Bevezetés

A szakdolgozati téma kiválasztásánál fő szempont volt, hogy gyakorlati feladat legyen egy olyan területről, ahol korábban még nem szereztem tapasztalatot, hogy ezzel is növelhessem későbbi elhelyezkedési esélyeimet. A Java nyelvre és azon belül is a J2ME-re ezért esett a választásom. Objektum orientáltsága és platform függetlensége miatt korszerű, könnyen használható és gyorsan elsajátítható nyelv.

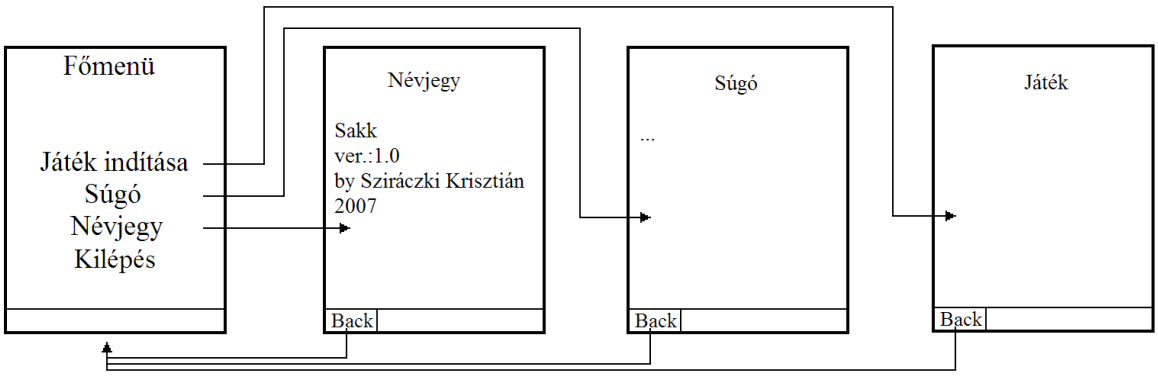
Mai modern világunkban már senki sem létezhet mobiltelefon nélkül és habár minden készülék képes Java programok futtatására, mégsem terjedt el igazán. Szinte csak játékok készülnek J2ME nyelven. Azért döntöttem én is egy játék megírása mellett, mert leginkább a grafika és multimédia foglalkoztat. Emellett a MIDP 2.0 verziója már tartalmaz egy beépített Game csomagot, ami nagy segítséget nyújthat a programozásban.

Szakdolgozatom célja egy játékprogram megírása, amellyel megismerhetem és bemutathatom a J2ME-ben rejlő lehetőségeket. A nyelv-be épített Game csomag egyelőre leginkább a platform játékokat támogatja, ezért esett a választásom egy sakk programra. Az elkészített alkalmazás csak két ember egymás elleni játékát támogatja, mivel egy komolyabb sakk logika túlnő az erőforrások adta lehetőségeken.

1. Tervezés

1.1.A felhasználói felület tervezése

A képernyők tervezésénél a játékprogramok általános felépítését, a készülék és J2ME nyelv adta lehetőségeket vettem figyelembe. Az elkészített program két fő részből épül fel, melyek a menürendszer és maga a játék. A menü részei a játékoknál megszokott legáltalánosabb elemek, Játék indítása, Súgó, Névjegy és Kilépés menüpontok (1. ábra). A játék indítása pontban új játékot indíthatunk, vagy ha visszaléptünk a menübe, ezzel folytathatjuk a játszmát. A Súgó egy kevés információt tartalmaz az irányításról és játékszabályokról. A Névjegy menüpont pedig a szokásos adatokat tartalmazza, verzió, kiadás éve, szerző, stb.



1. ábra: Menürendszer

A játék képernyő megtervezésénél a cél készülékek adottságait kellett figyelembe venni. A programot 320*240 felbontáshoz terveztem, ez megfelel a PDA-k és újabb 3. generációs mobiltelefonok képességeinek. Mivel a kép mindkét esetben meglehetősen kicsi, fontos volt odafigyelni, hogy a végeredmény élvezhető maradjon. A terület legnagyobb részét ezért a játéktérnek kellett fenntartani. (2. ábra)

