)) || (T[i-1,j+1] == 0 && T[i,j+1] == 16 &&
L_lista(L_idx-1) == (i-2,j+1,i,j+1) && léphető(T,személy,i,j,i-1,j+1))))) ||
((köv == gép) && EiEj(i == 4 && T[i,j] == 16 &&
((T[i+1,j-1] == 0 && T[i,j-1] == 6 && L_lista(L_idx-1) == (i+2,j-1,i,j-1) &&
léphető(T,gép,i,j,i+1,j-1)) || (T[i+1,j+1] == 0 && T[i,j+1] == 6 &&
L_lista(L_idx-1) == (i+2,j+1,i,j+1) && léphető(T,gép,i,j,i+1,j+1)))))

alkalmazás hatása:

ha köv == személy és az [s,o] pozíción álló gyalogra teljesül a feltétel:

$$T'[i,j] = \begin{cases} 0, \text{ ha } i == s \;\&\&\; j == o \\ 0, \text{ ha L\_lista(L\_idx-1) == (s-2,o-1,s,o-1) \&\& } i == s \;\&\&\; j == o - 1 \\ 0, \text{ ha L\_lista(L\_idx-1) == (s-2,o+1,s,o+1) \&\& } i == s \;\&\&\; j == o + 1 \\ 6, \text{ ha L\_lista(L\_idx-1) == (s-2,o-1,s,o-1) \&\& } i == s - 1 \;\&\&\; j == o - 1 \\ 6, \text{ ha L\_lista(L\_idx-1) == (s-2,o+1,s,o+1) \&\& } i == s - 1 \;\&\&\; j == o + 1 \\ T[i,j], \text{ egyébként} \end{cases}$$

L_lista’ = L_lista

L_lista’(L_idx) = (enpass,s,o)

L_idx’ = L_idx + 1

köv’ = gép

V_s_k’ = V_s_k

V_s_v’ = V_s_v

S_s_k’ = S_s_k

S_s_v’ = S_s_v

ha köv == gép és az [s,o] pozíción álló gyalogra teljesül a feltétel:

$$T'[i,j] = \begin{cases} 0, \text{ ha } i == s \;\&\&\; j == o \\ 0, \text{ ha } L\_lista(L\_idx-1) == (s+2,o-1,s,o-1) \;\&\&\; i == s \;\&\&\; j == o - 1 \\ 0, \text{ ha } L\_lista(L\_idx-1) == (s+2,o+1,s,o+1) \;\&\&\; i == s \;\&\&\; j == o + 1 \\ 16, \text{ ha } L\_lista(L\_idx-1) == (s+2,o-1,s,o-1) \;\&\&\; i == s + 1 \;\&\&\; j == o - 1 \\ 16, \text{ ha } L\_lista(L\_idx-1) == (s+2,o+1,s,o+1) \;\&\&\; i == s + 1 \;\&\&\; j == o + 1 \\ T[i,j], \text{ egyébként} \end{cases}$$

L_lista’ = L_lista

L_lista’(L_idx) = (enpass,s,o)

L_idx’ = L_idx + 1

köv’ = személy

V_s_k’ = V_s_k

V_s_v’ = V_s_v

S_s_k’ = S_s_k

S_s_v’ = S_s_v

2.4.4. Gyalog üt(x) :

// „x” a „-1”-et kapja értékül ha vezérszárnyi ütésről van szó

// „x” az „1”-et kapja értékül ha királyszárnyi ütésről van szó

alkalmazási előfeltétel:

((köv == személy) && EiEj(T[i,j] == 6 && T[i-1,j+x] > 6 &&
léphető(T,személy,i,j,i-1,j+x))) ||
((köv == gép) && EiEj(T[i,j] == 16 && T[i+1,j+x] <= 6 && T[i+1,j+x] != 0 &&
léphető(T,gép,i,j,i+1,j+x)))

alkalmazás hatása:

ha köv == személy és az [s,o] pozíción álló gyalogra teljesül a feltétel:

$$T'[i,j] = \begin{cases} 0, \text{ ha } i == s \;\&\&\; j == o \\ 6, \text{ ha } i == s - 1 \;\&\&\; j == o + x \;\&\&\; s != 1 \\ 2, \text{ ha } i == s - 1 \;\&\&\; j == o + x \;\&\&\; s == 1 \\ T[i,j], \text{ egyébként} \end{cases}$$

L_lista' = L_lista

L_lista'(L_idx) = (s,o,s-1,o+x)

L_idx' = L_idx + 1

köv' = gép

V_s_k' = V_s_k

V_s_v' = V_s_v

S_s_k' = S_s_k

S_s_v' = S_s_v

ha köv == gép és az [s,o] pozíción álló gyalogra teljesül a feltétel:

$$T'[i,j] = \begin{cases} 0, \text{ ha } i == s \;\&\&\; j == o \\ 16, \text{ ha } i == s + 1 \;\&\&\; j == o + x \;\&\&\; s != 6 \\ 12, \text{ ha } i == s + 1 \;\&\&\; j == o + x \;\&\&\; s == 6 \\ T[i,j], \text{ egyébként} \end{cases}$$

L_lista' = L_lista

L_lista'(L_idx) = (s,o,s+1,o+x)

L_idx' = L_idx + 1

köv' = személy

V_s_k' = V_s_k

V_s_v' = V_s_v

S_s_k' = S_s_k

S_s_v’ = S_s_v

2.4.5. Huszár lép(x) :

// „x” értéke a huszár lépésirányának az alábbi ábra alapján megfeleltetett szám:

	0		1	
7				2
		H		
6				3
	5		4	

// az irányokhoz tartozó koordináta növekmények mátrixa:

H = {-2,-2,-1, 1, 2, 2, 1,-1
-1, 1, 2, 2, 1,-1,-2,-2}

alkalmazási előfeltétel:

((köv == személy) && EiEj(T[i,j] == 4 && (T[i+H[0,x],j+H[1,x]] == 0 ||
T[i+H[0,x],j+H[1,x]] > 6) && léphető(T,személy,i,j,i+H[0,x],j+H[1,x]))) ||
((köv == gép) && EiEj(T[i,j] == 14 && T[i+H[0,x],j+H[1,x]] <= 6 &&
léphető(T,gép,i,j,i+H[0,x],j+H[1,x])))

alkalmazás hatása:

ha köv == személy és az [s,o] pozíción álló huszárra teljesül a feltétel:

$$T'[i,j] = \begin{cases} 0, \text{ ha } i == s \;\&\&\; j == o \\ 4, \text{ ha } i == s + H[0,x] \;\&\&\; j == o + H[1,x] \\ T[i,j], \text{ egyébként} \end{cases}$$

L_lista’ = L_lista

L_lista’(L_idx) = (s,o,s+H[0,x],o+H[1,x])

L_idx’ = L_idx + 1

köv’ = gép

V_s_k’ = V_s_k

V_s_v’ = V_s_v

S_s_k’ = S_s_k

S_s_v’ = S_s_v

ha köv == gép és az [s,o] pozíción álló huszárra teljesül a feltétel:

$$T'[i,j] = \begin{cases} 0, \text{ ha } i == s \;\&\&\; j == o \\ 14, \text{ ha } i == s + H[0,x] \;\&\&\; j == o + H[1,x] \\ T[i,j], \text{ egyébként} \end{cases}$$

L_lista’ = L_lista

L_lista’(L_idx) = (s,o,s+H[0,x],o+H[1,x])

L_idx’ = L_idx + 1

köv’ = személy

V_s_k’ = V_s_k

V_s_v’ = V_s_v

S_s_k’ = S_s_k

S_s_v’ = S_s_v

2.4.6. Futó lép(x,y) :

// „x” értéke a futó lépésirányának az alábbi ábra alapján megfeleltetett szám:

0		1
	F	
3		2

// az irányokhoz tartozó koordináta növekmények mátrixa:

F = {-1,-1, 1, 1

-1, 1, 1,-1}

// „y” a lépés mértékét jelöli [1..7]

alkalmazási előfeltétel:

((köv == személy) && EiEj(T[i,j] == 5 && (T[i+y*F[0,x],j+y*F[1,x]] == 0 ||

T[i+y*F[0,x],j+y*F[1,x]] > 6) &&

Vk((k > 0 && k < y) >> (T[i+k*F[0,x],j+k*F[1,x]] == 0)) &&
léphető(T,személy,i,j, i+y*F[0,x], j+y*F[1,x]))) ||
((köv == gép) && EiEj(T[i,j] == 15 && T[i+y*F[0,x],j+y*F[1,x]] <= 6 &&
Vk((k > 0 && k < y) >> (T[i+k*F[0,x],j+k*F[1,x]] == 0)) &&
léphető(T,gép,i,j, i+y*F[0,x], j+y*F[1,x])))

alkalmazás hatása:

ha köv == személy és az [s,o] pozíción álló futóra teljesül a feltétel:

$$T'[i,j] = \begin{cases} 0, \text{ ha } i == s \;\&\&\; j == o \\ 5, \text{ ha } i == s + y * F[0,x] \;\&\&\; j == o + y * F[1,x] \\ T[i,j], \text{ egyébként} \end{cases}$$

