Debreceni Egyetem

Informatikai Kar

JAVA MIDLET – SUDOKU JÁTÉK

Témavezető:
Dr. Fazekas Gábor
egyetemi docens

Készítette:
Veszeli Tamás
programozó matematikus

Debrecen

2007

Tartalomjegyzék

1. Bevezetés

1.1. A probléma felvázolása

A mobil információs eszközök hozzátartoznak mindennapjainkhoz. Hazánkban is szinte mindenki rendelkezik mobil eszközzel. A mobil eszközök legfőbb gyengesége a kis kijelző, és a korlátozott tároló- és számolókapacitás.

A fejlődésnek köszönhetően egyre újabb és gyorsabb technológiák jelennek meg, ezzel lehetővé téve bonyolultabb problémák megoldását a mobil eszközökön is. A fejlődés a mobil eszközök egyre szélesebb körű felhasználását teszi lehetővé. A játékok szinte egy időben jelentek meg a mobiltelefonokkal, és az óta is töretlen a népszerűségük.

A sudoku az utóbbi idők egyik legkedveltebb játéka. Sorra jelennek meg az ezzel foglalkozó oldalak, szinte az összes rejtvényújságban, és rengeteg napilapban, hetilapban is megtalálható. A sudoku nemcsak szórakoztató, hanem fejleszti az ember megoldó képességét és logikáját is.

A sudoku kedvelői egyre több ingyenes, és megvásárolható program közül választhatnak. Ezen programok legfőbb gyengesége, hogy általában előre elkészített feladványokat tartalmaznak, miközben több millió lehetséges feladvány létezik. Így új játék kezdésénél gyakran ugyanazt a feladványt kapjuk. A témával bővebben a 3. fejezetben foglalkozom.

Dolgozatom célja egy olyan sudoku mobilalkalmazás készítése, amely mindig új megoldható rejtvényt állít elő, és megoldhatóvá teszi a játékos számára különböző nehézségi szinteken.

A témával részletesebben a 3. fejezetben foglalkozom.

1.2. Technológiák

A Java Enterprise Edition (J2EE) a szerver és a vállalati alkalmazások platformja, a Java 2 Standard Edition (J2SE) a személyi számítógépeké. Mellettük a Java technológiát a Java Micro Edition (J2ME) valósítja meg a fogyasztói és beágyazott eszközökön. Pl.: mobiltelefonok, egyes PDA-k és egyéb, beágyazott eszközök.

A J2ME az Enterprise és a Standard Edition-höz hasonlóan szabványos Java API-kból épül fel, amelyeket a Java Community Process (JCP) program keretében fejlesztenek olyan csoportok, amelyek vezető készülékgyártókat és szoftverfejlesztőket, forgalmazókat tömörítnek.

A témával részletesebben a 2. fejezetben foglalkozom.

2. Java 2 Micro Edition (J2ME) technológiák

2.1. Java platformok

Nem mindegy, hogy egy Java programot egy mikro rendszerre, egy PC-re, vagy egy többprocesszoros szervergépre fejlesztettek ki. Itt természetesen nem csak a teljesítményre, hanem az egyes lehetőségek szűkülésére és bővülésére is gondolni kell. Éppen ezért a Java bevezette az úgynevezett Java platformokat. Ezek a platformok valamilyen általánosságban lefednek egy-egy felhasználási területet; maga a nyelv pedig az ottani igényeknek megfelelően szűkült, esetleg bővült.

2.1.1. Java SE (Standard Edition)

Kifejezetten munkaállomásokra (pl. PC) szánt változat. A Java itt indult az „Applet”-ekkel, majd az „önálló” asztali alkalmazások is teret hódítottak. Ennek a platformnak kellően nagy memóriája és processzora van, illetve fontos a felhasználó számára kényelmes felhasználói felület is.

2.1.2. Java EE (Enterprise Edition)

Üzleti alkalmazásra szánt változat, gyakran nagyon erős, többprocesszoros szervergépekhez, gigabájtos méretű memóriával. Leginkább webes alkalmazások fejlesztésére használják. Fontos eleme a servlet, amely egy olyan kis java alkalmazás, amely egy kliens felőli kérést hivatott kiszolgálni. Fontos még megemlíteni a JSP-t (Java Servlet Pages), amely hasonlóan az ASP vagy PHP nyelvekhez dinamikus oldalak előállításában vesz részt

2.1.3. Java ME (Micro Edition)

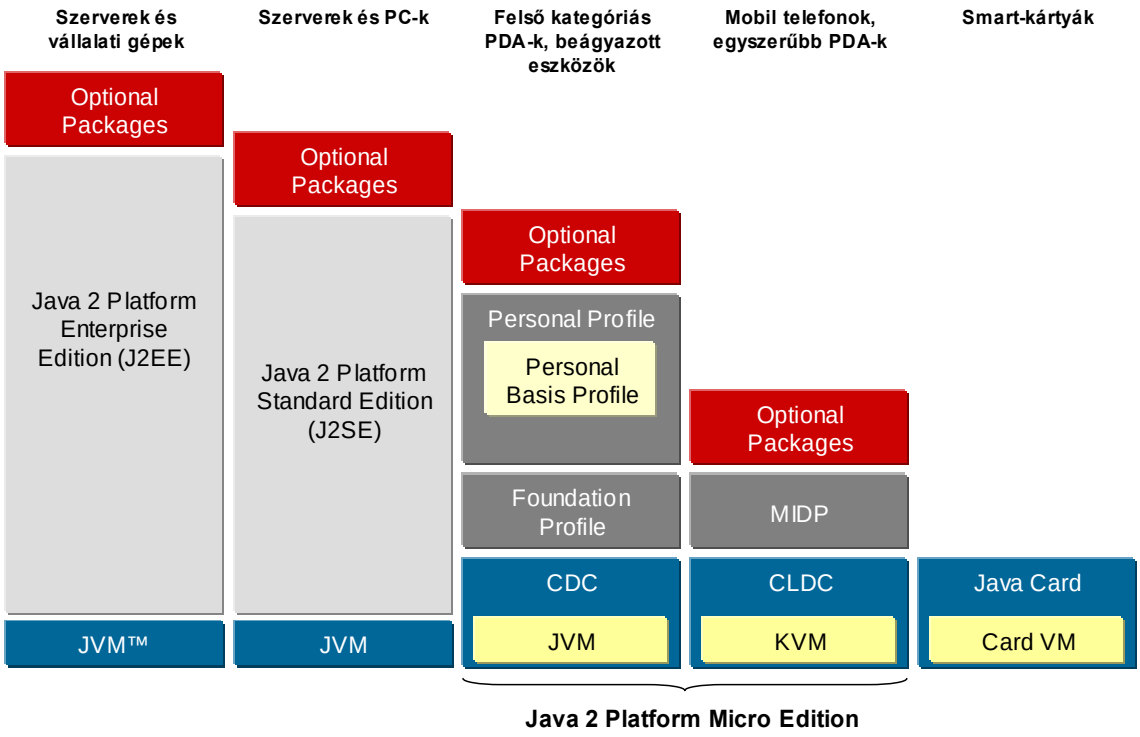
A Java Micro Edition-t elsősorban telepes üzemű, kis kijelzőjű, korlátozott beviteli lehetőségekkel és processzor teljesítménnyel rendelkező eszközökre fejlesztették ki (telefonok, PDA-k, személyhívók, stb.). A J2ME az ugyanilyen célból készült PersonalJava utódja (teljes mértékben le is váltotta azt). A J2ME virtuális gépe a KVM (Kilobyte Virtual

Machine), mely pár kilobájtos méretű, kifejezetten a mobil környezethez íródott, és igényeknek megfelelően modulárisan bővíthető.

A J2ME profilban az alkalmazások kezeléséért és telepítéséért a JAM (Java Application Manager) felelős. Az egyes alkalmazásokat jar fájlok képében telepíthetjük (ez lényegében egy ZIP fájl, amely tartalmazza az alkalmazásban szereplő osztályok bájtkódját, illetve információkat a JAM számára).

A J2ME platform biztosítja a Java technológia előnyeit e fenti eszközökön – rugalmas felhasználói interfész, robosztus biztonsági modell, hálózati protokollok széles skálája, valamint hálózati és offline alkalmazások támogatása.

2.2. A J2ME architektúra



1. ábra. A J2ME architektúra

A J2ME architektúrában különböző konfigurációkat, profilokat és opcionális csomagokat definiáltak. Ezen elemekből épül fel a teljes Java futtatókörnyezet (JRE). Minden egyes kombináció memóriára, processzor kapacitásra, illetve I/O műveletekre optimalizált – attól függően, hogy az adott eszköz melyik hardver-kategóriába tartozik.

2.2.1. Konfigurációk

A konfigurációkban egy virtuális gépet, illetve minimális könyvtárhalmazt definiáltak. Ezek biztosítják az egy kategóriába tatozó eszközök számára az olyan alapfunkcionalitásokat, mint a hálózati kapcsolat és memóriakezelés. Jelenleg két J2ME konfiguráció létezik: a CLDC (Connected Limited Device Configuration) illetve a CDC (Connected Device Configuration)

2.2.1.1. Connected Device Configuration (CDC)

Mivel egy J2ME alkalmazás mind konfigurációt, mind valamilyen profilt igényel, a CDC specifikációjakor a profilokat helyezték előtérbe. Vagyis minden CDC profil implementáció CDC osztálykönyvtár (class library) implementációt és Java virtuális gépet tartalmaz. Az opcionális csomagokat külön lehet letölteni attól függően, hogy milyen speciális profilt használunk.

