SZAKDOLGOZAT

Pálfi Norbert

Debrecen

2011

Debreceni Egyetem
Informatikai Kar

Alkalmazásfejlesztés mobil eszközökre

Sudoku játék megvalósítása

Témavezető:
Dr. Kósa Márk Szabolcs
egyetemi tanársegéd

Készítette:
Pálfi Norbert
Programtervező informatikus (BSc)

Debrecen
2011

Tartalomjegyzék

1. Bevezetés **5**
1.1. Bevezető a mobilos Java-ba 5
1.1.1. Konfigurációk (CDC, CLDC) 5
1.1.2. Mobile Information Device Profile (MIDP) 6
1.2. Sudoku 6
1.2.1. Játékszabály 7
1.2.2. Egy kis számmisztika 7

2. Vízió **8**
2.1. Miért ezt a játékot választottam ? („mit ?”) 8
2.2. A mobilos Java és a sudoku találkozása („milyen környezetben ?”) 8

3. Követelmények feltárása **10**
3.1. Funkcionális elvárások a programtól 10
3.2. Felhasználó felülettel kapcsolatos követelmények 11
3.2.1. A játék fő vizuális elemei : az ablakok 11
3.3. A vezérlés és a hozzá tartozó vizuális változás 13
3.4. Az egyes menüpontok vizuális hatásai 14
3.5. Rendszerkövetelmények 16
3.6. Fejlesztői környezet 16

4. Tervezés **17**
4.1. A Midlet osztály 17
4.1.1. A Midlet osztály főbb adattagjai 17
4.1.2. A Midlet osztály főbb metódusai 18
4.2. A SudokuField osztály 18
4.2.1. A SudokuField osztály főbb adattagjai 19
4.2.2. A SudokuField osztály főbb metódusai 19
4.3. A Sudoku osztály 20
4.3.1. A Sudoku osztály főbb adattagjai 21
4.3.2. A Sudoku osztály főbb metódusai 21

5. Implementálás **23**
5.1. A generálás 23
5.1.1. Adott játékállás megoldhatóságát eldöntő algoritmus 27

5.2. A SudokuField run() metódusa 28

6. Tesztelés **32**

6.1. Tesztterv 32

7. Evolúció **36**

8. Telepítési útmutató **38**

9. Összegzés **39**

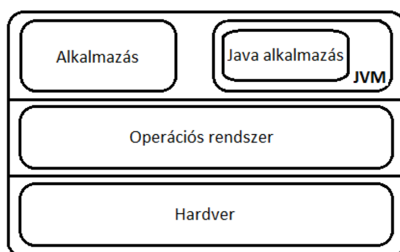
10. Irodalomjegyzék **41**

1. Bevezetés

Szakdolgozati témaválasztásom során két témakör keltette fel érdeklődésemet. Egy olyan játékot szerettem volna létrehozni, ami hasznos, élvezhető, és a logikai készségeinket fejleszti. A másik oldalról tekintve olyan platformon működjön, ami elsősorban mobil, bármikor kéznél van, akár a buszon ülve is szívesen használjuk. Így esett a választásom egy mobil telefonra szánt sudoku játék elkészítéséhez.

1.1. Bevezető a mobilos Java-ba

A Java Micro Edition (Java ME) platformot PDA és mobiltelefon eszközökre való fejlesztéshez találták ki. Manapság már az alsókategóriás mobilok is képesek Java nyelven írt programok futtatására. A Java alapú alkalmazások alapelve az, hogy az operációs rendszeren fut egy Java virtuális gép (Java Virtual Machine, JVM), amin keresztül a programok elérik az operációs rendszer szolgáltatásait, így gyakorlatilag az alkalmazások platformfüggetlenek.