L_lista' = L_lista

L_lista'(L_idx) = (s,o,s+y*F[0,x],o+y*F[1,x])

L_idx' = L_idx + 1

köv' = gép

V_s_k' = V_s_k

V_s_v' = V_s_v

S_s_k' = S_s_k

S_s_v' = S_s_v

ha köv == gép és az [s,o] pozíción álló futóra teljesül a feltétel:

$$T'[i,j] = \begin{cases} 0, \text{ ha } i == s \;\&\&\; j == o \\ 15, \text{ ha } i == s + y * F[0,x] \;\&\&\; j == o + y * F[1,x] \\ T[i,j], \text{ egyébként} \end{cases}$$

L_lista' = L_lista

L_lista'(L_idx) = (s,o,s+y*F[0,x],o+y*F[1,x])

L_idx' = L_idx + 1

köv' = személy

V_s_k' = V_s_k

V_s_v' = V_s_v

S_s_k' = S_s_k

S_s_v' = S_s_v

2.4.7. Bástya lép(x,y) :

// „x” értéke a básatya lépésirányának az alábbi ábra alapján megfeleltetett szám:

	0	
3	B	1
	2	

// az irányokhoz tartozó koordináta növekmények mátrixa:

B = {-1, 0, 1, 0

0, 1, 0,-1}

// „y” a lépés mértékét jelöli [1..7]

alkalmazási előfeltétel:

((köv == személy) && EiEj(T[i,j] == 3 && (T[i+y*B[0,x],j+y*B[1,x]] == 0 ||
T[i+y*B[0,x],j+y*B[1,x]] > 6) &&
Vk((k > 0 && k < y) >> (T[i+k*B[0,x],j+k*B[1,x]] == 0)) &&
léphető(T,személy,i,j, i+y*B[0,x], j+y*B[1,x]))) ||
((köv == gép) && EiEj(T[i,j] == 13 && T[i+y*B[0,x],j+y*B[1,x]] <= 6 &&
Vk((k > 0 && k < y) >> (T[i+k*B[0,x],j+k*B[1,x]] == 0)) &&
léphető(T,gép,i,j, i+y*B[0,x], j+y*B[1,x])))

alkalmazás hatása:

ha köv == személy és az [s,o] pozíción álló bástyára teljesül a feltétel:

$$T'[i,j] = \begin{cases} 0, \text{ ha } i == s \;\&\&\; j == o \\ 3, \text{ ha } i == s + y * B[0,x] \;\&\&\; j == o + y * B[1,x] \\ T[i,j], \text{ egyébként} \end{cases}$$

L_lista’ = L_lista

L_lista’(L_idx) = (s,o,s+y*B[0,x],o+y*B[1,x])

L_idx’ = L_idx + 1

köv’ = gép

$$V\_s\_k' = \begin{cases} \text{FALSE, ha } s == 7 \;\&\&\; o == 7 \\ V\_s\_k, \text{ egyébként} \end{cases}$$

$$V\_s\_v' = \begin{cases} \text{FALSE, ha } s == 7 \;\&\&\; o == 0 \\ V\_s\_v, \text{ egyébként} \end{cases}$$

S_s_k’ = S_s_k

S_s_v’ = S_s_v

ha köv == gép és az [s,o] pozíción álló bástyára teljesül a feltétel:

$$T'[i,j] = \begin{cases} 0, \text{ ha } i == s \;\&\&\; j == o \\ 13, \text{ ha } i == s + y * B[0,x] \;\&\&\; j == o + y * B[1,x] \\ T[i,j], \text{ egyébként} \end{cases}$$

L_lista’ = L_lista

L_lista’(L_idx) = (s,o,s+y*B[0,x],o+y*B[1,x])

L_idx’ = L_idx + 1

köv’ = személy

V_s_k’ = V_s_k

V_s_v’ = V_s_v

$$S\_s\_k' = \begin{cases} \text{FALSE, ha } s == 0 \;\&\&\; o == 7 \\ S\_s\_k, \text{ egyébként} \end{cases}$$

$$S\_s\_v' = \begin{cases} \text{FALSE, ha } s == 0 \;\&\&\; o == 0 \\ S\_s\_v, \text{ egyébként} \end{cases}$$

2.4.8. Vezér lép(x,y) :

// „x” értéke a vezér lépésirányának az alábbi ábra alapján megfeleltetett szám:

0	1	2
7	V	3
6	5	4

// az irányokhoz tartozó koordináta növekmények mátrixa:

V = {-1,-1,-1, 0, 1, 1, 1, 0

-1, 0, 1, 1, 1, 0,-1, -1}

// „y” a lépés mértékét jelöli [1..7]

alkalmazási előfeltétel:

((köv == személy) && EiEj(T[i,j] == 2 && (T[i+y*V[0,x],j+y*V[1,x]] == 0 ||
T[i+y*V[0,x],j+y*V[1,x]] > 6) &&
Vk((k > 0 && k < y) >> (T[i+k*V[0,x],j+k*V[1,x]] == 0)) &&
léphető(T,személy,i,j, i+y*V[0,x], j+y*V[1,x]))) ||
((köv == gép) && EiEj(T[i,j] == 12 && T[i+y*V[0,x],j+y*V[1,x]] <= 6 &&
Vk((k > 0 && k < y) >> (T[i+k*V[0,x],j+k*V[1,x]] == 0)) &&
léphető(T,gép,i,j, i+y*V[0,x], j+y*V[1,x])))

alkalmazás hatása:

ha köv == személy és az [s,o] pozíción álló vezérre teljesül a feltétel:

$$T'[i,j] = \begin{cases} 0, \text{ ha } i == s \;\&\&\; j == 0 \\ 2, \text{ ha } i == s + y * V[0,x] \;\&\&\; j == o + y * V[1,x] \\ T[i,j], \text{ egyébként} \end{cases}$$

L_lista’ = L_lista

L_lista’(L_idx) = (s,o,s+y*V[0,x],o+y*V[1,x])

L_idx’ = L_idx + 1

köv’ = gép

V_s_k’ = V_s_k

V_s_v’ = V_s_v

S_s_k’ = S_s_k

S_s_v’ = S_s_v

ha köv == gép és az [s,o] pozíción álló vezérre teljesül a feltétel:

$$T'[i,j] = \begin{cases} 0, \text{ ha } i == s \;\&\&\; j == o \\ 12, \text{ ha } i == s + y * V[0,x] \;\&\&\; j == s + y * V[1,x] \\ T[i,j], \text{ egyébként} \end{cases}$$

L_lista’ = L_lista

L_lista’(L_idx) = (s,o,s+y*V[0,x],o+y*V[1,x])

L_idx’ = L_idx + 1

köv’ = személy

V_s_k’ = V_s_k

V_s_v’ = V_s_v

S_s_k’ = S_s_k

S_s_v’ = S_s_v

2.4.9. Király lép(x) :

// „x” értéke a király lépésirányának az alábbi ábra alapján megfeleltetett szám:

0	1	2
7	K	3
6	5	4

// az irányokhoz tartozó koordináta növekmények mátrixa:

K = {-1,-1,-1, 0, 1, 1, 1, 0

-1, 0, 1, 1, 1, 0,-1, -1}

alkalmazási előfeltétel:

((köv == személy) && EiEj(T[i,j] == 1 && (T[i+K[0,x],j+K[1,x]] == 0 ||

T[i+K[0,x],j+K[1,x]] > 6) && léphető(T,személy,i,j,i+K[0,x],j+K[1,x]))) ||

((köv == gép) && EiEj(T[i,j] == 11 && T[i+K[0,x],j+K[1,x]] <= 6 &&
léphető(T,gép,i,j,i+K[0,x],j+K[1,x])))

alkalmazás hatása:

ha köv == személy és az [s,o] pozíción álló királyra teljesül a feltétel:

$$T'[i,j] = \begin{cases} 0, \text{ ha } i == s \;\&\&\; j == o \\ 1, \text{ ha } i == s + K[0,x] \;\&\&\; j == o + K[1,x] \\ T[i,j], \text{ egyébként} \end{cases}$$