2. ábra: Játék képernyő

1.2.A játék működésének megtervezése

A játék tervezésénél a valódi sakk felépítését vettem alapul. Szükséges kellékek a sakktábla és 32 bábú. Ezek megvalósításához kell egy bábu objektum, amiben a hozzá tartozó sprite-ok és a lépésszabályok kapnak helyet. A lépésszabályoknál ellenőrizni kell a bábuk haladását, egyenes, átlós, L alakú vagy egyenes és átlós a király és királynő esetén, figyelni kell, hogy ha lépéskor a bábu útjában van egy másik, akkor azt nem lépheti át, és természetesen hogy világos után a sötét lép. Ebből a bábu objektumból létrehozott 32 elemű tömb tartalmazza a 32 bábut. A sakktáblát pedig egy 8*8 elemű tömb jelképezi, ebben tároljuk azt az információt, hogy melyik mezőn milyen bábú található.

2. Megvalósítás

2.1.Fejlesztői környezet

A programfejlesztéshez a NetBeans IDE 5.5 (Integrated Development Enviroment) verzióját választottam, ami a Sun nyílt forráskódú ingyenes fejlesztői környezete. Ehhez szükséges a J2SE Development Kit (JDK) legalább 1.5 verzióját telepíteni, és hogy mobil alkalmazások készítésére is alkalmas legyen, kell még hozzá a Mobility Pack kiegészítő telepítése is. A Mobility Pack tartalmazza a Wireless Toolkit 2.2 verzióját, ami egy emulátort (3. ábra) és több hasznos funkciót is tartalmaz. Az elkészített alkalmazást az emulátoron próbálhatjuk ki. A továbbiakban én is ennek segítségével fogom bemutatni a programot. A felsorolt programok mindegyike letölthető a Sun Developer Network oldaláról (www.java.sun.com).

3. ábra: A Wireless Toolkit emulátora

Minden mobil gyártó kiegészíti a készülékein található J2ME-t saját csomagokkal, osztályokkal, ezek kezelik az olyan speciális elemeket, mit a billentyűk háttérvilágítása, rezgések és egyéb egyedi funkciók. Ezért az én esetemben a Motorola J2ME SDK-t is telepíteni kellett. Ez szintén tartalmaz egy emulátort, amiben a megfelelő modellt kiválasztva kipróbálhatjuk, hogy működik-e a program a cél készüléken.

2.1.1.Néhány szó a J2ME-ről

A J2ME (Java Micro Edition) a Java nyelv legkisebb változata, kifejezetten mobiltelefonok és más hordozható eszközök (PDA, Palmtop, stb.) számára fejlesztette ki a Sun Microsystems. Annyira a kis készülékek számára lett optimalizálva, hogy csak néhány kilobyte memóriára van szüksége. Nyelvi szabályait tekintve megegyezik a többi Java változattal. A különbség közöttük annyi, hogy a J2ME nem tartalmazza a Java minden csomagját és osztályát.

Alapfogalmak: (forrás: J2ME SDK dokumentáció)

- CLDC: Connected Limited Device Configuration, kisebb hordozható eszközökhöz készült konfiguráció, általában elemmel, korlátozott memória kapacitással, korlátozott számolási teljesítménnyel, alacsony sávszélességű kapcsolattal rendelkeznek.
- MIDP: Mobile Information Device Profile, CLDC készülékekhez készült profil, olyan funkciókat határoz meg, mint például felhasználói felület használata, hálózati működés, alkalmazás modell.
- MIDlet: MIDP alkalmazás
- JAR: Java Archive Resource, tömörített Java program, ebben vannak a .class file-ok és egyéb erőforrások (képek, hangok, stb.), valamint a manifest file is.
- JAD: Java Aplication Descriptor, információkat tartalmaz a MIDlet-ről, a jar file betöltése előtt megállapítható, hogy alkalmas-e az alkalmazás az eszközön való futtatásra.

2.1.2.MIDP 2.0

A MIDP második verziójának legtöbb újdonsága a játékfejlesztést támogatja. Beépítésre került egy Game csomag, amely több olyan osztályt tartalmaz, aminek segítségével egységesíteni lehet a különböző gyártók által használt megoldásokat, ezzel javítva a programok kompatibilitását. A MIDP 2.0 alapú szoftverek akár HTTPS protokollon keresztül is összeköttetést teremthetnek a világhálóval a mobil hálózaton keresztül. Hátránya viszont hogy ez a verzió már több memóriát is igényel.