- *CDC Class Library*
    - A CDC osztálykönyvtárak a J2SE-ből származnak, de a mobil eszközök korlátozásait figyelembe veszik.
    - A legtöbb CDC API megegyezik a neki megfelelő J2SE API-val, a környezetre alkalmazott implementáció a mobil eszköz memória és processzorkapacitására lett hangolva. Így a J2SE-re írt alkalmazások könnyen portolhatók CDC környezetre.

- *CDC Profilok*
    - Foundation Profile

- A legalapvetőbb CDC profil. Tartalmazza a legalapvetőbb alkalmazás-támogató osztályokat, mint pl. hálózat és I/O. Nem tartalmaz grafikus, illetve GUI szolgáltatásokat.
- Personal Basis Profile
- Pehelysúlyú komponenseket, illetve Xlet alkalmazásokat támogat. A Personal Basis Profile tartalmazza a teljes Foundation Profile API-t.
- Personal Profile
- Teljes AWT és korlátozott bean-támogatás. Ezenkívül a PP tartalmazza a teljes PBP API-t.

- *Opcionális csomagok*
  - RMI
  - JDBC

2.2.1.2. Connected Limited Device Configuration (CLDC)

Követelmények

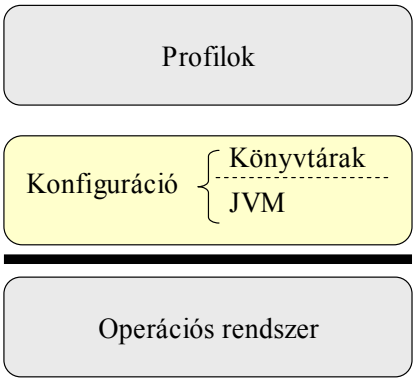
- Hardware követelmények
  - Legalább 160 kByte nem-felejtő, a JVM számára elérhető memória
  - Minimum 32 kByte felejtő a JVM számára futás alatt elérhető memória (pl. heap)

- Szoftver követelmények
  - Az eszközön fut operációs rendszer, mely kezeli a hardvert
  - Az operációs rendszer biztosít legalább egy ütemezhető egységet a JVM számára
  - Az operációs rendszernek nem szükséges támogatnia az elkülönített névtereket vagy folyamatokat és a real-time ütemezést.

- J2ME követelmények
    - Egy J2ME konfiguráció a Java technológia minimum-halmaza. Minden konfigurációban definiált tulajdonság általánosan alkalmazható kell, hogy legyen az eszközök széles skáláján. A további tulajdonságokat a profilokban definiálják.
    - Mivel a konfiguráció célja a hordozhatóság és az interoperabilitás, a konfiguráció nem definiálhat opcionális tulajdonságokat.
    - A J2ME konfiguráció értelemszerűen a Java technológia könyvtárainak és tulajdonságainak egy részhalmaza. A teljes specifikáció helyett a CLDC azt definiálja, hogy miben tér el a J2SE-től.

- A specifikáció által lefedett területek
    - Java nyelv és virtuális gép
    - A fő java könyvtárak (`java.lang.*, java.util.*`)
    - I/O (`java.io.*`)
    - Biztonság
    - Hálózat

CLDC architektúra



2. ábra. A CLDC architektúra

Java alkalmazás

A CLDC nem céloz meg semmilyen specifikus eszköz kategóriát. Sok eszköznek fejlett grafikus interfésze van, mások csak karakteres kijelzésre alkalmasak, megint más eszközöknek egyáltalán nincs látható felhasználói interfészük. Ezért a CLDC nagyon egyszerű.

JVM eltérése J2SE-től

A következő tulajdonságokat teljes mértékben eltávolítottak a CLDC virtuális gépből:

- Felhasználó által definiált osztálybetöltők
- Szálcsoportok és démonszálak
- Példányok véglegesítése
- Aszinkron kivételek

2.3. Connected Device Configuration (CDC)

Mivel egy J2ME alkalmazás mind konfigurációt, mind valamilyen profilt igényel, a CDC specifikációjakor a profilokat helyezték előtérbe. Vagyis minden CDC profil implementáció CDC osztálykönyvtár (class library) implementációt és Java virtuális gépet tartalmaz. Az opcionális csomagokat külön lehet letölteni attól függően, hogy milyen speciális profilt használunk.

2.3.1. CDC Class Library

A CDC osztálykönyvtárak a J2SE-ből származnak, de a mobil eszközök korlátozásait figyelembe veszik.
A legtöbb CDC API megegyezik a neki megfelelő J2SE API-val, a környezetre alkalmazott implementáció a mobil eszköz memória és processzorkapacitására lett hangolva. Így a J2SE-re írt alkalmazások könnyen portolhatók CDC környezetre.

2.3.2. CDC Profilok

Foundation profile:

A legalapvetőbb CDC profil. Tartalmazza a legalapvetőbb alkalmazás-támogató osztályokat, mint pl. hálózat és I/O. Nem tartalmaz grafikus, illetve GUI szolgáltatásokat.

Personal basis profile:

Pehelysúlyú komponenseket, illetve Xlet alkalmazásokat támogat. A Personal Basis Profile tartalmazza a teljes Foundation Profile API-t.

Opcionális csomagok:

- RMI
- JDBC

Az egyes profilok által támogatott Java csomagok

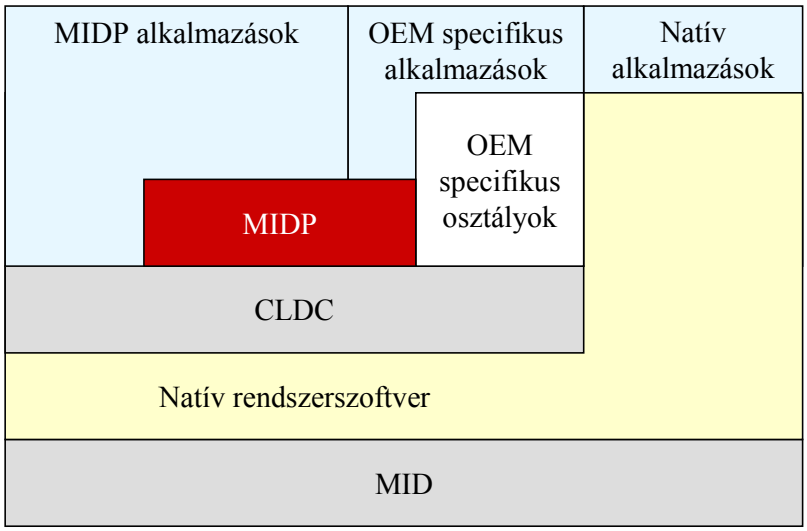
Package	J2SE	FP 1.0	PBP 1.0	PP 1.0
`java.applet`	+	–	–	részleges
`java.awt.*`	+	–	részleges	részleges
`java.beans.*`	+	–	részleges	részleges
`java.io`	+	+	+	+
`java.lang.*`	+	+	+	+
`java.math`	+	részleges	részleges	+
`java.net`	+	+	+	+
`java.rmi.*`	+	opcionális	opcionális	opcionális
`java.security.*`	+	+	+	+
`java.sql`	+	opcionális	opcionális	opcionális
`java.text`	+	+	+	+
`java.util.*`	+	+	+	+
`javax.accessibility`	+	–	–	–
`javax.naming.*`	+	–	–	–
`javax.rmi.*`	+	–	–	–
`javax.sound.*`	+	–	–	–
`javax.swing.*`	+	–	–	–
`javax.transaction`	+	–	–	–
`org.omg.*`	+	–	–	–
`javax.microedition.io.*`	–	+	+	+
`javax.microedition.xlet.*`	–	–	+	+

2.4. Mobile Information Device Profile (MIDP)

A Mobile Information Device Profile (MIDP) a CLDC-vel együtt a Java™ mobil eszközökre való futtatókörnyezete (JRE). CLDC és MIDP együtt olyan dinamikus és biztonságos platformot definiál, mely alkalmas magas szintű grafikus, hálózatot kezelő alkalmazás mobil eszközre történő fejlesztésére. A MIDP tartalmazza azokat az alapvető funkcionalitásokat, melyeket a szabványos Java környezetben futó mobil alkalmazások igényelnek. MIDP segítségével olyan alkalmazások írhatók, melyek a korszerű mobil eszközökre optimalizáltak.

A MIDP lehetővé teszi hálózati alkalmazások futtatását is. MIDP alkalmazás letöltéséhez a felhasználó kiválasztja a webszerveren a letöltendő alkalmazást. Ezután a mobil eszköz letölti az alkalmazást, verifikálja és lefordítja bájtkódra és elindítja. A grafikus felhasználói interfész a kisméretű kijelzőre, illetve a bemenetet megvalósító és más natív eljárásokra méretezett. A MIDP a telefon billentyűzetének és egyéb gombok teljes kihasználásával biztosítja az intuitív navigációt és adatbevitelt. A MIDP alkalmazások kapcsolat nélkül is képesek futni, a mobil eszköz biztonságosan tárolja, és helyben kezeli az adatokat.