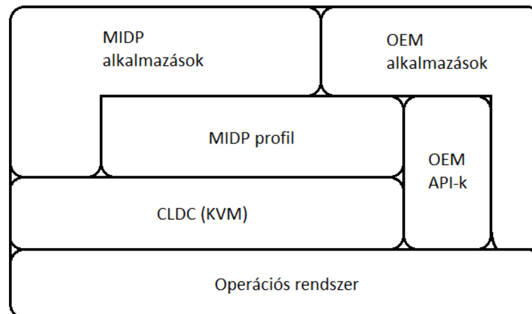
1. ábra. A JVM és az operációs rendszer kapcsolata

1.1.1. Konfigurációk (CDC, CLDC)

A fő különbség a mobil készülékek és a számítógépek közt a teljesítménybeli eltérés. A JVM túl nagy erőforrást igényel ahhoz, hogy egy mobiltelefonon futtatható legyen, éppen ezért definiáltak a Javában egy konfigurációt az azonos tulajdonságú eszközöknek. A konfigurációk a virtuális gépből és az osztálykönyvtárak egy minimális részéből állnak. A két legjelentősebb konfiguráció a Connected Limited Device Configuration (CLDC) és a Connected Device Configuration (CDC). A CDC konfigurációt azokra az eszközökre tervezték, amiknek gyorsabb a processzoruk, több memóriával rendelkeznek és nagyobb hálózati sebességre képesek(videotelefon, GPS). A CLDC-nek a Sun megalkotta a virtuális gép referencia implementációját, a K Virtual Machine-t (KVM). Tehát a CLDC a KVM és az alapvető API-k összessége. Az általam megírt program CLDC-t használ.

1.1.2. Mobile Information Device Profile (MIDP)

Az egy speciális kategóriába tartozó készülékekhez (egy családba tartozó készülékek) tartozó kiegészítő API-kat és jellemzőket egy J2ME MID profil definiál. Egy készülék több profilt is támogathat. Egy J2ME alkalmazás futási környezetét a MIDP és a CLDC együtt alkotja.



2. ábra. J2ME alkalmazások

Ahogy az ábrán is látszik, nem csak MIDP alkalmazásokat készíthetünk, hanem úgynevezett OEM alkalmazásokat is. A fő eltérés az, hogy az OEM alkalmazások eszközspecifikus API-kat is használnak, amiket a gyártók az adott eszközhöz adtak ki, így az alkalmazás már nem platformfüggetlen. Amennyiben van rá lehetőség, érdemesebb MIDP alkalmazásokat írni, amiket több típuson is lehet futtatni.

1.2. Sudoku

A hagyományos sudoku pálya egy nagy négyzetből, ami további 3x3-as kisebb négyzetből (későbbiekben egy ilyen négyzetre blokként fogok hivatkozni), és minden kisebb négyzet további 3x3-as még kisebb négyzetből (mezőből) áll. Tehát az egész pálya 9x9, azaz 81 mezőből áll, ahol minden mező vagy üres, vagy egy számot tartalmaz 1-től 9-ig.

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

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

3. ábra. Egy üres sudoku pálya, egy egyértelműen kirakható játék, az előző játék megoldása

1.2.1. Játékszabály

Alapszabály : Egy 9x9-es sudoku pálya minden sora és minden oszlopa egy számot pontosan csak egyszer tartalmazhat, továbbá egyik blokkban sem lehet számismétlődés.

– egy sudoku játék megoldása : Akkor mondjuk egy A játékállásra, hogy az megoldása egy másik B állásnak, ha B minden nem üres mezőjében lévő szám megegyezik az A állás ugyanazon a pozíciójában lévő számmal, és A nem tartalmaz üres mezőt, továbbá teljesül A-ra az alapszabály.

– egyértelműen kirakható játék : Egy játékállás akkor egyértelműen kirakható, ha érvényes rá az alapszabály, van legalább egy üres mezője, és egyetlenegy megoldása létezik.

Játékmenet : Adott egy egyértelműen megoldható játék, ez az állapot a kezdőállapot. A célállapot nem más, mint a kezdőállapothoz tartozó megoldás. A játékosnak a kezdőállapotból kell eljutnia a célállapotig úgy, hogy kétféle műveletet (operátort) használhat :

– beszúrás : Adott játékállás bármely üres mezőjébe írható 1 és 9 közé eső szám az alapszabály megsértése nélkül.

– törlés : Egy játékállás azon mezőit szabad törölni, amik a kezdőállapotba üresek voltak.

1.2.2. Egy kis számmisztika

Bertram Felgenhauer és Frazer Jarvis sheffieldi matematikusok foglalkoztak azzal a kérdéssel, hogy hány különböző sudoku játék állítható elő. Számítógépes program segítségével sikerült kiszámítaniuk, hogy 6670903752021072936960 db helyes sudoku kitöltés létezik.