L_lista' = L_lista

L_lista'(L_idx) = (s,o,s+K[0,x],o+K[1,x])

L_idx' = L_idx + 1

köv' = gép

V_s_k' = FALSE

V_s_v' = FALSE

S_s_k' = S_s_k

S_s_v' = S_s_v

ha köv == gép és az [s,o] pozíción álló királyra teljesül a feltétel:

$$T'[i,j] = \begin{cases} 0, \text{ ha } i == s \;\&\&\; j == o \\ 11, \text{ ha } i == s + K[0,x] \;\&\&\; j == o + K[1,x] \\ T[i,j], \text{ egyébként} \end{cases}$$

L_lista' = L_lista

L_lista'(L_idx) = (s,o,s+K[0,x],o+K[1,x])

L_idx' = L_idx + 1

köv' = személy

V_s_k' = V_s_k

V_s_v' = V_s_v

S_s_k' = FALSE

S_s_v' = FALSE

2.4.10. Rövid sánc :

alkalmazási előfeltétel:

((köv == személy) && V_s_k && !sakk(T,személy) && T[7,5] == 0 &&
T[7,6] == 0 && !támadott(T,személy,7,5) && !támadott(T,személy,7,6)) ||
((köv == gép) && S_s_k && !sakk(T,gép) && T[0,5] == 0 && T[0,6] == 0 &&
!támadott(T,gép,0,5) && !támadott(T,gép,0,6))

alkalmazás hatása:

köv == személy esetén:

$$T'[i,j] = \begin{cases} 0, \text{ ha } i == 7 \;\&\&\; (j == 4 \;||\; j == 7) \\ 1, \text{ ha } i == 7 \;\&\&\; j == 6 \\ 3, \text{ ha } i == 7 \;\&\&\; j == 5 \\ T[i,j], \text{ egyébként} \end{cases}$$

L_lista' = L_lista

L_lista'(L_idx) = (0,0,0,0)

L_idx' = L_idx + 1

köv' = gép

V_s_k' = FALSE

V_s_v' = FALSE

S_s_k' = S_s_k

S_s_v' = S_s_v

köv == gép esetén:

$$T'[i,j] = \begin{cases} 0, \text{ ha } i == 0 \;\&\&\; (j == 4 \;||\; j == 7) \\ 11, \text{ ha } i == 0 \;\&\&\; j == 6 \\ 13, \text{ ha } i == 0 \;\&\&\; j == 5 \\ T[i,j], \text{ egyébként} \end{cases}$$

L_lista' = L_lista

L_lista'(L_idx) = (0,0,0,0)

L_idx' = L_idx + 1

köv' = személy

V_s_k' = V_s_k

V_s_v' = V_s_v

S_s_k’ = FALSE

S_s_v’ = FALSE

2.4.11. Hosszú sánc :

alkalmazási előfeltétel:

((köv == személy) && V_s_v && !sakk(T,személy) && T[7,1] == 0 &&
T[7,2] == 0 && T[7,3] == 0 && !támadott(T,személy,7,2) &&
!támadott(T,személy,7,3)) ||
((köv == gép) && S_s_v && !sakk(T,gép) && T[0,1] == 0 && T[0,2] == 0 &&
T[0,3] == 0 &&!támadott(T,gép,0,2) && !támadott(T,gép,0,3))

alkalmazás hatása:

köv == személy esetén:

$$T'[i,j] = \begin{cases} 0, \text{ ha } i == 7 \;\&\&\; (j == 0 \;||\; j == 4) \\ 1, \text{ ha } i == 7 \;\&\&\; j == 2 \\ 3, \text{ ha } i == 7 \;\&\&\; j == 3 \\ T[i,j], \text{ egyébként} \end{cases}$$

L_lista’ = L_lista

L_lista’(L_idx) = (0,0,0,0)

L_idx’ = L_idx + 1

köv’ = gép

V_s_k’ = FALSE

V_s_v’ = FALSE

S_s_k’ = S_s_k

S_s_v’ = S_s_v

köv == gép esetén:

$$T'[i,j] = \begin{cases} 0, \text{ ha } i == 0 \;\&\&\; (j == 0 \;||\; j == 4) \\ 11, \text{ ha } i == 0 \;\&\&\; j == 2 \\ 13, \text{ha } i == 0 \;\&\&\; j == 3 \\ T[i,j], \text{ egyébként} \end{cases}$$

L_lista' = L_lista

L_lista'(L_idx) = (0,0,0,0)

L_idx' = L_idx + 1

köv' = személy

V_s_k' = V_s_k

V_s_v' = V_s_v

S_s_k' = FALSE

S_s_v' = FALSE

2.5. Célállapot felismerése,végeredmény megállapítása

2.5.1. Célállapotok felismerése

Legyen adott a játék egy „a" != „k" állapota, „O" pedig jelölje a fentebb említett operátorok halmazát.
Az „a" állapot célállapot (Cél(a) = TRUE), ha az alábbi feltételek valamelyike teljesül:

1. a soron következő játékos sakkban van és nem tud szabályosan lépni:
   sakk(T,köv) && !Eo((o € O) && alkalmazási_előfeltétel(o,a))

2. a soron következő játékos nincs sakkban,de nem tud szabályosan lépni:
   !sakk(T,köv) && !Eo((o € O) && alkalmazási előfeltétel(o,a))

3. mindkét játékosnak csak királya maradt:
   EiEjEkEl(T[i,j] == 1 && T[k,l] == 11 &&
   VmVn(((m!=i || n!= j) && (m!=k || n!= l)) >> T[m,n] == 0))

4. a sötétnek csak egy királya van, a világosnak pedig egy királya és vagy egy futója vagy egy huszárja:

EiEjEkElEpEq(T[i,j] == 1 && (T[k,l] == 4 || T[k,l] == 5) && T[p,q] == 11 &&
VmVn(((m!=i || n!=j) && (m!=k || n!=l) && (m!=p || n!= q)) >>T[m,n] == 0))

5. a világosnak csak egy királya van, a sötétnek pedig egy királya és vagy egy futója vagy egy huszárja:
   EiEjEkElEpEq(T[i,j] == 11 && (T[k,l] == 14 || T[k,l] == 15) && T[p,q] == 1 &&
   VmVn(((m!=i || n!=j) && (m!=k || n!=l) && (m!=p || n!= q)) >>T[m,n] == 0))

6. mindkét játékosnak csak egy királya és vagy egy futója vagy egy huszárja van:
   EiEjEkElEpEqEuEv(T[i,j] == 1 && (T[k,l] == 4 || T[k,l] == 5) &&
   T[p,q] == 11 && (T[u,v] == 14 || T[u,v] == 15) &&
   VmVn(((m!=i || n!=j) && (m!=k || n!=l) && (m!=p || n!=q) && (m!=u || n!=v)) >>
   T[m,n] == 0))

7. a sötétnek csak egy királya van,a világosnak pedig egy királya és két huszárja:
   EiEjEkElEpEqEuEv(T[i,j] == 1 && T[k,l] == 4 && T[p,q] == 4 &&
   (k!=p || l!=q) && T[u,v] == 11 &&
   VmVn(((m!=i || n!=j) && (m!=k || n!=l) && (m!=p || n!=q) && (m!=u || n!=v)) >>
   T[m,n] == 0))

8. a világosnak csak egy királya van,a sötétnek pedig egy királya és két huszárja:
   EiEjEkElEpEqEuEv(T[i,j] == 11 && T[k,l] == 14 && T[p,q] == 14 &&
   (k!=p || l!=q) && T[u,v] == 1 &&
   VmVn(((m!=i || n!=j) && (m!=k || n!=l) && (m!=p || n!=q) && (m!=u || n!=v)) >>
   T[m,n] == 0))

2.5.2. Végeredmény megállapítása

A játszma végeredményét a következő függvény segítségével határozhatjuk meg:

nyer:{a| Cél(a)} -> {személy,gép,döntetlen}

$$\text{nyer}(a) = \begin{cases} \text{személy, ha feltétel1 \&\& köv == gép} \\ \text{gép, ha feltétel1 \&\& köv == személy} \\ \text{döntetlen, egyébként} \end{cases}$$

3. Lépésajánló algoritmusok

Különféle lépésajánló algoritmusok ismertek, ezek közül a továbbiakban a „mini-max” és az „alfa-béta vágás” módszerét ismertetem.

3.1. Mini-max módszer

Cél: a támogatott játékosnak, J-nek,egy adott állásban „elég jó” lépést ajánlani.