2.4.1. MIDP architektúra



3. ábra. A MIDP architektúra

2.4.1.1. A mobil eszközzel szemben támasztott hardver követelmények

Kijelző: 96x54, színmélység 1 bit, közel négyzet alakú pixel

- Legalább egy a következő felhasználói input eszközökből: egykezes billentyűzet, kétkezes billentyűzet, érintőképernyő
- 256 kB nem felejtő memória a MIDP-nek (+ mai a CLDC-nek kell), ezenkívül 8 kB memória az alkalmazásoknak
- 128 kB felejtő memória (heap)
- Hálózat: kétirányú rádiós csatorna, korlátozott sávszélesség
- Hang: hangok lejátszására alkalmas HW eszköz vagy valamilye SW algoritmus

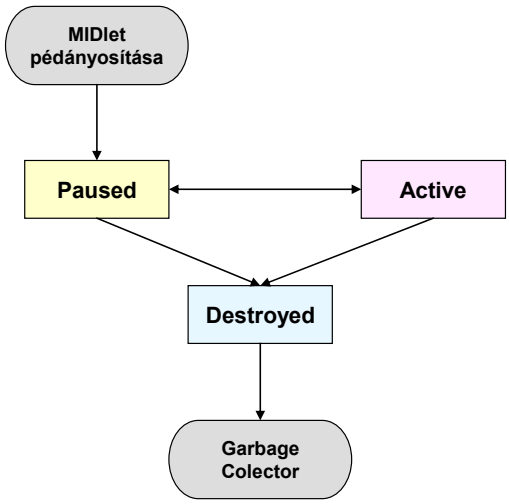
2.4.1.2. A mobil eszközzel szemben támasztott szoftver követelmények

- A fenti hardver tulajdonságokkal rendelkező eszközök még mindig nagyon sokféle szoftverfuttatási lehetőséggel rendelkezhetnek. a MID környezet eszközeinek szoftverlehetőségei nagyon eltérhetnek egymástól. Pl. néhány mobil eszköz teljes operációs rendszerrel, multiprocesszing támogatással és hierarchikus fájlstruktúrával rendelkezik, míg más eszközök kis, szálalapú operációs rendszerrel rendelkeznek, fájlrendszer nélkül A szoftverben is követelményeket kell tehát támasztanunk az eszközzel szemben. Ezek a következők:
- minimális kernel, amely kezeli a hardvert (megszakítások, kivételek, minimális ütemezés.). A kernel képes legyen futtatni legalább egy virtuális gépet.
- biztosítsa a nem felejtő memóriából a való olvasás, illetve az oda történő írás lehetőségét
- olvasás és írási hozzáférés az eszköz rádiós hálózati kapcsolatán keresztül
- időkezelés
- minimális bitmap megjelenítése a grafikus kijelzőn
- legalább 1 input kezelése az előző fejezetben említettek közül
- az alkalmazás életciklusának kezelése

A MIDP környezetben futó Java alkalmazásokat *MIDlet*eknek nevezzük. Ha alkalmazás töltünk le a webről, akkor nem a MIDletet töltjük le és indítjuk, hanem egy ún. *MIDlet suit*-ot, ami egy vagy több MIDlet-et tartalmaz összecsomagolva. A MIDlet suite többnyire több, hasonló funkciót ellátó vagy együttműködő MIDlet összessége. Az egy suite-ban lévő MIDletek osztozhatnak az erőforrásokon (adat, grafika) ugyanabban a suite-ban lévő MIDletek hozzáférhetnek egy suite-beli MIDlet információihoz, míg más suite-ban lévő MIDletek erőforrásaihoz nem.

2.4.2. A MIDlet életciklus modellje

A MIDP API javax.microedition.midlet csomagja a midletek életciklusával foglalkozik.



4. ábra. A MIDlet életciklus modellje

A MIDlet életciklusának három állapota van. Állapotváltozásokat kezdeményezhet maga az alkalmazás a MIDlet osztálytól örökölt metódusainak meghívásával, vagy külső események hatására az AMS értesíti a midletet az életciklus-változásokról. Ekkor szintén a megfelelő metódus hívódik meg.

A midlet a telepítéskor destroyed állapotba kerül. Ekkor nincs betöltődve a KVM-be. Az alkalmazás indításakor az AMS meghívja a midlet startApp() metódusát, és az aktív állapotba kerül. A futás szüneteltetésekor a pauseApp() hívódik meg, és az alkalmazás felfüggesztett állapotba kerül. A szüneteltetés magától is bekövetkezhet például egy bejövő

telefonhívás alkalmával, vagy ha új üzenet érkezik. Bár ebben a kérdésben a készülékgyártók MIDlet implementációi különböznek egymástól. Van ugyanis olyan implementáció, amely az előbb említett események hatására nem függeszti fel a működést, csak úgy veszi, hogy az alkalmazás képernyőjét egy másik képernyő eltakarja. Amikor az alkalmazást befejezzük, mielőtt az AMS törölné a midletet, meghívja a destroyApp() metódust. Ha ennek a paramétere nem igaz, akkor a midlet megakadályozhatja a befejeztetést, ha igaz, akkor nem.

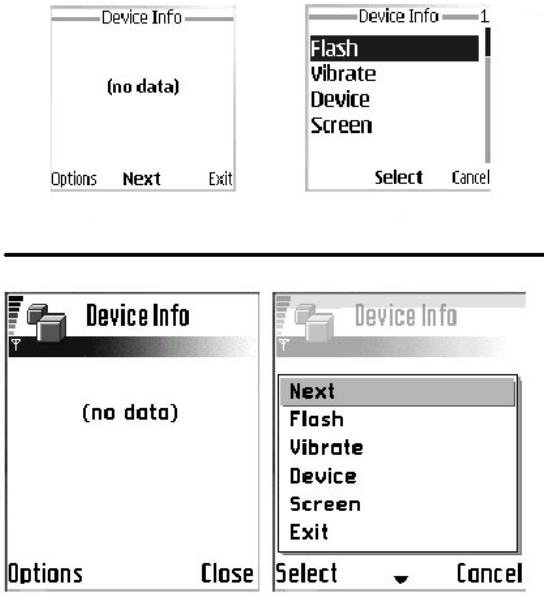
2.4.3. Felhasználói felületek

A javax.microedition.lcdui a felhasználói felületekért felelős csomag. A kis méret- és erőforrásigény elérése érdekében a KVM nem támogatja az olyan létező felhasználói felület API-kat, mint az AWT és a Swing. Ezek nemcsak nagy méretük miatt alkalmatlanok a mobiltelefonos használatra, hanem azért is, mert a mobil eszközök eltérő megközelítésből indulnak ki a felhasználói felületekkel kapcsolatban. Ezért ezek helyett egy új, a kis memóriával rendelkező eszközökre illeszkedő, és az azok különbözőségeiből fakadó követelményeknek is megfelelő API-t hoztak létre. Az LCDUI központi fogalma a képernyő.

Az alkalmazások futása során a felhasználó képernyőről-képernyőre halad. A kijelzőn mindig az aktuális képernyő látható, amit a kijelző objektum (Display) setCurrent(Displayable d) metódusával állíthatunk be. A MIDP API-ban képernyőt a Displayable osztály reprezentálja. A Displayable két közvetlen kiterjesztése a Canvas és a Screen osztály.

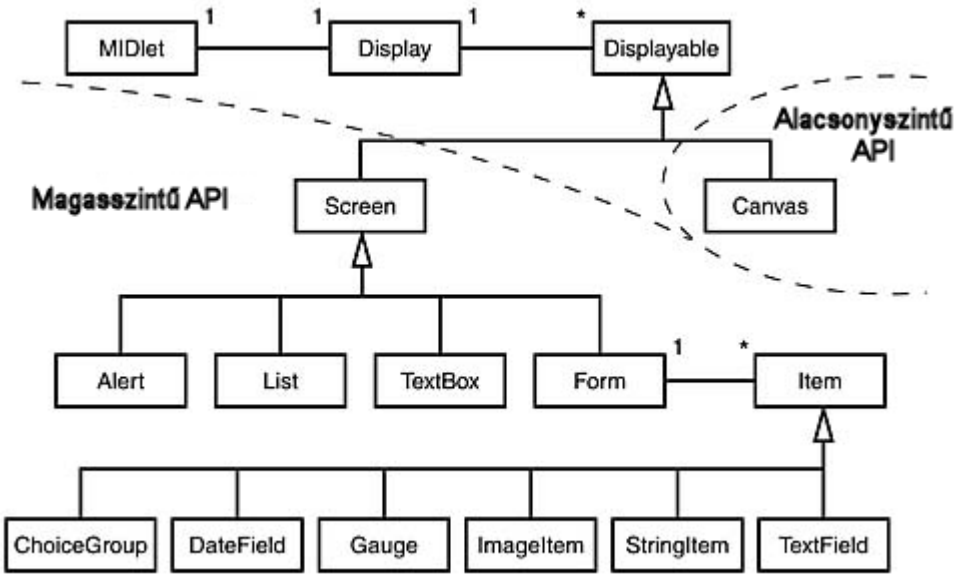
A Canvas és leszármazottjai az alacsonyszintű felhasználói felületek, amelyek teljes hozzáférést biztosítanak a kijelzőhöz. A Canvas osztály definiál egy paint(Graphics g) absztrakt metódust, amelyet a leszármazottaknak implementálniuk kell, és amely a képernyő kirajzolását végzi a Graphics objektumon keresztül. A kijelzőnek ez a fajta programozása nagyon fontos például játékok fejlesztésénél. A sudoku játékban a táblát megjelenítő képernyő (SudokuCanvas) is a Canvas leszármazottja.