Azt még nem sikerült bizonyítani, hogy legalább hány mező kitöltése szükséges ahhoz, hogy egyértelműen megoldható játékot kapjunk, habár olyan rejtvényt még senki nem tudott előállítani, amiben kevesebb, mint 17 mező van kitöltve. Gordon Royle ausztrál matematikus már (2006-os adatok szerint) 35396 olyan lényegesen különböző sudoku rejtvényt halmozott fel, amikben 17 mező van kitöltve. A és B rejtvényt lényegesen különbözőnek mondjuk, ha A nem áll elő B alábbi transzformációiból :

– A kilenc számjegy permutációja.

– A mátrix transzponálása (sor-oszlop csere).

– A sorok vagy oszlopok permutálása egy 3x3-as blokkon belül.

– A 3x3-as sor-blokkok vagy oszlop-blokkok permutálása.

2. Vízió

Minden szoftverfejlesztési folyamatot megelőzi a vízió, azaz egy alapvető elképzelés arra nézve, hogy az elkészítendő alkalmazás mit és milyen környezetben valósítson meg. Jelen esetben a „mit”-re a válasz: egy játszható egyszemélyes sudoku játék, melyhez tartozó funkcionális és megjelenítési elképzeléseimet a következő bekezdésben fogom részletezni. A „milyen környezetben”-re a válaszom: nagyobb felbontású kijelzővel rendelkező Java alkalmazást támogató mobil készülékek. Az alkalmazáshoz szükséges minimális hardverkövetelményeket szintén a későbbiekben tárgyalom.

2.1. Miért ezt a játékot választottam? („mit?”)

Egy pár éve, mikor először a kezembe vettem egy sudoku rejtvényt tartalmazó magazint, szinte azonnal megtetszett a játék. A keresztrejtvényekkel szemben nem szükséges a játékhoz semmilyen háttértudás, elegendő mindössze az előzőekben leírt játékszabályokat ismerni. Sokszor töltöttem ki sudoku rejtvényeket, mikor mással nem igazán tudtam eltölteni az időmet, leginkább a vonaton, utazás közben. Minden rejtvény kitöltése közben algoritmusokat találtam ki, majd alkalmaztam is azokat, annak érdekében, hogy minél gyorsabban oldjak meg egy rejtvényt. Később egyre jobban foglalkoztatott a játék mögött lévő matematika, az, hogy milyen megoldó algoritmusok léteznek, és az, hogy hogyan lehet előállítani egy játékot. Az előbb említett két feladattal a mesterséges intelligencia foglalkozik. Miután az egyetemi tanulmányaim során elegendő tudást szereztem ahhoz, hogy egy sudoku játékot megvalósító programot írjak, elég nagy kihívásnak tartottam, hogy ez adja a szakdolgozati témámat.

2.2. A mobilos Java és a sudoku találkozása („milyen környezetben?”)

Maga az alkalmazás elképzelésekor egy minden téren mobil játékot akartam létrehozni. Hordozható legyen egyrészt olyan szempontból, hogy maga az az eszköz legyen mobil, amire fejlesztek, másrészt platformfüggetlen legyen. Mivel a legtöbb mobiltelefon, illetve PDA képes Java ME alkalmazások futtatására, ezért ezen programozási nyelv mind a két szempontnak eleget tesz. Minden egyes fejlesztési lépésnél, leginkább az implementáció során nagy figyelmet kell szentelni annak, hogy azok az eszközök, amikre az alkalmazás készül, közel se rendelkeznek akkora erőforrással, mint egy számítógép. Annak ellenére, hogy a legújabb okos telefonok és PDA-k megközelítik egy PC teljesítményét, és a kijelzőjük felbontása pedig egy monitorét, a Java ME futtató környezet csak a profilban definiált erőforrást használja. Jelen esetben olyan algoritmust kell választani mind a pálya generálásához, mind a megoldás kereséséhez, hogy az a lehető legkevesebb erőforrást használja. Nem túl szerencsés dolog egy brute-force algoritmus

használata, főleg nem ezen a platformon. A víziómban szereplő játszható játék kifejezése magában foglalja azt, hogy használható legyen, ami egy bizonyos sebességet feltételez az egyes interakciók és a rájuk adott válaszképernyők között. Egy sudoku játék során a legnagyobb erőforrást a generáló és a megoldó algoritmusok igényelnek, az összes többi művelet ezek töredékét használja. Összefoglalásként elmondhatjuk, hogy a mobilprogramozás összes hátránya ellenére a mobilunk az, ami mindig nálunk van, és sokkal hamarabb vesszük kezünkbe szórakoztatási jelleggel, mint egy laptopot vagy egy magazint, amihez még toll is kell. Saját tapasztalatból merítve, egy közlekedési járművön nehézkes a toll használata, ugyanakkor egy program esetén gyakori opció a törlés vagy a visszavonás. Következésképp a legmegfelelőbb platform egy ilyen jellegű játék számára a Java ME.