A Game csomag osztályai: (forrás: J2ME SDK dokumentáció)

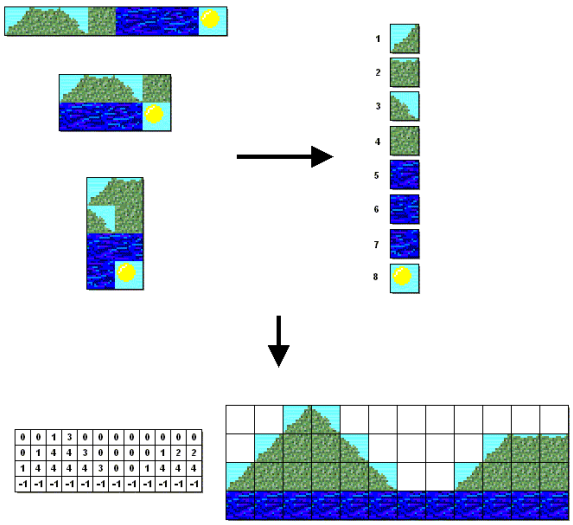
- *GameCanvas*
- *Layer*
- *LayerManager*
- *Sprite*
- *TiledLayer*

GameCanvas: Az LCDUI Canvas osztályának alosztálya, játékokra specializálódott képernyő funkciókat szolgáltat, billentyű állapot lekérést és folyamatos képernyőfrissítést.

Layer: Absztrakt osztály, a rétegek alaptulajdonságait adja, elhelyezkedés, méret, láthatóság.

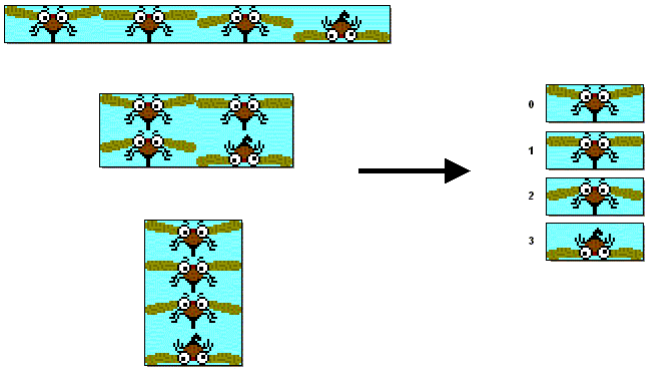
LayerManager: Automatikusan rendereli az általunk létrehozott rétegeket, elkészíti a végleges képet.

TiledLayer:(4. ábra) Nagyobb képeket hozhatunk létre mozaikokból, egyes képkockák cserélgetésével akár animációt is készíthetünk. Ez alkalmas arra, hogy néhány kockányi képből akármekkora dinamikus hátteret hozzunk létre a játék számára. Néhány kép vagy az egész készlet lecserélésével megváltoztathatjuk a környezetet akár futás közben is.



4. ábra: TiledLayer példa (forrás: J2ME SDK dokumentáció)

Sprite: Egy vagy több képkockából álló animáció, minden képkockának azonos méretűnek kell lenni. A program automatikusan osztja fel a képet kockákra (5. ábra). később alkalmazhatunk transzformációkat is az egyes képkockákra, pl.: forgatás, tükrözés. A képkockák sorba fűzését is átállíthatjuk, így egy négy kockás képből akár több különböző komolyabb animációt is létrehozhatunk (az alábbi mintára pl.: {0,1,2,1,0,1,2,1,0,1,2,1,0,1,2,1,1,1,1}). A sprite-ra beállítható egy Reference Pixel is, amivel az animációk pontos illesztését tudjuk beállítani.



5. ábra: egy Sprite felépítése (forrás: J2ME SDK dokumentáció)

2.2. A program működése

Az elkészített Sakk csomag 6 osztályból épül fel, ezek a *mainMidlet*, *myGameCanvas*, *Babu*, *Menu*, *Help* és *About*. Ezeken kívül a csomag tartalmazza még a játék háttérképét, a menü hátterét, a 12 különböző bábu képét, és a kiválasztó négyzet két állapotához egy-egy képet.