A Screen osztály a magasszintű felhasználói felületeket megvalósító osztályok őse. A magasszintű azt jelenti, hogy a képernyőtartalmak elemeinek a kirajzolása a telefon szoftverének a feladata, az határozza meg az elemek méretét, színét, helyét, stb. A programozó dolga az, hogy a képernyőn megjelenítendő tartalmat megadja. Így ugyanaz a képernyő különböző eszközökön különbözőképpen jelenhet meg.



5. ábra. Két különböző megjelenítés

A Screen osztályból származik közvetlenül az Alert, a List, a TextBox, amelyek egy egész képernyőt foglalnak el, tartalmuk kötött, szerkezetüket nem lehet megváltoztatni és a Form osztály, amely űrlapok gyors és egyszerű készítését teszi lehetővé. Egy Formon különböző elemeket helyezhetünk el, mint például képek, szöveg elemek, szövegbeviteli mezők vagy választási lehetőségek. Ezek az elemek az Item osztály leszármazottjai.



6. ábra. A képernyők osztályhierarchiája

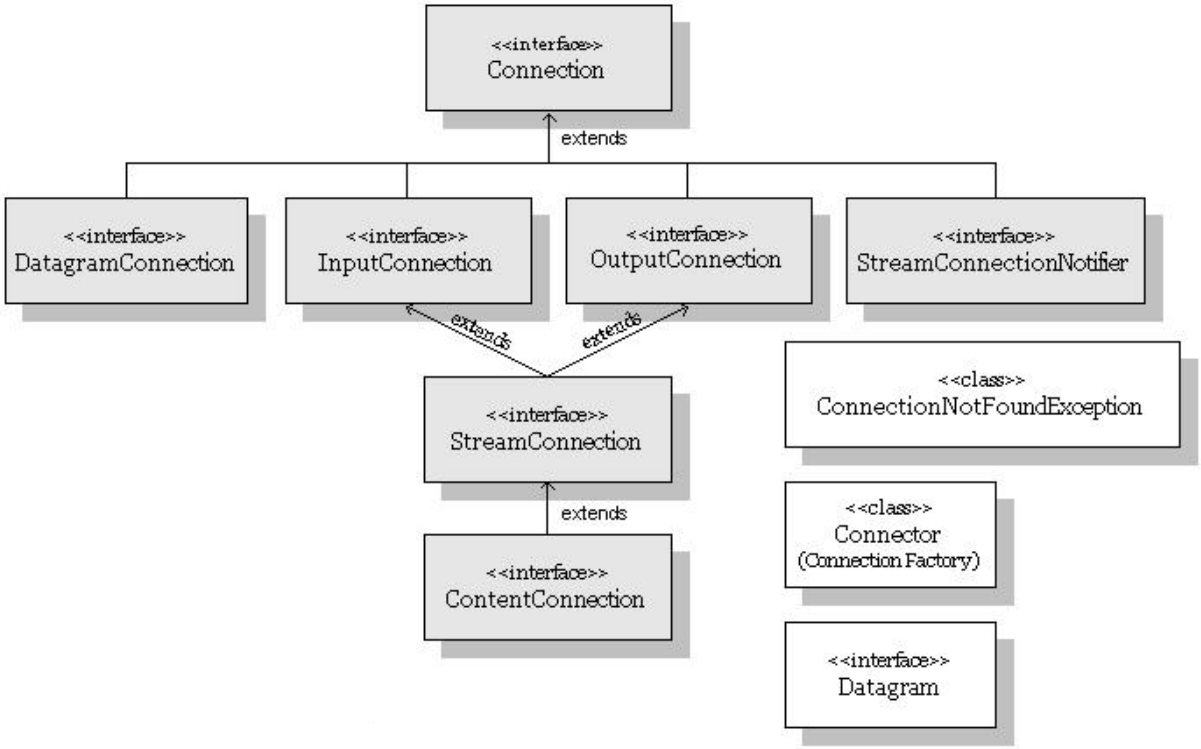
Az alacsony és magasszintű API-k nemcsak a tartalom megjelenítés módjában különböznek, hanem az események kezelésében is.

- Az alacsonyszintű API hozzáférést enged a billentyűzethez, azaz különböző gombnyomás eseményekhez eseménykezelőt tudunk rendelni. Egy alacsonyszintű esemény bekövetkezésekor egy, az eseménynek megfelelő metódus hívódik meg, amelyben az eseményeket kezelhetjük. Ilyen metódus például a keyPressed(int keyCode) vagy a keyReleased(int keyCode). A keyCode az eseményt kiváltó gomb kódját jelenti, amelyek a Canvas osztályban statikus adattagokként szerepelnek.

- A magasszintű felhasználói eseményeknek két fajtája van:
    - Egy Item típusú elem állapotának megváltozása. Ezek figyelését az elemhez rendelhető, egy az ItemStateListener interfészt implementáló példány végzi. Az esemény bekövetkezésekor a példánynak az eseménykezelő interfésztől örökölt itemStateChanged(Item item) metódusa hívódik meg. Az Item típusú paraméter az eseményt kiváltó elemet hivatkozza.
    - Egy Command esemény bekövetkezte. Az ilyen típusú események Figyelése a CommandListener interfész egy implementációjának a feladata, amelynek meg kell valósítania az interfész CommandAction(Command c, Displayable d) metódusát, ami esemény bekövetkeztével hívódik meg. A paraméterek itt is az eseményt kiváltó parancs és az azt tartalmazó képernyőt jelenti.

2.4.4. Hálózatok kezelése

A javax.microedition.io csomag a Generic Connection keretrendszert (GCF) valósítja meg. A legegyszerűbb általános kapcsolattípust a Connection interfész írja le. Ezt terjeszti ki a csomag többi interfésze, amelyek specializálják a kapcsolat típusát.



7. ábra. Connection interfész hierarchia

A MIDP négyféle kapcsolattípust támogat:

- Http
- Datagram
- Socket
- Comm (logikai soros port kapcsolat)

Ezek közül a Http kötelezően támogatott típus.
Egy kapcsolat megnyitása a Connector osztály open() metódusával történik. Ennek a kötelező paramétere egy String, ami a cél URL-t reprezentálja a kapcsolattípusnak megfelelően. Visszatérési értéke egy megfelelő kapcsolatobjektum.

2.4.5. Perzisztens tárolás

A MIDP egy egyszerű lehetőséget kínál adatok tárolására és visszatöltésére az alkalmazás leállítása és újraindítása után is. Ennek a megvalósítása javax.microedition.rms csomagban érhető el. Az RMS a Record Management System rövidítése. Az RMS kezeli a tárolt adatokat, ami egy rekordorientált adatbázis modellen alapul. Használatakor névvel ellátott tárolókat (RecordStore) hozhatunk létre, amelyek a MIDlet készlethez kötődnek és ennek törlésekor maguk is törlődnek.

A rekordtárolóhoz a MIDlet készlet tagjai férhetnek hozzá, más MIDlet készlet tagjai nem. Ezekhez rekordként adhatunk adatokat és olvashatunk belőlük. A rekordok bájt tömbök, amelyek számozva kerülnek a tárolóba. Kiolvasásuk történhet sorszámuk alapján, vagy kérhetjük a rekordok felsorolását a RecordEnumeration segítségével.

A mobiltelefonok erre a célra néhány tíz kilobájt és több megabájt között változó méretű területet biztosítanak a kategóriáktól függően.

3. A Sudoku

3.1. Játékszabály

A Sudoku egy 9 × 9 cellából álló rács. A rács kilenc kisebb, 3 × 3-as blokkra oszlik, amelyben elszórva néhány 1-től 9-ig terjedő számot találunk. Az üresen maradt cellákat a játékosok töltik ki saját (ugyancsak 1-től 9-ig terjedő) számaikkal úgy, hogy minden vízszintes sorban, függőleges oszlopban, és 3 × 3-as blokkban az 1-től 9-ig terjedő számok pontosan egyszer szerepeljenek.

	5	9		2		6		8
	2	8	6			4	3	
6		1	8	9	4	5	7	2
5	1		2	3		8		6
			1	4			5	7
3	8		5	6	7			
					6	9		4
9	6	3	4	7	2		8	5
8	4			1				3

8. ábra. Egy sudoku tábla

3.2. Története

Latin négyzet-nek nevezik az olyan NxN-es négyzetet, aminek minden sorában és oszlopában az 1-N számok egy-egy permutációja áll. Mivel ezzel Euler foglalkozott sokat, vannak, aki tőle származtatják a sudoku-t.