Az algoritmus számára át kell adni :
- az operátorok halmazát
- J azon „a” állását,ahol lépni következik
- egy mélységi korlátot
- az állások „jóságát” J szempontjából becslő h_j : A -> R heurisztikát, ahol „A” az állapotok, R pedig a valós számok halmazát jelöli

Az algoritmus fő lépései :
1. A játékfa „a” állapotot szemléltető csúcsából kiinduló részének előállítása korlát mélységig.
2. A részfa leveleiben található állások jóságainak becslése a heurisztika segítségével: jóság(n_b) = h_j(b), ahol n_b a „b” állást szemléltető csúcs.
3. Szintenként csökkenő sorrendben a részfa nem levél csúcsai jóságainak számítása: ha n csúcs gyermekei rendre n_1,…,n_k,akkor

$$\text{jóság}(n) = \begin{cases} \max\{\text{jóság}(n_1),\ldots,\text{jóság}(n_k)\}, \text{ ha n szintje páros} \\ \min\{\text{jóság}(n_1),\ldots,\text{jóság}(n_k)\}, \text{ ha n szintje páratlan} \end{cases}$$

Javaslat: az „a” állásból egy olyan lépést tegyen meg J,amelyik az n_a csúcs „jóság” értékével megegyező értékű gyermekébe vezet.

```
function operator minimax_lépés(O,állapot,korlát,hj){
   max = -∞
   operátor = null
   for all o ϵ O do
      if ( előfeltétel(állapot,o) ) {
         új_állapot = alkalmaz(állapot,o)
         v = minimax_érték(O,új_állapot,korlát-1,hj)
         if ( v > max ){
            max = v
            operator = o
         }
      }
   }
   return operator
}

function int minimax_érték(O,állapot,mélység,hj){
   if ( Cél(allapot) || mélység == 0 ) return hj( állapot )
   else  {
      if ( állapot.köv == J){
         max = -∞
         for all o ϵ O do{
            if ( előfeltétel(állapot,o) ){
               új_állapot = alkalmaz(állapot,o)
               v = minimax_érték(O,új_állapot,mélység-1,hj)
```

```
            if ( v > max ) max = v
          }
        }
        return max
      }else{
        min = ∞
        for all o ϵ O do{
          if ( előfeltétel(állapot,o) ){
            új_állapot = alkalmaz(állapot,o)
            v = minimax_érték(O,új_állapot,mélység-1,hj)
            if ( v < min ) min = v
          }
        }
        return min
      }
    }
}
```

3.2. Alfa-béta vágás

Ez a módszer is ugyanazt a javaslatot eredményezi, mint a „mini-max” módszer. Előnye, hogy csökken a játékfa mérete, mert bizonyos részeket nem kell előállítani.

```
function operator alfabéta_lépés(O,állapot,korlát,hj){
  max = -∞
  operátor = null
  for all o ϵ O do{
    if ( előfeltétel(állapot,o) ) {
      új_állapot = alkalmaz(állapot,o)
      v = alfabéta_érték(O,új_állapot,max, ∞, korlát-1,hj)
      if ( v > max ){
        max = v
```

```
            operátor = o
          }
        }
    }
    return operátor
}

function int alfabéta_érték(O,állapot,alfa,beta,mélység,h_j){
   if ( Cél(allapot) || mélység == 0 ) return h_j( állapot )
   else{
     if (állapot.köv == J){
        max = -∞
        for all o ϵ O do{
           if ( előfeltétel(állapot,o) ) {
              új_állapot = alkalmaz(állapot,o)
              v = alfabéta_érték(O,új_állapot,alfa,béta, korlát-1,h_j)
              if ( v > max ){
                 max = v
                 if (max >= béta) return max
                 if (max > alfa) alfa = max
              }
           }
        }
        return max
     }else {
        min = ∞
        for all o ϵ O do{
           if ( előfeltétel(állapot,o) ) {
              új_állapot = alkalmaz(állapot,o)
              v = alfabéta_érték(O,új_állapot,alfa,béta, korlát-1,h_j)
              if ( v < min ){
                 min = v
```

```
                if (alfa >= min) return min
                if (min < béta) béta = min
            }
        }
    }
    retrun min
  }
 }
}
```

4. Heurisztika

A kiértékelő függvény jelentősége abban áll, hogy segítségével heurisztikusan meghatározhatjuk egy állás hozzávetőleges értékét, illetve a nyerés esélyességét. Ha minden egyes játszma folyamán meg tudnánk állapítani a végeredményt, a becslésünknek minössze a következő három értéket kellene használnia :

-1 – veszítés
0 – döntetlen
1 – nyereség

A gyakorlatban azonban, nem tudjuk megállapítani az egzakt értékét egy állánsnak, így becslésekhez kell folyamodnunk. A kezdő sakk játékosok ennek a meghatározásához maguknak a tiszteknek az értékeit veszik alapul. A számítógépes kiértékelő függvények továbbá felhasználják a „material”, továbbiakban összerőpont értéket, mint legjelentősebb tényezőt, mely mellé más egyéb meggondolások is társulnak.

4.1. Általánosságban a heurisztika mögötti filózófiáról

A következőkben áttekintjük a heurisztika mögötti általános meggondolásokat. A specifikus implementációs kérdések tárgyalása helyett, a heurisztika egészét érintő általános konstrukciós szempontokat vizsgáljuk meg.

A heurisztika tulajdonképpen „gyakorlati, tapasztalati szabályok” gyűjteménye, melyek az évszázadok során felhalmozódott emberi tapasztalatokból származtathatóak. Lehetetlen volna azonban leprogramozni minden egyes szabályt, amellyekkel az évek során az emberek előhozakodtak. Még ha le is tudnánk programozni valamennyiüket, akkor sem lenne tanácsos teljesítménybeli okok miatt. Egyensúlyt kell teremtenünk az ismeret és a gyorsaság közt. Minél több időt fordítunk az állás kiértékelésére, annál kevesebb időnk marad a keresésre és ennek következtében a programunk egykre kisebb mélységekben képes kalkulálni.
Néhány program nagyon egyszerű heurisztikus függvénnyel rendelkezik, mely csak a legalapvetőbb szempontokat veszi figyelembe, így több idő marad a keresésre, míg bizonyos programok a bonyolult becsléseket részesítik előnyben, annyi ismeretet felhasználva, amennyi csak lehetséges. A legtöbb program valamilyen, a két véglet közötti becslést használ, mely az utóbbi évek tendenciájának megfelelően egyre inkább az egyszerűség felé tart, azaz egyre elemibb becslő függvényeket használnak. Ez a tendencia annak a meggondolásnak köszönhető, mely szerint ezek a becslések tartalmaznak a legkevesebb “bugot” és ezek a legkönnyebben megvalósíthatóak, ami jóval fontosabb a gyakorlatban mint a sok kisebb jelenőségű ismeret felhasználása. A „Fruit” megjelenése nagy előrelépést jelentett ebbe az irányba, amely igen egyszerű, de mégis nagyon megbízható, komoly, igen erős becslő függvénnyel rendelkezik.

Türelemre, korlátozásokra, alapos tesztelésre és nem túl bonyolult filozófiára van szükségünk, ahhoz hogy célt érjünk. Tapasztalatok szerint az alábbiak a legfontosabb tulajdonságai egy hatékony becslő függvénynek:

1. Folytonosság. Ha két állás (X és Y) közel áll egymáshoz olyan értelemben, hogy X-ből el lehet jutni Y-ba jó lépések egy rövid sorozatán keresztül, akkor a két állásnak ideális esetben hasonló becslésének kellene lennie. Következésképp, ha konkrét sémákhoz nagyobb jutalmat vagy büntetést rendelünk, megfontolandó kisebb jutalom vagy büntetés bevezetése az ilyen sémákhoz valo közel kerülés esetére. Például, amikor nagy jutalomban részesítünk egy „outpost” mezőn álló huszárt, jó ötletnek látszik kisebb jutalomban részesíteni az „outpost” mezőt támadó huszárt is.

félig nyílt vonal : olyan vonal, amelyen nincs gyalogunk, viszont az ellenfélnek legalább egy gyalogja van rajta.

„outpost” : egy olyan mező, mely félig nyílt vonalon található, az ellenfél oldalán helyezkedik el, és gyalog által védett.