2.2.1.A program életciklusa

Minden J2ME programnak tartalmaznia kell egy Midlet-et, ez az én esetemben a *mainMidlet*, ezt tölti be elsőként az alkalmazásvezérlő, ez létrehoz egy-egy példányt a program osztályaiból, és elindítja a menüt. Innentől láthatunk képet is. A főmenüből indíthatjuk a játékot, érhetjük el a Súgó és Névjegy menüpontokat, vagy a kilépést. A játék elindítása után *myGameCanvas* osztály konstruktora elhelyezi a bábukat a kezdő pozícióban, a kijelölő négyzetet pedig a sakktábla közepén, egy üres mezőn. Az első lépést mindig a világos teszi. Az időmérők első lépés után indulnak, külön-külön mérik a két szín idejét. Az órákat a bábuk lerakása váltja. A billentyűzet kezelését a *myGameCanvas* osztály *Input* metódusa végzi, lekéri a billentyűzet állapotát, és ez alapján dönti el, hogy mi történjen. Mozgatja a kijelölő négyzetet, vagy a tűz gomb lenyomása esetén elindítja a lépés folyamatát.

2.2.2.A megjelenítés

A játék grafikája a MIDP 2.0-val bevezetett *Game* csomagon alapul, a bábukat sprite-ként jeleníti meg. A *myGameCanvas* osztály konstruktora a játékhoz szükséges összes képet betölti egy-egy *Image* objektumba, ezekből hozza létre a sprite-okat, amelyek a háttérkép és kiválasztó négyzetek esetén helyben tárolódnak, a bábuk esetében pedig a *Babuk[]* tömb elemeiben. Minden bábú egy egyede a *Babu* osztálynak, ami tartalmazza a sprite-ot és a lépésszabályokat. Következő lépésként szintén még a konstruktor beállítja a bábuk kezdő helyét a *Sprite* osztály *setPosition* metódusával. Végül az összes sprite-ot hozzáfűzi a *layerManager*-hez, mindegyiket egy új layer-ként. Mindig az utoljára hozzácsatolt réteg lesz a legtávolabbi, vagyis legalul, ezért elsőnek a kijelölő négyzet, utána a bábuk, és végül a háttérkép kerül hozzáfűzésre. A megjelenítés további feladatait a *myGameCanvas* osztály *drawScreen* metódusa végzi el. Beállítja a háttérszínt, a karakter típusokat, a *LayerManager paint* metódusával elkészíti a megjelenítendő grafikát, majd ezt

kiegészíti még az időmérőkkel és az aktuális oldalt jelölő feliratokkal. A használhatóság érdekében szükséges jelezni a felhasználó felé, hogy mikor vett fel bábut és mikor vár a program arra, hogy befejezze a lépést. Ennek megvalósításához azt a módszert választottam, hogy a kijelölő négyzet képét megváltoztatom felvételkor és lerakáskor. Ezt a Sprite osztály *setImage* metódusával lehet megoldani, ami a sprite képét cseréli le.

2.2.3.Lépés bábuval és a lépés érvényességének ellenőrzése

A bábuk mozgatását a *myGameCanvas* osztály *Move(int X, int Y)* metódusa végzi. Amit az *Input()* metódus hív meg abban az esetben, ha megnyomtuk a ”tűz” gombot. Az *X* és *Y* a sakktábla kiválasztott mezőjének koordinátái, amit a kiválasztó négyzet pozíciójából, képernyő koordinátából sakktábla mező koordinátákra alakítás után kap paraméterként. A *Move* metódusban található egy feltételes elágazás, aminek feltétele egy *from* nevű boolean változó és az, hogy a kiválasztott mező üres legyen. Első alkalommal a *from* hamis, ezért az elágazás első ága fut le, ami elmenti a kapott *X* és *Y* paramétereket a *fromX* és *fromY* változókba, majd átváltja a *from* változót igaz állásba, ezzel jelezve, hogy megvan a kiinduló koordináta, és lecseréli a kijelölő négyzet képét. Következő alkalommal, amikor a *Move* metódus lefut már az else ág indul el, ami meghívja a *fromX* és *fromY* által kiválasztott bábu *move* metódusát, az megkapja paraméterként a kezdő koordinátákat, és a második alkalommal kapott *X* és *Y* paramétereket.