A kiegészítő szabályt egy nyugdíjas amerikai építész, Howard Garns találta ki, 1979-ben. Egy New York-i rejtvény újságban közölt néhány rejtvényt "Number Place" néven. (Abban az évtizedben, mint Rubik a kockát)

1984-ben a "Nikoli" nevű japán rejtvény társaság átvette a rejtvényt, és a "sudoku" elnevezést adta neki. Japánban azóta töretlen a népszerűsége. Több folyóirat csak ezzel foglalkozik, és azt állítják, hogy ők még mindig kézzel csinálják a rejtvényeket.

2004 végén az új-zélandi származású, hong-kongi Wayne Gould ajánlotta a számítógéppel készített rejtvényeit néhány neves angol újságnak, akik "kipróbálták", és európai siker lett belőle, sőt Amerikába is visszatért a játék.

Howard Garns nem érte meg a nagy sikert, egyik forrás szerint 1981-ben, másik szerint 1989-ben meghalt.

3.3. Matematikája

Egy konkrét N-re nyilvánvalóan véges sok sudoku kitöltés létezik. A 4x4-es esetben ezeket programmal gyorsan elő lehet állítani, a 9x9-es esetben (és a még nagyobb N-kre) számítógépnek is sok.

Bertram Felgenhauer és Frazer Jarvis sheffieldi matematikusok programmal kiszámították, hogy 6670903752021072936960 különböző helyes (9x9-es) sudoku kitöltés létezik.

Az alábbi - lényegesen különböző - transzformációkkal lehet jó sudoku kitöltésből másik jót készíteni:

- A kilenc számjegy permutációja;
- A mátrix transzponálása (sor-oszlop csere);
- A sorok permutálása egy 3x3-as blokkon belül;
- Az oszlopok permutálása egy 3x3-as blokkon belül;
- A 3x3-as sor-blokkok permutálása;
- A 3x3-as oszlop-blokkok permutálása.

(A "lényegesen különböző" azt jelenti, hogy pl. a forgatások, tükrözések az előzőkben benne vannak.)

Ha ezt figyelembe vesszük, akkor kiderül, hogy 5 472 730 538 lényegesen különböző kitöltés létezik. Ezzel még el lesz egy darabig az emberiség.

Ez azért meglepő csökkenés, de vegyük figyelembe, hogy "a kilenc számjegy permutációja" egyetlen kitöltésből 362879 (9!-1) különböző másikat eredményez. És ezek mindegyikére végrehajthatók a fenti további műveletek!

Ha úgy határozzuk meg a rejtvényt, hogy csak egy megoldása lehessen, akkor nem tudjuk, hogy minimum hány négyzetnek kell kitöltve lenni egy rejtvényben. Gordon Royle ausztrál matematikus már 35396 olyan lényegesen különböző egy megoldású sudoku rejtvényt halmozott fel, amiben 17 mező van kitöltve. Olyan rejtvényt még senki nem talált, amiben 17-nél kevesebb mező van kitöltve, és egy megoldása van.

Sudoku rejtvény készítésénél célszerű először egy teljes megoldást készíteni, és ezután kitörölni belőle bizonyos mezőket (pl. nehézségi szintnek megfelelően). Az első elem meghatározása könnyű, a kilenc számjegy közül bármelyiket választhatjuk. Az összes többi elemnél figyelembe kell vennünk, hogy melyik mezőkkel van függésben (pl. a második elem felvehető értékei csak az első mező értékétől függenek). Az utolsó sor/oszlop, és a 3x3-as dobozok értékei egyértelműen meghatározhatóak. Természetesen elég nagy a valószínűsége annak, hogy ha elérkezünk egy adott mezőhöz, az előzőleg kitöltött mezők alapján nem vehet fel már semmilyen értéket. Ekkor vissza kell lépnünk az előző mezőhöz, és egy másik számjegyet választanunk értékül.

A témával bővebben a program megvalósításánál foglalkozom.

4. Alkalmazás specifikációk

4.1. Áttekintés

4.1.1. Általános leírás

Az alkalmazás a sudoku játékot valósítja meg mobil környezetben. A játékot egy időben egy személy játszhatja.

4.1.2. Általános követelmények

- A játék a sudoku (9x9) szabályai szerint történik. Egy 81 elemű (9x9) táblázat előre nem megadott mezőit lehet kitölteni.

- Lehetőség van új játék kezdésére.

- A játékot menteni lehet, és később folytatni. A játékból való kilépéskor, vagy a játék szüneteltetésekor (pl. telefonhívás) a játék elmentődik.

- Lehetőség van nehézségi szint beállítására.

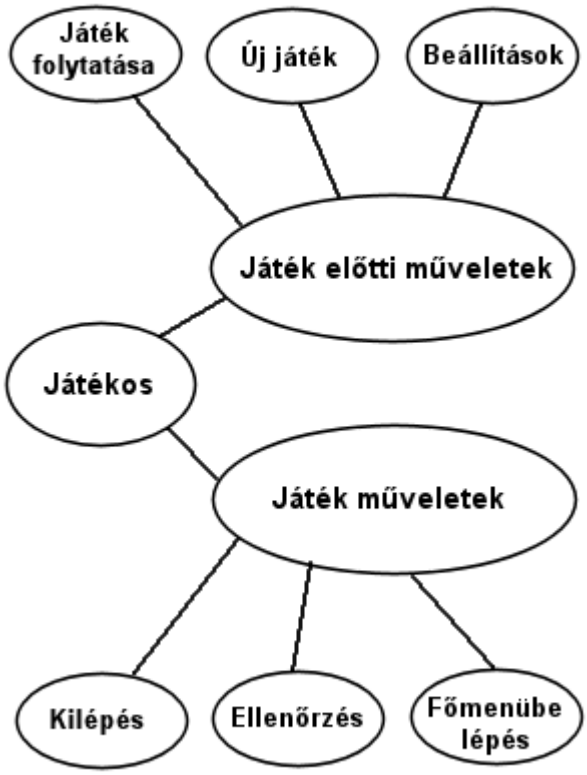
- A programban meg lehet nézni az elkezdett játékok számát, és a játékszabályt is.

4.1.3. Rendszerkövetelmények

- A rendszer platformfüggetlen.
- Programozási nyelv: Java
- Célhardver: CLDC1.0 / MIDP1.0 képes eszköz (elsősorban mobileszköz)
  Minimum 256 színű kijelző javasolt, de lehet szürkeskálás is.

4.2. Felhasználói esetek

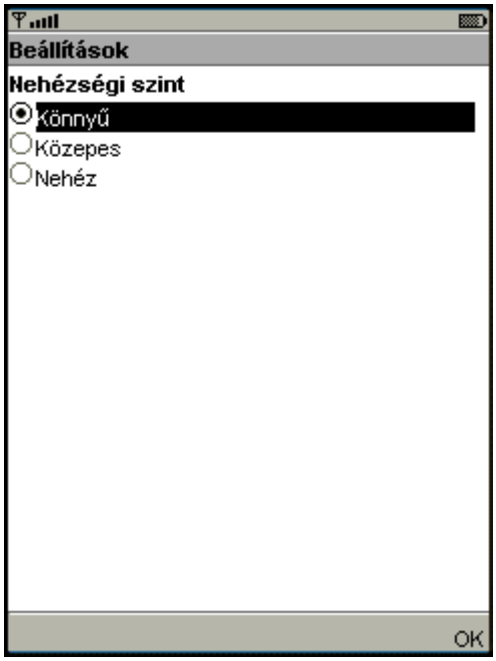
- Játék előtti műveletek
  - Játék folytatása
  - Új játék
  - Beállítások
- Játék műveletek
  - Kilépés + mentés
  - Ellenőrzés
  - Főmenübe lépés + mentés



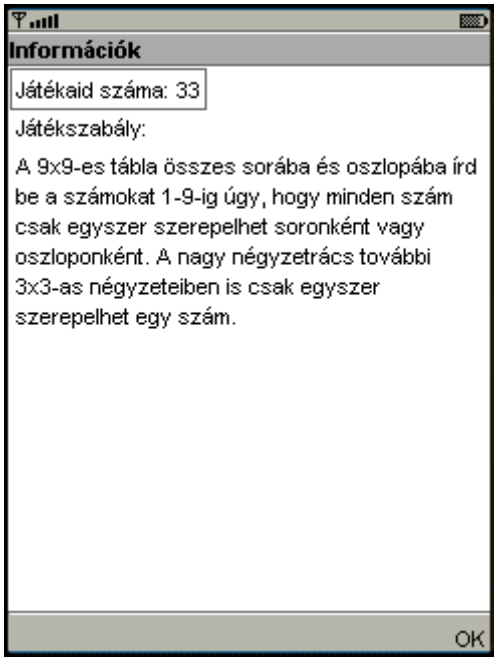
9. ábra. Felhasználói esetek

4.3. Felhasználói felületek

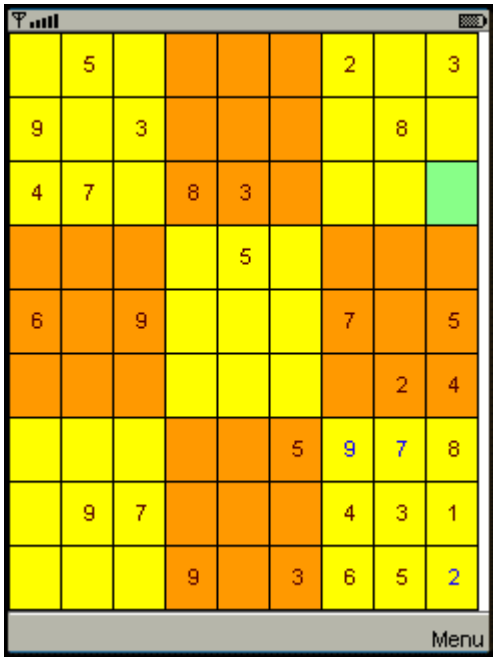
főmenü



beállítások



információk



játék közben

5. Alkalmazás architektúra

Az alkalmazás kapcsolatot tart a felhasználóval, a felhasználó által végzett műveleteket információvá alakítja.