3. Követelmények feltárása

Ahogy a vízióbeli elképzelések elmélyülnek a fejlesztőkben és/vagy a megrendelőkben, úgy alakulnak ki a követelmények a szoftverrel szemben. Ez az a fázis a rendszer fejlesztésében, amikor konkrétan megadható az, hogy mit csináljon az elkészítendő alkalmazás, vagy az, hogy éppenséggel mit ne csináljon. Jelen probléma esetében két részre osztom a követelményeket. Egyik csoportba azokat a követelményeket sorolom, amik egy funkciót fednek le, ezek lesznek a funkcionális követelmények. A másik csoportot azon elvárások alkotják, melyek a megjelenéssel vannak kapcsolatban. A felhasználói felület az, amit a végfelhasználó lát az alkalmazásból, ezért a megrendelő ezeket az elvárásait részletesen szokta megadni. A fejlesztők akkor végeztek jó munkát, ha az elkészült szoftver minden követelménynek eleget tesz.

3.1. Funkcionális elvárások a programtól

Ezek a követelmények a vízióban szereplő „mit valósítson meg?" kérdésre adott választ részletezik, konkretizálják, megadják azt hogy ennek a sudoku játéknak pontosan mit kell tudnia.

1. Legyen képes az alkalmazás létrehozni egy véletlenszerű, egyértelműen megoldható feladványt.

2. Az előállított tábla üres mezőit a felhasználó képes legyen módosítani, azaz üres mezőbe számot beszúrni 1 és 9 között, a felhasználó által korábban beszúrt értéket törölni, illetve módosítani.

3. Az előállított tábla azon mezőit nem módosíthatja a felhasználó, ahol a mező nem üres a generálást követő állapotában.

4. A feladvány előállítása előtt a felhasználónak ki kell tudnia választani azt, hogy milyen nehézségi szintű játékot szeretne. Legyen alkalmas a program 3 féle nehézségi szint alapján generálni játékot. A nehézségi szintet valamilyen heurisztika segítségével határozza meg.

5. Az előállítás után, tehát a játékmenet közben legyen lehetőség az alábbi opciók kiválasztására:

- új játék: Amennyiben a felhasználó új feladványt szeretne előállítani, akkor ezzel az opcióval képes legyen megadni egy nehézségi szintet, és a program ennek a nehézségnek megfelelő új játékot hoz létre.

- megoldás: Bármely játékállásban ez az opció jelenítse meg a felhasználónak a feladvány megoldását.
- segítség: Ennek az opciónak kiválasztásának hatására egy, még üres mezőbe beszúrja a program az onnan hiányzó értéket.
- validálás: Egy adott játékállás során ez az opció arra szolgál, hogy a felhasználó által bevitt értékek helyességét megvizsgálja, és azt jelezze a játékos felé. A helyesnek ítélt értéket kezelje a program úgy, mintha ez is generált lenne.
- cheat mód be/ki: Ennek az opciónak a neve változik a program futása során attól függően, hogy a cheat mód aktív („cheat mód ki"), vagy sem („cheat mód be"). Alapértelmezetten ez a mód kikapcsolt állapotban van. A különbség a két mód közt: Amennyiben kikapcsolt cheat módban használom a játékot, úgy egy felhasználó által módosítható mezőbe bármilyen számot be tudok szúrni, illetve, ha ott már volt érték, akkor bármilyen számra tudok módosítani (természetesen a számokat 1 és 9 között kell érteni). A cheat mód aktív állapotában ezen helyekre csak azokat a számokat engedi beszúrni az alkalmazás, amelyek még az adott sorban, oszlopban és blokkban még nem szerepelnek. A törlés funkció mind a két esetben ugyanúgy működik.