A *Move* metódus programkódja:

```
public void Move(int X,int Y){
    int key;
    if(from && table[X][Y]!=0){
        fromX=X;
        fromY=Y;
        selSprite.setImage(sel2,28,28);
        do{
           key=getKeyStates();
        }while((key&FIRE_PRESSED)!=0);
        from=false;
    }else if(!from){
        babuk[table[fromX][fromY]].move(fromX,fromY,X,Y);
        selSprite.setImage(sel,28,28);
        do{
           key=getKeyStates();
        }while((key&FIRE_PRESSED)!=0);
        from=true;
    }
}
```

A mozgatás további feladatait a *Babu* osztály *move* metódusa veszi át. Ami paraméterként kapja a lépés kiindulási és cél koordinátáit. Minden bábu tpus rendelkezik egy feltételes leágazással, ahol a hozzá tartozó lépés érvényességi ellenőrzések történnek. A belépési feltétel a *myGameCanvas* osztályban található *table* mátrixból a kiinduló koordináták alapján megállapítható bábu típus és szín, *és a whiteTurn* boolean változó állapota. Ilyen leágazásokkal rendelkezik a királynő, király, futó, huszár bástya, valamint a világos gyalog és a sötét gyalog, ezt a kettőt nem érdemes egyszerre kezelni, mert az ellenkező irányú mozgásuk miatt nem lehet összevonni.

A gyalogok és huszárok esetében egyszerű a vizsgálat, ezeknél csak azt kell ellenőrizni, hogy foglalt-e a cél mező, vagy ha nem akkor leüthető-e. Amennyiben azonos színű bábu található a cél mezőn a lépés érvénytelen. Ha a lépés érvényes az ott található bábut leveszi a megjelenítőről a *layerManager remove(layer l)*metódusával, majd a kiválasztott bábu sprite-ját áthelyezi a cél mezőre, a *table* mátrix kiindulási koordinátáknak megfelelő elemét átmásolja a cél helyére, és nullázza a forrást, ami üres mezőt jelent. Végül meghívja a *myGameCanvas* aktuális színhez tartozó kör átadó metódusát, *darkTurn()* vagy *whiteTurn().* Ezek az időmérők átállítására szolgálnak, és átváltják a *whiteTurn* logikai változót, ezzel jelezve, hogy a másik szín jön.

Példa a világos gyalog lépésének ellenőrzésére:

```
if(SakkCanvas.table[fromX][fromY]>16 &&
       SakkCanvas.table[fromX][fromY]<25 && SakkCanvas.whiteTurn){
   if((SakkCanvas.table[toX][toY]==0 && fromX==toX && fromY-1==toY) ||
         (SakkCanvas.table[toX][toY]<17 && SakkCanvas.table[toX][toY]>0
                 &&((fromX+1==toX || fromX-1==toX) && fromY-1==toY))){
      if(SakkCanvas.table[toX][toY]!=0)
         SakkCanvas.layerManager.remove
                 (SakkCanvas.babuk[SakkCanvas.table[toX][toY]].sprite);
      SakkCanvas.babuk[SakkCanvas.table[fromX][fromY]]
                           .sprite.setPosition(5+(toX*21),5+(toY*21));
      SakkCanvas.table[toX][toY]=SakkCanvas.table[fromX][fromY];
      SakkCanvas.darkTurn();
      if(fromX!=toX || fromY!=toY)
         SakkCanvas.table[fromX][fromY]=0;
   }
}
```

Bástya, futó és királynő esetében bonyolultabb az ellenőrzés, mert itt már azt is figyelni kell, hogy nem léphetnek át bábut. Ezt az összes lehetséges haladási irányhoz elkészített else-if rendszerben vizsgálja meg. A bábutól a cél mezőig az összes útba eső mező tartalmának vizsgálatával. A bástya esetén ez a négy egyenes irányt jelenti, futónál az átlósak, királynő esetében pedig egyenest és átlóst is. Az ellenőrzés és lépés többi eleme megegyezik a gyalognál alkalmazottal.

A futó lépés érvényesség ellenőrzésének részlete:

```
if(((toX-fromX)*(toX-fromX))==((toY-fromY)*(toY-fromY))){
      if(fromX>toX && fromY>toY){
         x=fromX; y=fromY;
         while(x-1!=toX && y-1!=toY){
            x--; y--;
            if(SakkCanvas.table[x][y]!=0)
               ok=false;
         }
      }else if(fromX>toX && fromY<toY){
         x=fromX; y=fromY;
         while(x-1!=toX && y+1!=toY){
            x--; y++;
            if(SakkCanvas.table[x][y]!=0)
               ok=false;
         }
      }else if(fromX<toX && fromY<toY){
         x=fromX; y=fromY;
         while(x+1!=toX && y+1!=toY){
            x++; y++;
            if(SakkCanvas.table[x][y]!=0)
               ok=false;
         }
      }else if(fromX<toX && fromY>toY){
         x=fromX; y=fromY;
         while(x+1!=toX && y-1!=toY){
            x++; y--;
            if(SakkCanvas.table[x][y]!=0)
               ok=false;
      }
   }
```