Két fő részből épül fel:

- Felhasználói interfész
- Adattárolás

A felhasználói interfész megjeleníti a felhasználó számára a sudoku táblát a játék aktuális állásával. Ezen kívül további információkat is ad a játékról. Figyeli a játékos lépéseit, és kirajzolja a változásokat.

Az adattárolási rész az RMS segítségével a rekordtárolóba való adatmentésért és az adatok szükség esetén történő visszatöltéséért felel. A nehézségi szintet és az elkezdett játékok számát is ide kell eltárolni. Egy játék mentésekor a sudoku táblát leíró információk is ide kerülnek, és a játék folytatásakor innen töltődnek be.

6. A játék megvalósítása

Az alkalmazás egy MIDP alkalmazás, fő osztálya a Sudoku osztály, mely a MIDlet osztály leszármazottja. Továbbá a Sudoku osztály implementálja a CommandListener interfészt. Ezzel kezeli a magas szintű eseményeket, mellyel meghatározza a parancsok viselkedését.

A SudokuCanvas (A Canvas osztály leszármazottja) osztály valósítja meg a tábla megjelenítését. A SudokuCanvas alacsonyszintű eseménykezelő és alacsonyszintű felhasználói felület.

A Table osztály egyik feladata a sudoku tábla értékeinek generálása, újragenerálása, valamint adott számú mező nem láthatóvá tétele a nehézségi foknak megfelelően. Ugyancsak a Table osztály tartja számon annak a mezőnek a koordinátáit, amelyiken adott pillanatban a játékos áll. A játék megoldásának ellenőrzéséért is ez az osztály felel. A tábla egyes mezői TableItem típusú objektumokból állnak.

A TableItem osztályban tárolódnak az adott mezők tulajdonságai, továbbá a mező értékének meghatározására szolgáló metódusok.

6.1. A MIDlet

A Sudoku példányosításakor létrejönnek az alkalmazásban használt parancspéldányok, a megjelenítendő táblázat egy példánya, és a képernyő. Alkalmazásindításkor a Sudoku osztály startApp() metódusa hívódik meg. A startApp() metódus a főmenüt teszi az aktuális képernyővé.

Ezenkívül megnyitásra kerülnek a beállítások és a mentett játék rekordjai, és betöltődnek a beállítások.

6.2. A főmenü

A List segítségével egyszerűen és gyorsan lehet menüket létrehozni:

```
mainMenu = new List("Sudoku", Choice.IMPLICIT);
mainMenu.append("Játék folytatása", null);
mainMenu.append("Új játék", null);
mainMenu.append("Beállítások", null);
mainMenu.append("Információk", null);
mainMenu.append("Kilépés", null);
```

10. ábra. A főmenü megvalósítása

A főmenü öt menüpontból áll:

- **Játék folytatása**
  Ebből a pontból lehet folytatni a mentett játékot. Amennyiben nem volt még mentve játék (pl. első játék), akkor jelzi a felhasználó felé ezt.

- **Új játék**
  Innen indíthat új játékot a felhasználó. Ezt a menüpontot választva egy az aktuális feladvány táblázata jelenik meg a képernyőn.

- **Beállítások**
  A beállítások pont a nehézségi fokozat beállítására szolgál. Három nehézségi fokozat közül választhat.

- **Információk**
  Itt tekinthető meg az elkezdett játékok száma, továbbá a játékszabály leírása is itt érhető el.

- **Kilépés**
  Az alkalmazásból való kilépésre szolgál. Kilépéskor az aktuális feladvány mentésre kerül.

6.3. A játék menete

A játék során az aktuális képernyő a SudokuCanvas. Ez csak ellenőrzéskor változik meg, amikor valamilyen üzenet jelenik meg a képernyőn.

6.3.1. Új játék indítása

Új játék indításakor növeljük az elkezdett játékok számát, majd elmentjük a beállításokat, amely ezen adatot is tartalmazza. Ezután generálunk egy új feladványt, és a SudokuCanvas lesz a képernyő.

6.3.1.1. A feladvány generálása

A SudokuCanvas tartalmazza a Table osztály egy példányát (table), melyben a táblázatot egy 81 elemű (9x9) TableItem típusú tömb tárolja. A choosableFieldIndexes egy Vector típusú objektum, ezt tárolja, mely mezők értékei nem kerültek törlésre a feladvány készítésekor. A Table osztály példányosításakor még nem jönnek létre a táblázat mezői, hanem csak a choosableFieldIndexes vektort töltjük föl, természetesen az összes index-szel.

```
public Table(){
        for(int i = 0; i < 9; i++){
                for(int j = 0; j < 9; j++){
                        int ind[] = new int[2];
                        ind[0] = i;
                        ind[1] = j;
                        choosableFieldIndexes.addElement(ind);
                }
        }
}
```

11. ábra. A Table osztály konstruktora

A feladvány generálására a generateSudoku() metódust használhatjuk. A metódus megvizsgálja, hogy az alkalmazás indítása után indítottunk-e már játékot. Amennyiben igen, akkor alapállapotba állítja a táblázatot. Ezután a Table osztály makeAPuzzle() metódusával

generál egy új megoldást. A metódus bejárja a 81 elemű kétdimenziós tömböt. Amennyiben visszalépés történt, az adott mező felvehető értékeiből „töröljük” a mező aktuális értékét, így későbbi iterációs lépésekben már nem veheti fel. Ha nem történt visszalépés, egy új elemmel tölti fel a kétdimenziós tömböt. Beállítja az elem felvehető értékeit, majd beállít egy véletlen értéket. Amennyiben nem sikerül új értéket beállítani (nincs több felvehető érték), akkor visszalép az előző elemre, egyébként a következő elemre.

```
public boolean makeAPuzzle(){
        boolean backstep = false;
        for(int i = 0; i<9; i++)        {
                for(int j = 0; j<9; j++)        {
                        if(backstep)table[i][j].setAValidNumberFalse(table[i][j].getValue()-1);
                        else{
                                table[i][j] = new TableItem();
                                setValidNumbers(i,j);
                        }
                        backstep = false;
                        if(!table[i][j].setRandomValue()){
                                backstep = true;
                                if(j>1)j-=2;
                                else{
                                        if(i>0){
                                                i-=1;
                                                if(j==0)j = 7;
                                                else j = 8;
                                        }
                                        else return false;
                                }
                        }
                }
        }
        return true;
}
```

12. ábra. A makeAPuzzle metódus

A generálásban használt metódusok:

- <TableItem>.setAValidNumberFalse(): a paraméteréül kapott értéket „törli” a felvehető értékek közül.

- <Table>.setValidNumbers(): a paraméteréül kapott indexű elemnek beállítja a felvehető értékeit.
- <TableItem>.setRandomValue (): beállít egy véletlen értéket az adott elemhez a felvehető értékek közül.

Miután elkészült a teljes megoldás, nem láthatóvá kell tennünk bizonyos elemeket. Ezek lesznek azok a mezők, amiket a játékosnak kell kitöltenie (A többi mező értékét a játékos nem tudja megváltoztatni). Ezt a deleteNumbers() metódus végzi. A nehézségi foknak megfelelően bizonyos számú mezők értékeit véletlenszerűen nem láthatóvá teszi. A könnyű fokozat az alapértelmezett.