2. Megfelelő ítélő képesség. Sokkal fontosabb, hogy a becslő függvényünk képes legyen pontosan megítélni, hogy két nagyon hasonló állás közűl melyik az ígértesebb, mintsem, hogy ugyanezt meg tudja tenni két teljesen különböző állás esetén. A becslő függvénynek nem kell tudnia megválaszolni olyan kérdéseket mint például, hogy egy bizonyos klasszikus „king’s indian” névre hallgató középjáték vajon ígéretesebb e mint például a szicíliai Richter-Rauzer védelemből származó végjáték. Olyan dolgokat szükséges, hogy tudjon meghatározni, mint például, hogy vajon valamely játékosnak meg kellene e próbálni lecserélni a futót egy huszárra, vagy vajon király vagy vezér szárnyon előnyösebb sáncolnunk.

3. Jó-legroszabb eset viselkedés. Jobb tévedni 10 századgyalogot minden esetben, mint 99.9%-ban tökéletesen becsülni és 300 századgyalogot tévedni 0.1%-ban.

4.2. Alapvető becslési szempontok

A sakk terminológiájában a „tiszt” kifejezés kissé kétértelmű és különböző jelentésekkel bírhat. Az általunk használt terminológia a tisztek közé sorolja a gyalogokat továbbá a királyt is. A huszár és a futó könnyűtisztek, a bástya és a vezér pedig nehéz tisztek. A bástyát, a futót és a vezért továbbá sikló tiszteknek nevezi, mivel egy megfelelő irányba tetszőleges számú mezőt léphetnek, amíg útjukat nem állja egy saját vagy ellenséges tiszt, melyet adott esetben leüthetnek.

Az összerőpont értéke általában a legmeghatározóbb tényező a becslésben. Álltalában ez az érték a jelenlévő tisztek konstans értékeinek az összegét jelenti(valamely játékos szempontjából), amely „század gyalog” pontosságú. A gyalogot tekintjük az alap egységnek, melynek értéke 100.

A legelterjedtebb értékek Claude Shannon nevéhez fűzhetőek:

gyalog, huszár, futó, bástya, vezér
(100) (300) (300) (500) (900)

A királynak gyakran egy kiemelkedően magas értéket feleltetnek meg.(például:10000)

4.2.1. További összerőponttal kapcsolatos meggondolások

Ismert tény a sakkban, hogy különféle anyagi jellemzők előnyt jelentenek, mint például a futó par(amely akár egy fél gyalog előnyt is jelenthet). A program növelheti a bástyák értékét abban az esetben, há mar csak kevés gyalog van a táblán, és előnyben részesítheti a huszárokat sok gyalog esetén.

Egyéb tényezők, amelyek hatással lehetnek az összerőpont becslésére :

- jutalmazzuk a futó párt (a futók kiegészítik egymást különböző színű mezőket uralva)
- büntetjük a huszár párt (mivel két huszár kevésbé sikeres a bástya ellen, mint bármely más könnyű tisztekből álló páros)
- csökkentjük a bástya vonalon álló gyalogok értékét valamint növeljük a centrum gyalogok értékét (habár ezek a tiszt-mező táblákban is megtehetőek)
- büntetés, ha mar egyetlen gyalogja sincs egy játékosnak, mivel ez a tény nehezebbé teszi a végjáték megnyerését.

Az imént felsorolt szempontok nem megfelelő használata azonban rossz játék stílushoz vezethet.

4.2.2. Elégtelen összerőpont

A legtöbb program speciális kódot, vagy táblákat használ a döntetlen illetve valószínűsíthetően döntetlen anyagi kombinációk feltárására. Például KF vs K döntetlen, csakúgy mint KH vs K és KHH vs K.

Továbbá egy csoportba sorolhatjuk a majdnem biztos döntetleneket, amelyekről nem szól a FIDE szabályzata, a mattadás elvi lehetősége miatt.Ilyenek például a KHH vs KF, KFH vs KF, vagy a KFH vs KB stb.

Egy általános, bar nem tökéletes szabály arra, hogy sok valószínűsíthetően döntetlen állást felismerjünk :

Ha az egyik félnek mar nincs egyetlen gyalogja sem, négy gyalognak megfelelő anyagi többlettel kell rendelkeznie a nyeréshez.

Például : KBH vs KB általában döntetlen (csak 3 gyalognak megfelelő az előny), míg KBB vs KFH általában győzelem bástya oldalon.

4.2.3. Összerőpont mérleg

Az összerőpont mérleg csaknem a legmeghatározóbb kiértékelési tényezőként jelenik meg, melyet általában a lépni következő játékos szemszögéből számítunk. Alapjában véve a két játékos összerőpontjának a különbsége, azaz:

M = összerőpont[lépni következő játékos] – összerőpont[lépni következő játékos ellenfele]

4.2.4. Tiszt-mező táblák

Az úgynevezett „tiszt-mező táblák” használata egy egyszerű módszer arra, hogy meghatározott helyeken álló meghatározott tisztekhez értékeket rendeljünk. Mind sötét, mind világos oldalon minden fajtájú tiszthez létrehozunk egy táblát, melyeknek minden egyes mezőjéhez értéket rendelünk. A módszer gyors, mivel a becsléshez szükséges adatok a tiszt-mező táblákból növekményesen frissíthetők, amint lépés történik a játékfában. Ezen gyorsaságnak köszönhetően a tiszt-mező táblák nagy jelentőségűek lassú kiértékelés esetén.

Néhány program, mint például a Fruit, két csoport táblát használ. Egyet a megnyitásban, egyet pedig a végjátékban. A végső becslés ezek interpoláltja, a játék állásának megfelelően(amelyet gyakran a táblán található anyagi helyzet jellemez). A két tábla használata továbbá lehetővé teszi a tisztek becslésben betöltött értékeinek a változását a játék előrehaladtával. Például a gyalogok értékesebbé válhatnak a végjátékban ezen módszer alkalmazásával.

Az összerőpont és a tiszt-mező táblák együttes alkalmazása önmagában mar elégséges egy féligmeddig intelligens sakk program számára.

4.3. Egy egyszerű heurisztika

Két részből tevődik össze a heurisztikánk. Az első rész a tiszteknek megfeleltetett értékekről, a második pedig a tiszt-mező táblákról szól.

4.3.1. Tiszt értékek

A tisztek értékeinek meghatározásánál a következő tényezők elérésére törekedtem:
1. elkerüljem a könnyűtisztek három gyalogra valo lecserélését.
2. ösztökéljem a motort a futó pár megtartására
3. elkerüljem két könnyűtisztnek egy bástyára és egy gyalogra valo cseréjét
4. az emberi tapasztalatoknak megfelelően járjak el

Az 1. pont egyszerűen teljesíthető :

F > 3GY
H > 3GY

Természetesen vannak helyzetek, amikor három gyalog (vagy akár kettő is)többet ér mint egy tiszt. Viszont általánosságban elmondható, hogy egy könnyűtisztnek három gyalogra való lecserélése rossz meggondolás eredménye, mivel ezen gyalogok támadás alá kerülhetnek és egy elvesztése is kritikus helyzetet eredményez.

A 2. pont a következőképpen teljesíthető :

F > H

Természetesen ez nem garantál semmit, mivel kerülhetünk olyan végjátékba, hogy csak egy futó marad egy huszár ellen. Általánosságban elmondható, hogy a sakkozók gyakran ütik huszárral a futót, míg ritkán ütnek futóval huszárt. Például a d3 mezőn álló világos futó ritkán üti az e4 centrum mezőn álló sötét huszárt. Ezzel szemben az e4 mezőn álló világos huszár könnyedén ütheti az f6 mezőn álló sötét futót. Természetesen mondhatnánk, hogy néha a g5 mezőn álló világos futó üti az f6 mezőn álló sötét huszárt. Azonban ez általában a h6 kikényszerítése után vagy időbeli tényezők esetén fordul elő, melyet jelen esetben figyelmen kívül hagyunk.

Szóval elkerüljük a futónak a huszárral való nem szükségszerű cseréjét. Növeljük a futó pár meglétének az esélyét. Másfelől fennál annak a kockázata, hogy egy olyan végjátékba kerülünk, amelyben vagy nem megfelelő a futó, vagy csak egy futónk van egy huszár ellen és csak az egyik oldalon állnak gyalogok, vagy zárt állás alakul ki. Tehát szükségünk volna további becslésekre a gyalogok helyzetét előtérbe helyezve, azonban ezen a szinten ezt nem tehetjük.