2.3. Az elkészített osztályok leírása

2.3.1.Menürendszer

6. ábra: Menürendszer

A menüt alapvetően két módon lehet elkészíteni, az egyik a lista, ez nagyon egyszerű, de nem mutatós, magas szintű nyelvi elemekből (list, form, alert) épül fel. Ezt választottam az egyes menüpontoknak. A másik megoldás a grafikus menü. A főmenünek ezt választottam, mert lényeges hogy esztétikus legyen.

***Menu* osztály:**

A főmenüt tartalmazza, elkészíti a grafikát és meghívja a többi almenüt a *mainMidlet*-en keresztül. A *Canvas* osztály leszármazottja. Implementálja a *Runnable* interfészt, ami a szálak futtatásához szükséges.

Metódusai:

menu(mainMidlet midlet):

A Menu osztály konstruktora. Betölti a háttérkép fájlt. A képernyő paramétereit változókba menti a menüpontok elhelyezkedésének kiszámításához és elindítja a menü szál futását.

paint(Graphics g):

Beállítja a háttérszínt és háttérképet, kirajzolja a menüpontokat a képernyő közepére egymás alá.

run():

A konstruktorban létrehozott *menuThread* szál *start()* függvénye hívja meg. Egy végtelen ciklusban meghívja a *Canvas* osztály *repaint()* függvényét, ezzel folyamatos képernyőfrissítést biztosít.

keyPressed(int code):

A *Canvas* osztályból származik. Automatikusan indul minden billentyűleütésnél, lépteti a kijelölő négyzetet a menüpontokon és a Select gomb lenyomására a MIDlet-en keresztül elindítja a játékot vagy egy másik menübe lép.

***Help* és *About* osztály:**

Mindkettő a *Form* osztály leszármazottja és implementálják a *CommandListenert*. A két menü csak a megjelenített szövegben tér el egymástól, az egyik a játék működéséről tartalmaz információkat a másik pedig a programról.

Metódusaik:

Konstruktorok:

Beállítja a megjelenítendő szöveget és a vissza gombot. Elindítja a CommandListenert.

commandAction(Command c, Displayable d):

A vissza gomb megnyomására visszalép a főmenübe.

A *mainMidlet* osztály

Ezt indítja el az alkalmazásvezérlő szoftver, tartja a kapcsolatot a program és a futtató környezet között. Tartalmazza a program életciklusához szükséges metódusokat. A MIDlet osztály leszármazottja.

Metódusai:

startApp():

Létrehoz egy példányt a program összes osztályából és a menuScreen() metóduson keresztül elindítja a menüt.

getDisplay(): Átadja a képernyő kezelését a MIDlet-nek.

startGame(): Elindítja a game szál futását és átadja a képernyőt a játéknak.

menuScreen(), sugoScreen(), aboutScreen():

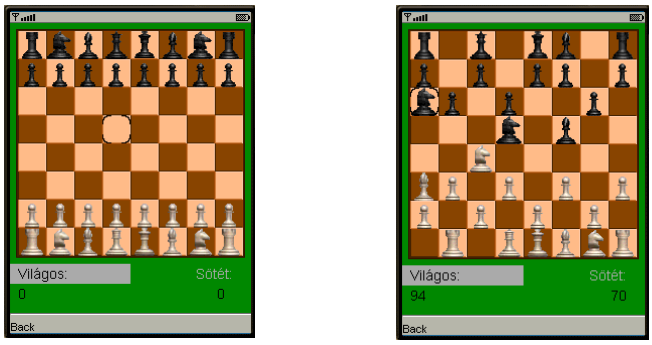
Képernyőt váltanak másik menüpontra. A későbbiekben ezeken keresztül érjük el a főmenüt vagy az almenüket.

pauseApp(), destroyApp(), exit():

A progam életciklusa alatt automatikusan futnak le. Meghívja a garbage collectort és jelenti a környezetnek, hogy a program futása véget ért.