Három nehézségi szint van:

- könnyű: 33 elem lesz láthatatlan
- közepes: 43 elem lesz láthatatlan
- nehéz: 53 elem lesz láthatatlan

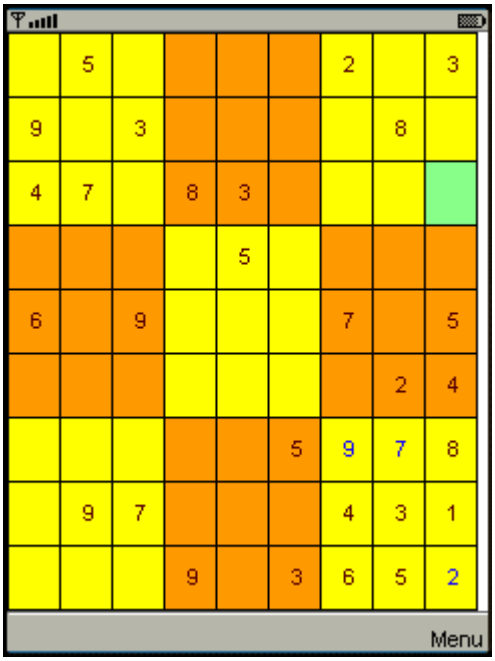
A véletlen koordinátákat a getARandomIndexFromTable() metódustól kapjuk, amely a choosableFieldIndexes vektorból véletlenszerűen választ egy elemet. Egyúttal törli az indexeket a choosableFieldIndexes vektorból, így a következő alkalommal nem kaphatjuk ugyanazokat az indexeket.

A feladvány generálása után a SudokuCanvas lesz a képernyő.

6.3.1.2. A játékfelület

A játékfeületen a 3x3-as dobozok különböző színnel vannak jelölve a könnyebb átláthatóság érdekében. Bordó számjegyek a feladvány generálása után látható mezőkön vannak. A játékos által kitöltött mezőkön a számjegyek kékek. Zöld színnel az a mező van jelölve, amelyik mezőn éppen állunk.

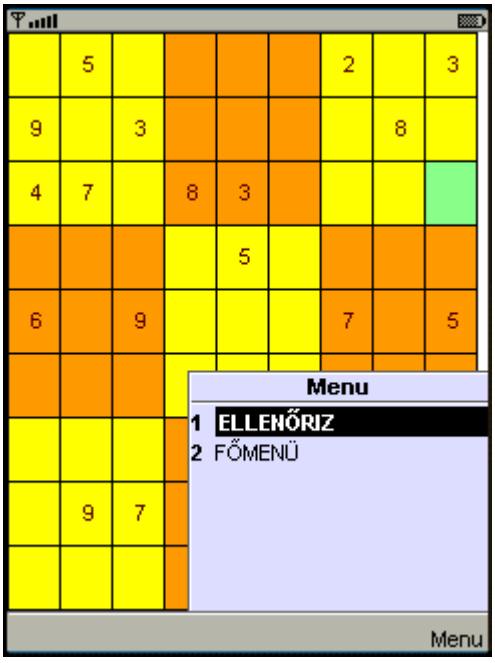
A játékfelület kirajzolása a SudokuCanvas paint() metódusával történik. A metódus a kijelző méretének függvényében rajzolja ki a játékteret.



13. ábra. A játékfelület

A számjegyeket a „billentyűzet” számjegyeivel tudjuk beírni. A 0-s billentyű lenyomásával s törölhetjük az adott elemet, de csak akkor, ha a felhasználó által kitölthető mező az aktuális. A mezőkön lépkedni a készüléken lévő kurzorok segítségével lehet.

6.3.1.3. A megoldás ellenőrzése



14. ábra. Játék közbeni menü

Ha kitöltöttük a feladványt, lehetőségünk van a megoldás helyességének ellenőrzésére. A menüből az ellenőrzés gombot kiválasztva az alkalmazás ellenőrzi a kitöltést. Az ellenőrzésnek három lehetséges kimenetele van:

- Nincs kitöltve az összes mező.
- Helytelen a megoldás
- Helyes a megoldás

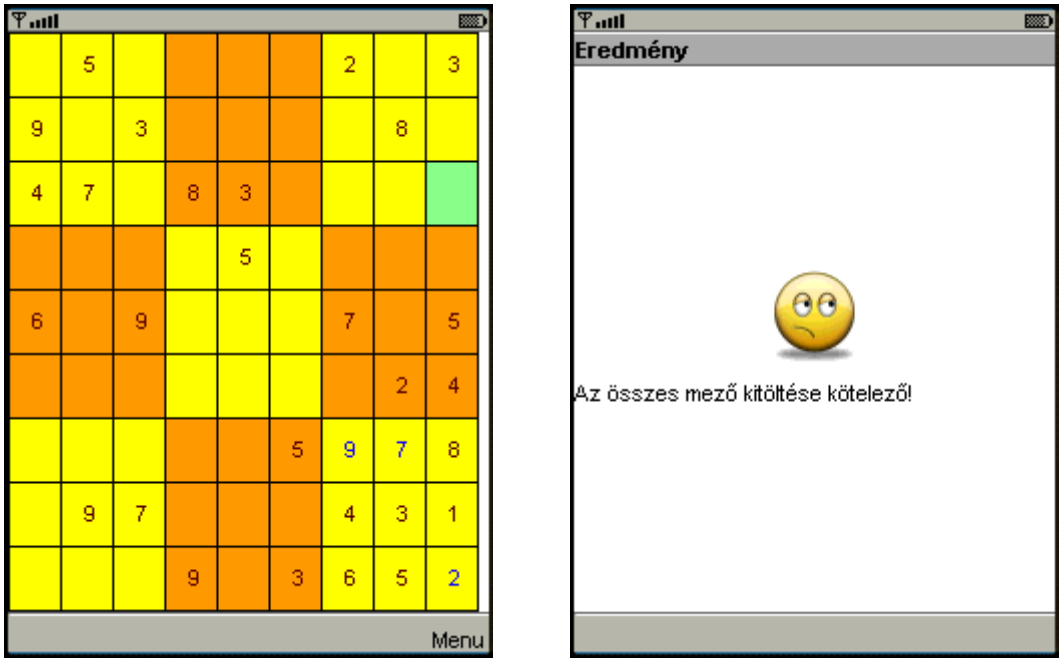
Mindhárom esetben egy üzenet jelenik meg a képernyőn. Üzenetet az Alert osztály segítségével könnyen tudunk eljuttatni a felhasználóhoz. Nagyon egyszerű az üzenethez szöveget vagy akár képet is hozzárendelni. Megadható továbbá, hogy mennyi ideig jelenjen meg az üzenet

```
Image img;
try{
        img = Image.createImage("/images/wrong.png");
}
catch (IOException e) {img = null;}
Alert a = new Alert("Üzenet","Nincs mentett játék!",img,null);
a.setTimeout(3000);
dpy.setCurrent(a);
```

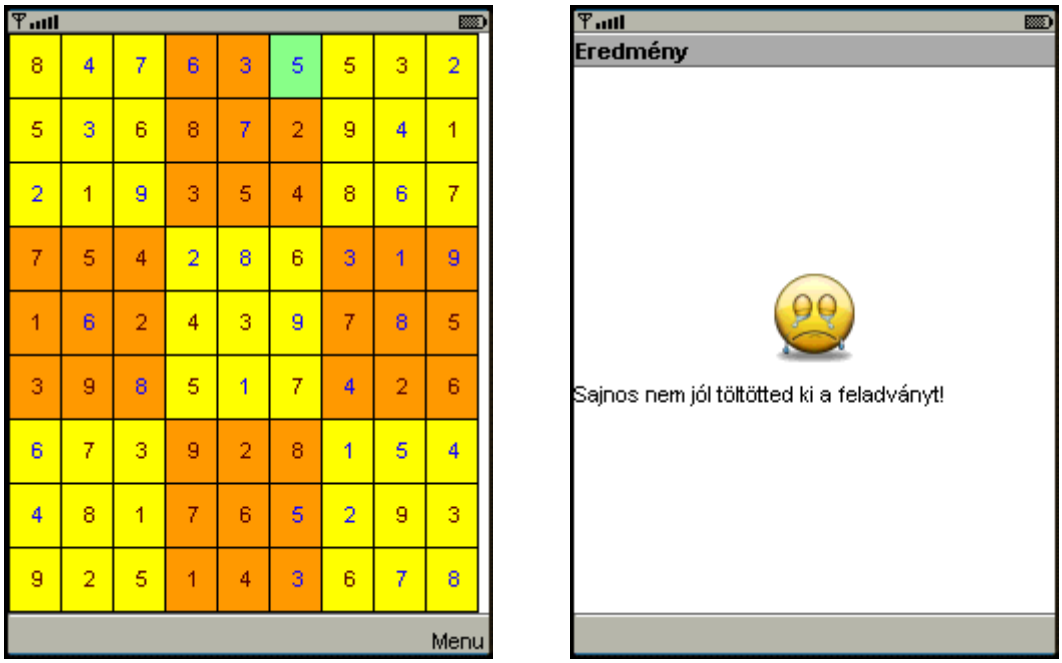
15. ábra. Egy üzenet megjelenítése képpel

Az allFieldSetCheck() metódus ellenőrzi, hogy az összes mezőt kitöltöttük-e, amennyiben nem, üzenetet küld.