Így a következőt tehetjük :

F > H > 3GY

A 3. pontnak a következőképp tehetünk eleget :

F + H > B + GY

Egy orosz sakkönyv szerint két könnyűtiszt egy bástya és két gyalog értékének felel meg. Azonban ez kissé túlzásnak tűnik. Vissza tudom idézni a Karpov-Kasparov mérkőzést, ahol Kasparov megnyerhette volna a végjátékot egy bástyával és két gyaloggal két könnyű tiszttel

szemben(plussz néhány gyalog mindkét oldalon). A party döntetlennel fejeződött be, azonban egy bástya és két gyalog akkor is egy kicsit „sok”…

Így ehelyett : B + 2GY > F + H > B + GY
továbbá : F + H = B + 1.5GY

Itt említeném meg a szimmetria szó fontosságát. A sakkban minél szimmetrikusabbak az állások és az összerőpont mérlegek, annál nagyobb a döntetlen esélye.
Esetünkben, mivel az erőviszonyok nem szimmetrikusak(melyet az 1.5 szorzóval is hangsúlyozunk) a rendszer arra fog törekedni, hogy megtartsa két könnyűtisztjét egy bástyával és egy gyaloggal szemben és, hogy megtartsa egy bástyáját és két gyalogját két könnyű tiszttel szemben. Ne feledjük, még mindíg csak erőpontbeli becslésről beszélünk, még a tiszt-mező tábláknál sem tartunk, nem beszélünk valós állás kiértékelésről. Egyszóval lehetőségek tárházát használhatjuk fel a jövőbeli fejlesztésekhez.
Hozzáfűzném még az utolsó egyenlethez, hogy 2H egy kicsit gyengébb lenne mint B + 1.5GY, míg 2F egy kicsit erősebb lenne, de ez most számunkra nem fontos.

Mint látható, igyekszünk eleget tenni a 4. pontban foglaltaknak is.
Az utolsó egyenlet, amelyre hivatkoznék :

V + GY = 2B

Összességében a következő megállapításokat tehetjük :

F > H > 3GY
F + H = B + 1.5GY
V + GY = 2B

A következő érték megfeleltetések kielégítik a megelőző feltételeket :

gyalog, huszár, futó, bástya, vezér, király
(100) (320) (330) (500) (900) (20000)

Továbbá láthatjuk, hogy így egy újabb feltétel teljesül :

F + 2GY > H + 2GY > B

Azonban nem igazán javallott ez utóbbi megállapítás használata, mivel, ahhoz, hogy lecseréljük egy bástyánkat egy könnyűtisztre és két gyalogra nem elég, hogy egyszerűen leütünk egy (mondjuk) huszárt a bástyánkkal majd leütjük a visszaütő gyalogot, ugyanis ezenfelül két plussz „hajlamra" van szükségünk egy újabb gyalog leütéséhez. Így ennek a megvalósítása nem túl gyakori.
Ehelyett egy bástyának egy könnyűtisztre és egy gyalogra valo cseréje sokkal gyakoribb, mellyel némi kompenzáció is elérhető. De jelen pillanatban ez nem a mi dolgunk.

Jelen helyzet szerint :

2F + H > V
B + F + GY > V (vagy B + H + GY > V)
V + GY = 2B

Néhány gondolat a király (K = 20000) értékéhez fűződően:
A király ezen értéke Shannon-tól származik, amely néha hasznunkra válhat annak felismerésében, hogy a király hátrányos helyzetbe került e. Következésképp a maximális összerőpont megközelítőleg :

E = 20000 + 9 * 900 + 2 * 500 + 4 * 300 = 30300

Itt felidéznék egy példát az egyik játékból, melyben a világos nyert anélkül, hogy az a1 mezőn álló bástyát elmozdította volna, mely csendesen várta a b1 mezőn álló huszár kifejlődését, mely szintén nem lépett. Feltehetjük a kérdést, vajon a világos nyerne ha nem lenne kezdettől fogva bástya az a1 mezőn? Valószínüleg nem! Miért? Mert megjelennének olyan csere variánsok, amelyek a sötétnek kedveznek. Szóval az összerőpont mérlegnek meg volt a maga szerepe.

4.3.2. Tiszt-mező táblák

Térjünk át a tiszt-mező értékekre. Általánosságban jutalmazzuk azokat a tiszteket, amelyek előnyös mezőn állnak és büntessük azokat, amelyek hátrányos mezőkön. A többi mező a semleges „0” értéket kapja. Természetesen a fentebb elmített tiszt értékek beleolvaszthatóak a tiszt-mező táblákba. Ettől mi most eltekintünk.

4.3.2.1. Gyalogok

A gyalogokat egyszerűen rá kell vegyük a haladásra. Ezenfelül megpróbáljuk meggátolni a motort abban, hogy a centrum gyalogokat ne mozdítsa előre. A problema ezzel az, hogy ellentmond annak, hogy a gyalogok a király előtt maradjanak. Figyelmen kívük hagyjuk, hogy egy gyalog vonalat váltott vagy sem. Ehhez bonyolultabb becslésre lenne szükségünk, különös figyelmet szentelve annak, hogy „a gyalog a játék lelke”.

0	0	0	0	0	0	0	0
50	50	50	50	50	50	50	50
10	10	20	30	30	20	10	10
5	5	10	25	25	10	5	5
0	0	0	20	20	0	0	0
5	-5	-10	0	0	-10	-5	5
5	10	10	-20	-20	10	10	5
0	0	0	0	0	0	0	0

Néhány szóban a tábláról. Előszöris beszéljünk a védelemről a világos rövid sánc előtt. Az f2, g2, h2 mezőkön álló gyalogokat jutalmazzuk. Továbbá negatív értékeket rendelünk az f3 és g3 mezőkhöz, melyek „lyukat” hagynak a király körül. A h2 gyalognak a h2 és a h3 mezőn is ugyanaz az értéke, így a rendszer szükség esetén létrehozhat „lyukat”. Ezenfelül, ha a gyalog a g3 a futó pedig a g2 mezőn áll, még mindíg megjátszahtó a h3. Ezért ugyanaz az érték társul h3-hoz mint h2-höz.

A centrumban az e2 illetve a d2 mezőn találjuk a legnegatívabb értékeket. Ezen gyalog állások nem előnyösek. A d3 és e3 mezők sem kedvezőek, d4, e4 a centrumban viszont mar igen, d5, e5 pedig méginkább. A 6.ik sorral kezdődően jutalmazzuk a haladó gyalogokat.

4.3.2.2. Huszárok

A huszárokat a centrumba kerülésre kell buzdítanunk. A széleken való állás kedvezőtlen. A sarokban való állás pedig egyenesen borzalmas. „Ha egy tiszt előnytelen mezőn áll, az egész állás hátrányos” .

-50	-40	-30	-30	-30	-30	-40	-50
-40	-20	0	0	0	0	-20	-40
-30	0	10	15	15	10	0	-30
-30	5	15	20	20	15	5	-30
-30	0	15	20	20	15	0	-30
-30	5	10	15	15	10	5	-30
-40	-20	0	5	5	0	-20	-40
-50	-40	-30	-30	-30	-30	-40	-50

Amint látható boldogan lecserélném bármely szélen álló huszárt 3 gyalogra.
Kisebb jutalmakat rendeltem a következő mezőkhöz : d2, e2, b3, b5, g3, g5,
valamint jutalmazzuk a centrumban álló huszárokat is.

4.3.2.3. Futók

-20	-10	-10	-10	-10	-10	-10	-20
-10	0	0	0	0	0	0	-10
-10	0	5	10	10	5	0	-10
-10	5	5	10	10	5	5	-10
-10	0	10	10	10	10	0	-10
-10	10	10	10	10	10	10	-10
-10	5	0	0	0	0	5	-10

-20	-10	-10	-10	-10	-10	-10	-20

Kerüljük a sarkokat és a széleket. Továbbá jónak találjuk az olyan mezőket mint például: b2, b3, b5, g2, g3, g5, valamint a centrum belieket.

4.3.2.4. Bástyák

0	0	0	0	0	0	0	0
5	10	10	10	10	10	10	5
-5	0	0	0	0	0	0	-5
-5	0	0	0	0	0	0	-5
-5	0	0	0	0	0	0	-5
-5	0	0	0	0	0	0	-5
-5	0	0	0	0	0	0	-5
0	3	4	5	5	0	0	0

A bástyát illetően csak a következő néhány gondolat jutott eszembe:központosítás, birtokba venni a 7. sort és elkerülni az „a” valamint a „h” vonalakat.

4.3.2.5. Vezér

-20	-10	-10	-5	-5	-10	-10	-20
-10	0	0	0	0	0	0	-10
-10	0	5	5	5	5	0	-10
-5	0	5	5	5	5	0	-5
0	0	5	5	5	5	0	-5
-10	5	5	5	5	5	0	-10
-10	0	5	0	0	0	0	-10
-20	-10	-10	-5	-5	-10	-10	-20

Általánosságban elmondható, hogy negatív értékeket rendeltem azokhoz a mezőkhöz, amelyeken nem szívesen tartanám a vezérem. Továbbá minimális jutalmat rendeltem a

centrum mezőkhöz a vezér centrumban való tartása érdekében, valamint a b3 és c2 mezőkhöz külső javaslatra.