2.3.2.A játék osztályai

A *myGameCanvas* osztály



7. ábra: A játék grafikája

Ez az osztály tartalmazza a játék működtetésének minden elemét. Elkészíti a grafikát, kezeli a billentyűzetet. Leszármazottja a *GameCanvas* osztálynak és implementálja a *Runnable* és *CommandListener* interfészeket. A *Runnable* a game szál futását kezeli, a *CommandListener* segítségével pedig a vissza gombot állíthatjuk be.

Metódusai:

Konstruktor:

Létrehozza a *table* mátrix-ot és a *babuk* tömböt, betölti a megjelenítéshez szükséges képeket, létrehozza a sprite-okat és átadja őket a *layerManager*-nek.

Imput():

A billentyű lenyomások alapján mozgatja a kijelölő négyzetet a képernyőn, és a tűz gomb lenyomására meghívja a *Move* metódust.

Move(int X, int Y):

Átveszi a kijelölő négyzet koordinátáit, letárolja őket először a kiinduló majd cél mezőként, végül paraméterként átadja a bábu *move* metódusának.

whiteTurn() és *darkTurn():*

Kiszámítja az időmérőn megjelenítendő időt, amit mindkét szín esetén külön tárol. Átkapcsolja a *whiteTurn* logikai változót, amivel jelzi, hogy a másik szín jön.

start() és *stop():* Elindítja és leállítja a game szál futását.

run():

Ezt futtatja a szál, egy while ciklusban meghívja az *input()* és *drawScreen()* metódusokat, a *Thread sleep()* metódusával késleltetést végez minden ciklusban, ezzel egyenletes futást biztosít a programnak.

commandAction:

Vissza gombot ad a játék képernyőhöz, amivel a menübe léphetünk vissza.

drawScreen():

Kirajzolja a képernyőn megjelenítendő grafikát, és az időmérőt.

A *Babu* osztály:

Absztrakt osztály, egy-egy eleme tartalmazza a bábuk spritejait, és a lépésszabályokat.

Metódusa:

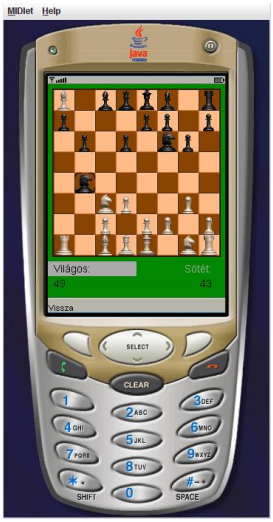
move(int fromX,int fromY,int toX,int toY):

A bábuk mozgatását végzi és ellenőrzi a lépés érvényességét.

3. *Tesztelés*

A tesztelés első lépése a program írása közben történik. Minden új funkciót ki kell próbálni, hogy működik-e. Ez a Wireless Toolkit emulátorával történik, ami a Mobility Pack kiegészítővel a NetBeans-be is beépült, és bármikor elérhető. Az elkészült programban is ellenőrizni kell minden lehetőségét, hogy megfelelően működik-e. Ki kell próbálni a menürendszert, az egyes menüpontok megjelenítését, és legfőképpen magát a játékot. A játékot elindítva minden egyes bábu típust ki kell próbálni, hogy úgy lép-e ahogy kell és enged-e a program érvénytelen lépést. Tesztelés közben figyelni kell az időmérők működését is, hogy helyesen mentse el a két oldal idejét, és következő lépéskor onnan folytassa, ahonnan kell.

8. ábra: Menü tesztelése a Wireless Toolkit emulátorában



9. ábra: A játék tesztelése

Következő lépés mobil készüléken ellenőrizni a program működését. A NetBeans minden fordításkor elkészíti a programból a jar-csomagot és a hozzá tartozó jad fájlt is, ezeket a projekt helyén a dist almappában helyezi el. Ezek már alkalmasak arra, hogy telepítsük mobil készülékre.

A legtöbb gyártó alkalmaz a mobil készülékein telepített J2ME-ben egyedi megoldásokat, ezért érdemes tájékozódni, hogy a cél készülék nyelve miben tér el a szabványostól. Ez lehet a például kép formátumok vagy a sprite-ok kezelésében eltérés. Sok esetben a gyártók rendelkeznek saját fejlesztő környezettel, vagy teszteléshez használható emulátorral, így van ez a Motorola-nál is. Ezért telepítettem a Motorola SDK-t ami tartalmaz egy emulátort. Ennak segítségével több készülék típuson is ki lehet próbálni a programot.