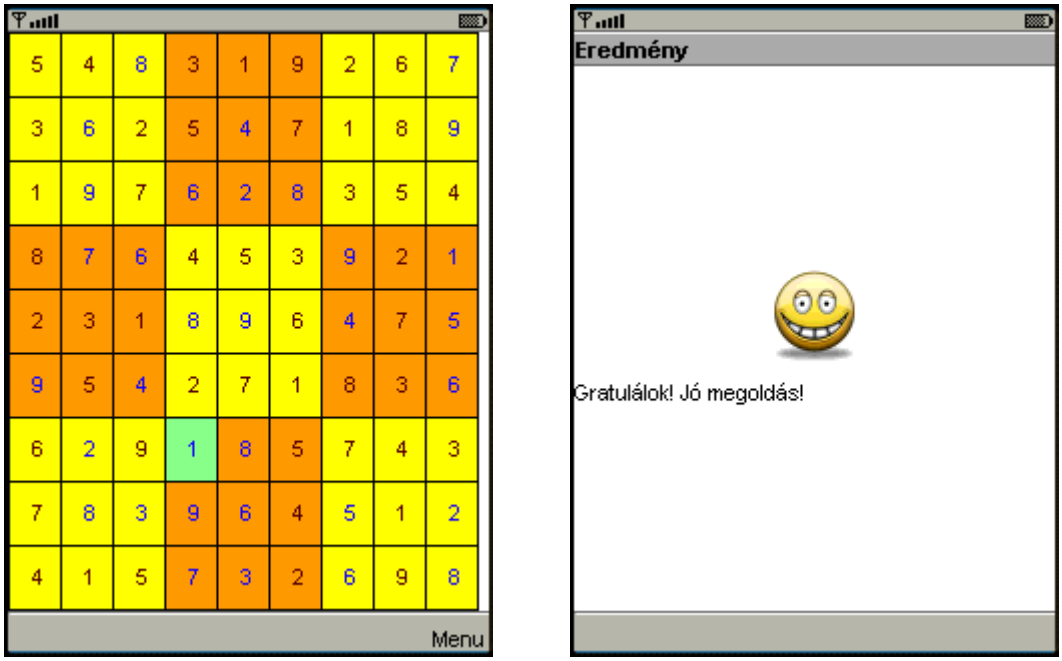
A solutionCheck() metódus visszatérési értéke igaz, ha helyesen töltötte ki a játékos a feladványt, és hamis, ha helytelenül. Természetesen ezekről is tájékoztatja a felhasználót. A feladvány generálásakor használt metódusokat használja. Megkeresi az adott mezővel függésben lévő mezőket (egyező sorban, osztlopban, vagy 3x3-as dobozban vannak.). Amennyiben valamelyikben is egyezést talál, hamis értékkel tér vissza.



16. ábra. Nem teljesen kitöltött feladvány, és az ellenőrzéskor kapott üzenet



17. ábra. Nem jól kitöltött feladvány, és az ellenőrzéskor kapott üzenet



18. ábra. Jól kitöltött feladvány, és az ellenőrzéskor kapott üzenet

A főmenü gombot választva a főmenübe jutunk, ekkor a játék eddigi állása mentésre kerül, és később ebből az állásból folytatható.

6.3.2. Játék folytatása

Játék folytatásakor ellenőrizzük, hogy van-e már mentett játékállásunk. Amennyiben még nincs, egy üzenettel közöljük a felhasználó felé. Egyébként betöltjük a mentett állást. A játék betöltését a restoreGameStore() metódussal végezzük. Előzőleg az állás mentésekor külön eltároltuk az egyes mezőkhöz tartozó értékeket, valamint azt, hogy az adott mező értéke látható-e vagy sem a játékos számára. Létrehoz egy üres sudoku táblát, majd a mentett értékekkel és láthatósági tulajdonságokkal feltölti azt.

A betöltés után a játék az előzőleg abbahagyott állapotából folytatható.

```
private boolean restoreGameStore() {
      if (gameStore != null && gameVisibilityStore != null) {
         try {
            byte[] game = gameStore.getRecord(2);
            byte[] gameVisibility = gameVisibilityStore.getRecord(3);
            booolean vis;
            if (game.length == 81 && gameVisibility.length == 81) {
                  stable.table.makeAnEmptyPuzzle();
                  int i,j;
                  for(int k = 0; k < 81; k++){
                           i = k / 9;
                           j = k % 9;
                           stable.table.table[i][j].setValue(game[k]);
                           vis = (gameVisibility[k] == 1)?true:false;
                           stable.table.table[i][j].setIsVisible(vis);
                  }
            return true;
            }
         }
         catch (RecordStoreException ex) {}
      }
      return false;
   }
```

19. ábra. Mentett játékállás betöltése

6.3.3. Beállítások

A beállításoknál megadhatjuk a nehézségi szintet. A nehézségi szint megadását egy Form segítségével oldottam meg. A Form könnyen megjeleníthető a képernyőn, a fokozatok közötti választást pedig a ChoiceGroup osztálly segítségével valósítottam meg.

```
optionsForm = new Form("Beállítások");
optionsForm.addCommand(okCommand);
optionsForm.setCommandListener(this);
difficultyChoice = new ChoiceGroup("Nehézségi szint", Choice.EXCLUSIVE);
difficultyChoice.append("Könnyű", null);
difficultyChoice.append("Közepes", null);
difficultyChoice.append("Nehéz", null);
difficultyChoice.setSelectedIndex(stable.table.getDifficulty(), true);
optionsForm.append(difficultyChoice);
```

20. ábra. A beállítások formjának létrehozása

6.3.4. Információk

Az információk képernyőt a beállításokhoz hasonlóan a Form osztály segítségével valósítottam meg, a játékok számát pedig az előzőleg elmentett információkból nyerem vissza. Itt olvashatja el a felhasználó a játékszabályt is.

6.3.5. Kilépés

A játékból való kilépéskor mentésre kerülnek a beállítások, és az utolsó játékállás, majd véget ér az alkalmazás

7. Következtetés

Az alkalmazás architektúrája nem csak a sudoku játék megvalósítására alkalmas. A játék megoldó algoritmusainak átírásával / cseréjével bármilyen táblás játék elkészítésére lehetőség van. A játéknak rengeteg továbbfejlesztési módja lehetséges.

Egy lehetséges továbbfejlesztési mód kiegészíteni az alkalmazást hálózati kapcsolattal, mellyel lehetőség nyílna egy szerveren eltárolni a játékosok játékainak adatait, így versenyt felállítani közöttük.

Egy másik lehetőség, hogy a játékos testre szabhatja saját játékát, tehát nem csak a nehézségi szinteket adhatja meg. Megadhatja például, hogy lehessen-e több, vagy csak egy megoldása a feladványnak, a kitöltött feladványból kitörlendő mezők számát, a kitörlendő mezők szimmetrikusak legyenek-e vagy sem.

Manapság egyre gyakoribb, hogy nem a megszokott 9x9-es táblán, hanem ennél nagyobb, például 16x16-os, vagy akár ennél is nagyobb táblán játszanak. Ekkor természetesen a számjegyeken kívül újabb karakterekre van szükség. Általában az abc betűit használják erre. Léteznek a 9x9-esnél könnyebb (pl. 5x5), sőt, még nem szimmetrikus táblájú feladványok is.

Egy érdekes fejlesztés lehet az alkalmazás más platformra való átvitele (pl. webes alkalmazás), így a játék szinte bárhonnan játszható lenne kényelmesebb felületen is.

Az elkészített alkalmazás könnyen továbbfejleszthető az előzőekben leírt, és más irányokba is, melyek javíthatják a játékélményt (Színek, karakterek beállítása stb.).

Irodalom

- **CLDC specifikáció (JSR 30)**
  http://jcp.org/en/jsr/detail?id=30

- **CDC specifikáció (JSR 36)**
  http://jcp.org/en/jsr/detail?id=36

- **CDC Whitepaper**
  http://java.sun.com/products/cdc/wp/cdc-whitepaper.pdf

- **MIDP specifikáció (JSR 37)**
  http://jcp.org/aboutJava/communityprocess/final/jsr037/index.html

- **MIDP leírások (JSR 37, JSR 118)**
  http://developers.sun.com/mobility/midp/

- **Makay Géza**
  http://www.math.u-szeged.hu/Sudoku/sudoku.pdf

- **Wikipedia – sudoku**
  http://en.wikipedia.org/wiki/Sudoku

- **CDC – Personal Profile**
  http://java.sun.com/javame/reference/apis/jsr216/

- **CDC – Personal Basis Profile**
  http://java.sun.com/javame/reference/apis/jsr217/

- **RMI**
  http://java.sun.com/javame/reference/apis/jsr066/

Függelék

A MANIFESTN.MF és a JAD fájlok

Sudoku.jad:

MIDlet-1: Sudoku, Sudoku.png, Sudoku
MIDlet-Jar-Size: 20876
MIDlet-Jar-URL: Sudoku.jar
MIDlet-Name: Sudoku
MIDlet-Vendor: Veszeli Tamás
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0

MANIFEST.MF:

MIDlet-1: Sudoku, Sudoku.png, Sudoku
MIDlet-Name: Sudoku
MIDlet-Vendor: Veszeli Tamás
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0

Köszönetnyilvánítás

Köszönetet mondok Dr. Fazekas Gábor témavezetőmnek, hogy lehetőséget biztosított munkám sikeres elvégzéséhez és dolgozatom megírásához. Köszönöm segítőkész támogatását és dolgozatom alapos és kritikus átnézését.