4.3.2.6. Király

-30	-40	-40	-50	-50	-40	-40	-30
-30	-40	-40	-50	-50	-40	-40	-30
-30	-40	-40	-50	-50	-40	-40	-30
-30	-40	-40	-50	-50	-40	-40	-30
-20	-30	-30	-40	-40	-30	-30	-20
-10	-20	-20	-20	-20	-20	-20	-10
20	20	0	0	0	0	20	20
20	30	10	0	0	10	30	20

A király esetében két táblát használunk. Egyet a megnyitástól a végjátékig, egyet pedig a végjáték megkezdésétől. Az első tábla igyekszik a királyt a gyalog védelem mögött tartani. A végjátékban az értékek a következőképpen változnak :

-50	-40	-30	-20	-20	-30	-40	-50
-30	-20	-10	0	0	-10	-20	-30
-30	-10	20	30	30	20	-10	-30
-30	-10	30	40	40	30	-10	-30
-30	-10	30	40	40	30	-10	-30
-30	-10	20	30	30	20	-10	-30
-30	-30	0	0	0	0	-30	-30
-50	-30	-30	-30	-30	-30	-30	-50

Természetesen az imént említett értékek a világos játékosra vonakoznak, sötét esetében a tükrözött értékekkel kell számolnunk. Továbbá definiálnunk kell, hogy hol is kezdődik a végjáték.

Két javaslat közül választhatunk :

1. egyik félnek sincs már vezére
2. ha valamelyik játékosnak van vezére, legfeljebb egy könnyűtiszje lehet ezenfelül.

4.4. Mobilitás

A mobilitás egy játékos által adott állásban megtehető szabályos lépések mértéke. A mobilitás gyakran használt szempont a becslő függvény megkonstruálásakor. Hasznossága azon az elgondoláson alapszik, mely szerint minél több lehetőségünk van egy állásban, annál erősebb a helyzetünk az adott állásban. Egy 380 versenyjátszmát, melyekben még a 20-adik lépés után is egyforma volt a két játékos összerőpont mérlege,magában foglaló tanulmány szerint jelentős összefüggés van a játékos mobilitása és a nyert játékok száma között.

4.4.1. A mobilitás számítása

A sakk programokban a mobilitás néha a szabályos lépések számának összegétől eltérően számítódik. Gyakran tisztenként számítódik, és a szabályos lépésenkénti mobilitás bónusz nem minden tiszt esetében azonos. Például a megnyitásban a futók és huszárok mobilitása sokkal fontosabb mint a bástyáké. Olykor az előrehaladó mobilitás nagyobb értékű mint a visszafelé haladó, néha (a bástyák esetén) a függőleges mobilitás elsőbbséget élvez a vízszintessel szemben. Továbbá, néha számitásba vesszük az olyan jellegű „lépést” is, amikor egy tiszt egy saját tiszt mezőjére léphet. Bár ezen lépés nem lenne szabályos, mégis fontos szerepet játszik, a saját tiszt védelmét tekintve.

Néhány program az úgynevezett „biztonságos mobilitást” használja a becslésben. Azaz csak azokat a mezőket számolja bele a lehetőségekbe, amelyekre lépve nem kerül támadásba az adott tiszt. Ez meglehetősen költséges lehet, hacsak a program nem tartalmaz növekményesen frissített támadási táblákat. Néhány esetben, leginkább a huszárok esetében, a centrumhoz közelítő lépések (azon mezőket leszámítva melyek ellenséges gyalogok által támadottak) tünnek a legértékesebbeknek

.

4.5. Tőrbe esett tisztek

Rendkívül gyenge mobilitás esetén beszélünk tőrbe esett tisztekről, melyek hátrányos helyzetűek és gyakran veszélyeztetettek. Egy tipikus példája a tőrbe esett tiszteknek a h7 mezőn álló világos futó, melyet az f7 és g6 mezőn álló sötét gyalogok gátolnak. Az esetek többségében végül leütik, azonban ez oly sok időt vesz igénybe, hogy könnyen túlmutathat a „látóhatáron". Az esetek többségében a legjobb dolog amit egy ilyen helyzetű futó tehet, hogy beáldozza magát egy gyalogért, így az értékét körülbelül 150.-el kellene csökkentenünk.

Egyéb lehetséges példák hasonló helyzetekre :

- a h8 mezőn álló világos huszár a h7 vagy f7 mezőn álló sötét gyalog jelenlétében.
- a h6 mezőn álló világos futó az f6 és g5 mezőkön álló sötét gyalogok jelenlétében.(ezt kevésbé szigorúan kell pontoznunk, kb -50)
- a h7 mezőn álló világos huszár a g7 és h6 mezőn álló sötét gyalogok jelenlétében.
- a h1/g1/h2/g2 mezők valamelyikén álló világos bástya az f1/g1 mezők valamelyikén álló világos királlyal.(kb -40, hogy megelőzzük az „ál-sáncolást" a körülvevő bástyával)

4.6. Horizon effektus

A „horizon effektus" a mesterséges intelligencia egy jellegzetes problémája.
Amikor hatalmas játékfákban keresünk (például minimax vagy alfa-béta vágást alkalmazva), gyakran nem lehetséges a teljes fa felépítése, így cask egy bizonyos mélységig keresünk a fában. Ez eredményezi a „horizon effektust", mely esetén épp a „látóhatáron" túl jelentkezik a jelentős „változás"(nemsokkal a keresési mélységen túl). A részleges játékfa kiértékelés ennél fogva félrevezető eredményt ad.

Egy jó példa a „horizon effektus" előfordulására, amikor valamilyen negative esemény elkerülhetetlen de elhalasztható, viszont mivel csak egy részfa analizálása történt meg, a rendszer úgy érzékeli, mintha az esemény elkerülhető lenne, holott valójában nem ez a helyzet. Például vegyük azt a helyzetet, amikor a sötét 6. szintig keres a játékfában, és az

aktuális állásból látja, hogy el fogja veszíteni a vezért a 6. szinten. Tegyük fel továbbá, hogy a bástya feláldozásával a „motor” elodázhatja a vezér elvesztését a 8. szinre. Természetesen ez rossz lépés, mivel a vezér elvesztésén túl a bástya elvesztéséhez is vezet. Azonban, mivel a vezér elvesztése a keresés „látókörén” túlmutat, a keresés ezt az információt nem tudja számításba venni. A bástya feláldozása jobbnak tűnik, mint a vezér elvesztése, így a feláldozó lépés tér vissza legjobb lehetőségként.

A „horizon effektus” csillapítható a kereső algoritmus „quiescence kereséssel” valo kiegészítésével. Így az algoritmusunk képes lesz bizonyos fajta, a játék állására nagyobb hatást gyakorló lépések esetén(például ütések) „túllátni” a „látóhatárán”.

A becslő függvényünknek a levél csúcsokra vonatkozó újraírásával, és/vagy lényegesen több csúcs analizálásával a „horizon effektus” problémák száma csökkenthető.

5. Sakk implementáció

A következőkben összefoglalom az állapottér reprezentációt megvalósító Java implementációmra vonatkozó legfontosabb tudnivalókat.

5.1. Futtatás és játék

A játék futtatásához szükség van egy előre létrehozott C:\Chess\Lastgame.txt filera, amely az utoljára lejátszott játszma lépéseit tartalmazza. A Java bájtkód fájlokat ugyanezen könyvtárba másolva a program a következőképpen indítható parancssorból :

```
cd\
java Chess.Game
```

Ennek hatására a program kiírja a játék használatához szükséges paramétereket és megadási módjukat. Lehetőség nyílik szemény vs személy, gép vs személy valamint gép vs gép játékra.