10. ábra: Tesztelés A760-as készüléken a Motorola SDK emulátora segítségével

11. ábra: Tesztelés Motorola A780-as készüléken

Nekem sajnos nem áll rendelkezésemre 3. generációs mobil, vagy PDA, ezért elkészítettem egy olyan verziót a programból, ami 176*220-as felbontásra van optimalizálva. Ehhez át kellett méreteznem a sakkbábuk képét, újrarajzolni a sakktáblát olyan mező méretekkel, amivel a legtöbb helyet tudom kihasználni a kisebb képernyőn, újraszámolni a kijelölő négyzet léptetését és a bábuk lépéseit az új mezőmérethez.

11. ábra: Az alacsony felbontású verzió tesztelése Motorola V980-as készüléken

Az emulátoros tesztek után következhet a program végleges tesztelése mobil készüléken és az üzembe helyezés, ami az én Motorola V300-as telefonomon történt. A program a valóságban is működőképesnek bizonyult, bár a készülék teljesítménye nem elegendő a játék megfelelő sebességen való futásához.

4. *További fejlesztési lehetőségek*

A programban és a nyelvben egyaránt rejlenek lehetőségek a játék fejlesztésére. A nyelv lehetőségei közül a legerősebb a 3D grafika, amit a J2ME legújabb verziói támogatnak, de szükséges hozzá a CLDC 1.1 verziója és a lebegőpontos számításra alkalmas készülék. Erre a legtöbb még nincs felkészítve. Amennyiben maradunk a 2D megjelenítésnél, még az is tartogat lehetőségeket. A sprite alapvetően egyszerűbb animációk megvalósítására a legalkalmasabb. Így ha szakítunk a hagyományos sakkfigurákkal, a megfelelő mennyiségű kép megrajzolása után animált „élő” sakkbábukat kaphatunk, esetleg a sakktáblát is lecserélhetjük egyéb környezetre. A sprite-ok képeinek lecserélésével akár futás közben is válthatjuk a modelleket, hagyományos sakkról animált bábukra, vagy több különböző animált készletet is beépíthetünk, mint például lovagi stílusú bábukészlet, sci-fi bábukészlet vagy különböző játékokon és filmeken alapuló stílusok.

A program jelenlegi állapotában alkalmas lehetne játékállások elmentésére és visszatöltésére, vagy hálózati játékra, úgy hogy a tábla állapotát mentjük el, vagy küldjük át a hálózati módszerek valamelyikén, közvetlen kommunikációval, Bluetooth vagy infravörös kapcsolattal, vagy Wap segítségével egy szerverre, esetleg sms-ben.

5. Összefoglalás

Szakdolgozatom célja volt tapasztalatokat szerezni egy új területen, és elkészíteni egy alkalmazást mobiltelefonra, amely a nyelv legtöbb oldalát bemutatja. Ezeket véleményem szerint sikerült megvalósítani. Önálló munkával elkészítettem egy játékot J2ME nyelven, ami teljesíti az elvárásokat, tartalmaz grafikát, alacsony és magas szintű nyelvi elemeket, emulátoron és a legtöbb mobilkészüléken egyaránt használható. Eközben átfogó ismereteket szereztem a nyelvről, és egy alkalmazás elkészítésének általános módszeréről.

6. *Irodalomjegyzék*

1. Michael Kroll, Stefan Haustein:

   MIDP programing with J2ME

   http://www.developer.com/java/j2me/article.php/10934_1561591_1

   2007.05.17

2. J2ME SDK Dokumentáció – Sun Microsystems

   2007

3. Carol Hamer:

   MIDP 2.0 Games: a Step-by-Step Tutorial with Code Samples

   09/10/2003

   http://www.microjava.com/articles/techtalk/midp2_games

   2007.05.17

4. Michael Powers:

   Getting Started with Mobile 2D Graphics for J2ME

   2005

   http://developers.sun.com/techtopics/mobility/midp/articles/s2dvg/index.html

   2007.05.17.

7. Mellékletek

- 1 db CD melléklet

  Tartalma:

  - forráskódok
  - Mobil alkalmazás (320x240-es verzió)
  - Mobil alkalmazás (176x220-es verzió)