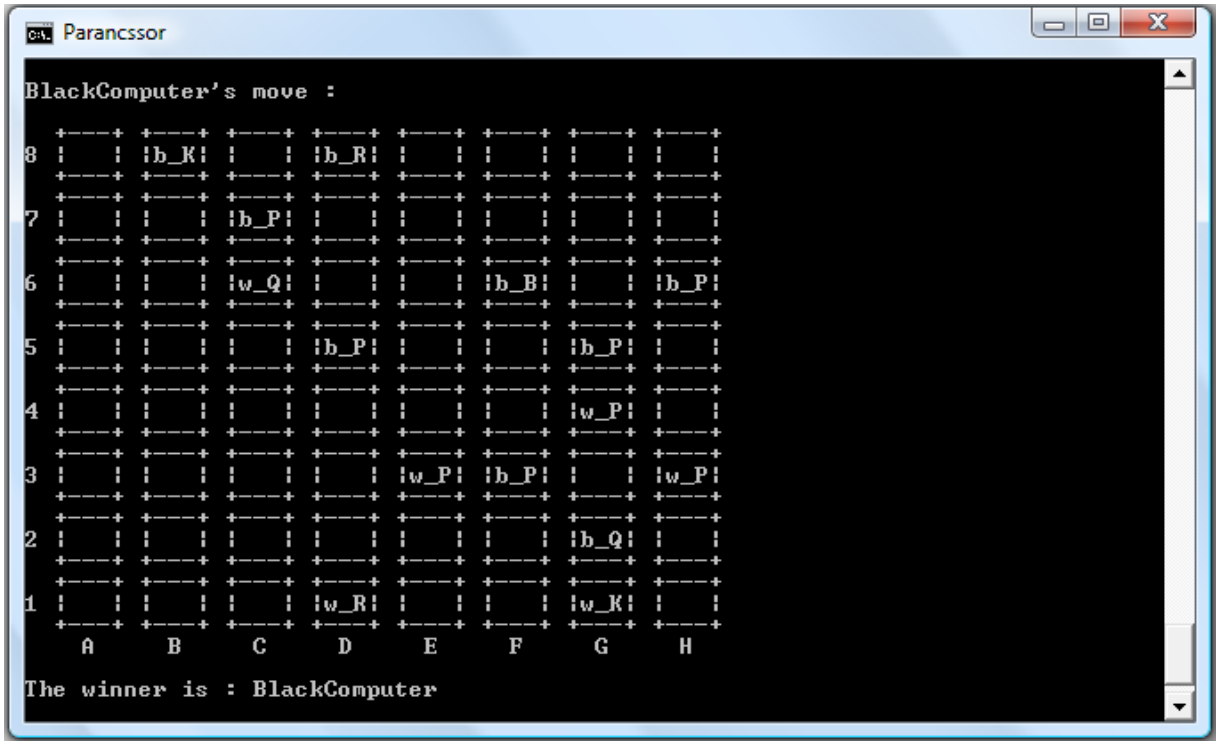
A paraméterként megadott két név küzül az első világosként a második pedig sötétként fog játszani. A „Computer” név megadása gépi játékos beállítását jelenti.
A játékosok úgy léphetnek, hogy megadják kezdő mező majd a cél mező vonal illetve sor azonosítóit. Szabálytalan lépés esetén új lépés megadására van lehetőség, egészen addig, míg a játékos szabályosat nem lép.

Lépésre példák : E2E4,B1C3…

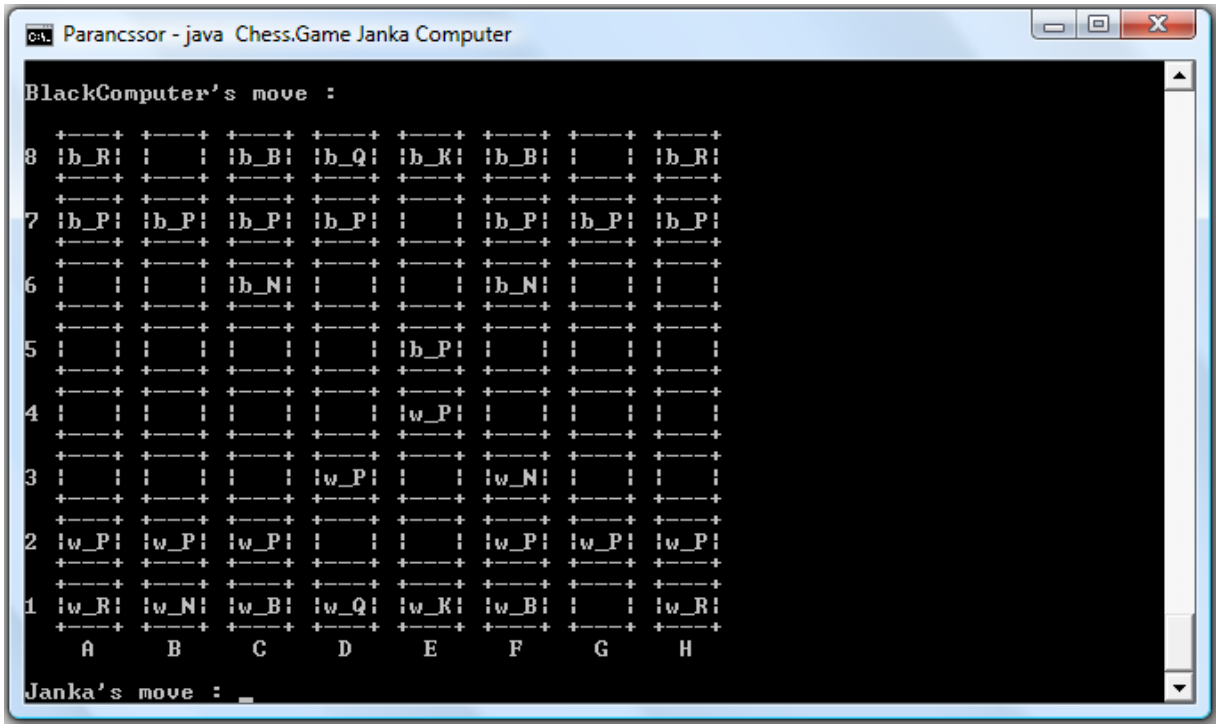
Itt hívnám föl a figyelmet a kis és nagy betű különbségekre. Csak a nagybetűs és szabályos lépések lesznek elfogadva.

5.2. Képek a játékból

Az alábbiakban két kép látható a játékból.



Gép vs gép játék végeredménye.



Sötét gép ellen folytatott játékom kezdő lépései.

6. Összefoglalás

Az elemzésem első szakaszában bemutattam egy lehetséges sakk állapottér reprezentációt, mely egyben a java implementációm alapját is képezi. A következő szakaszban két lépésajánló algoritmust ismertettem, a mini-max módszert és az alfa-béta vágást, melyek ugyanazt a lépést javasolják, azonban teljesítménybeli okokból kifolyólag az alfa-béta vágás került implementálásra. Ezt követően a heurisztika megkonstruálásához használható szempontok tárházából igyekeztem a leginkább meghatározókról szót ejteni.. Végezetül pedig a konkrét java implementációm működéséről és használatának szabályairól írtam.

Az állapottér reprezentáció megkonstruálása során az operátor alkalmazások előfeltételeinek meghatározása jelentette a legnehezebb feladatot, mely során korábban ismertetett pszeudo kódú függvények alkalmazásával tettem átláthatóbbá a logikai feltételeket.

Az implementálás során felmerülő kérdések közül egyértelműen a heurisztika meghatározása tűnt a legnagyobb kihívásnak, ugyanis megfelelő bonyolultságú kiértékelés esetén még a mai számítógépek sem képesek nagyobb mélységekben kalkulálni.
A program a diplomamunkában említett szempontok közül mindössze a megfelelően meghatározott tiszt értékekkel és az egyes tisztekhez készített súlyozott „tiszt-mező” táblákkal kalkulál.

A sakkal kapcsolatos vizsgálódásaimhoz jó kiindulópontot jelentett a http://chessprogramming.wikispaces.com/ weboldal tanulmányozása, ahol a sakkot érintő kérdésekben hasznos információk tömkelege áll rendelkezésre.

Végezetül pedig szeretnék szólni néhány szót a program rendeltetéséről :
a program megírásával nem az volt a célom, hogy a forgalomban lévő sakk programok optimalitásával és erősségével felvegyem a versenyt, hanem az, hogy egy olyan teljes értékű programot készítsek, melynek implementációja könnyen átlátható, elemezhető, esetenként egyéni igény szerint bővíthető. Így a sakk számítógépes megvalósítása iránt érdeklődők egy biztonságosan működő forráshoz jutahatnak, melyet fejlesztéseik alapjául tekinthetnek.

7. Irodalomjegyzék

Nyékyné G. Judit (2001):

Java 2. útikalauz programozóknak: 1.3. ELTE TTK Hallgatói Alapítvány, Budapest

Várterész Magda(2006):

Mesterséges intelligencia 1 előadások

Chess programming:

http://chessprogramming.wikispaces.com/

John L. Jerz(2009):

A Proposed Heuristic for a Computer Chess Program

http://mysite.verizon.net/vzesz4a6/current/jerz.pdf