6.. Amennyiben a játék során egy üres mező sem marad, akkor a játéknak értesítenie kell a játékost, hogy a kitöltés sikeres volt-e vagy sem.

7.. A játékos minden esetben legyen képes az alkalmazás szabályos bezárására.

3.2. Felhasználó felülettel kapcsolatos követelmények

Az előző rész arról szólt, hogy mit kell a játéknak megvalósítania, mik is a funkciói. Ezzel szemben ez a rész azt fogja taglalni, hogy mindezen funkciók, milyen kinézet mellett valósuljanak meg, tehát a felhasználói felületet specifikálja.

3.2.1. A játék fő vizuális elemei: az ablakok

A játék alapjában véve, a funkciók miatt két nagyobb megjelenítési részre osztható. Az első ablak a feladvány előállítása előtti funkciókat fedi le, a második ablak a játékmenetet. A két ablak közti átmenetet a későbbiekben részletezem.

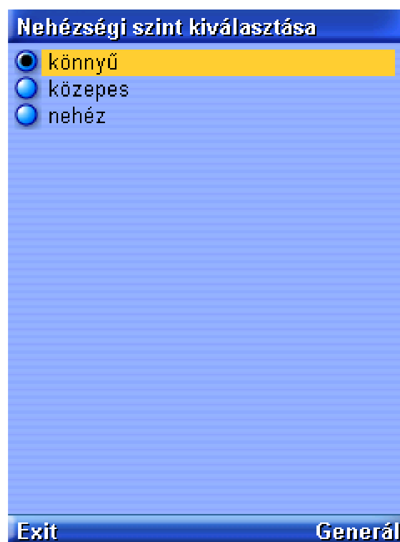
A „feladványt előállító" ablak:.

Ez az ablak a legelső, amit az alkalmazás indításakor látunk. Ahhoz, hogy a felhasználó tudja, hogy most mit is kell tennie, az ablak címsorában az alábbi üzenet jelenjen meg:

„Nehézségi szint kiválasztása". A címsor alatt jelenjen meg a 3 nehézségi szintből álló lista, egymás alatt. Minden szint előtt legyen egy „radio button", amivel ki tudja választani a játékos a kívánt nehézségi szintet. Alapértelmezetten a könnyű legyen aktív, ha egy másik elemet választunk ki, akkor az összes többinek inaktívnak kell lennie. Ez az ablak tartalmazzon továbbá két gombot az ablak alján:

– kilépés (bal alsó gomb): Az alkalmazás szabályos leállítása következik be megnyomásakor.

– generál (jobb alsó gomb): Megnyomásakor a kiválasztott nehézségi szintnek megfelelő feladvány állítja elő és továbbvisz minket a program a következő ablakra.

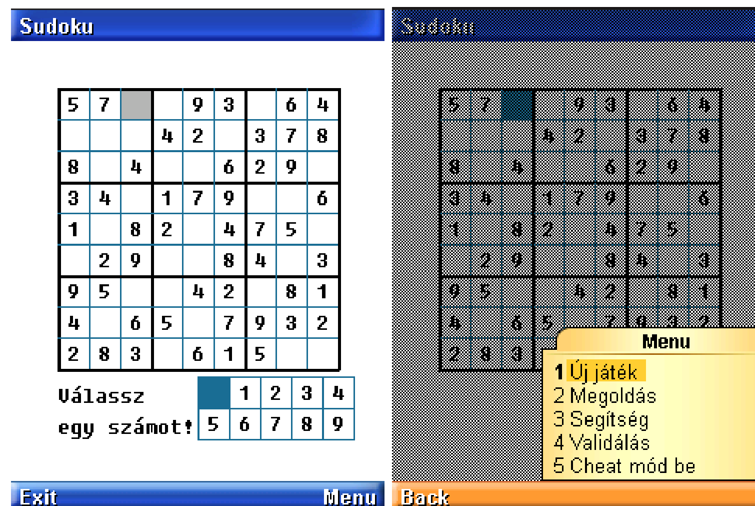


4. ábra. A játék indításakor megjelenülő ablak

A „játékmenet" ablak:.

Ez az ablak látszódik az egész játékmenet alatt, éppen ezért eléggé dinamikus. Ebben a részben csak a főbb elemeit részletezem, az egyes komponensek vizuális változását a következő részben fejtem ki. A címsor a „Sudoku" szöveget tartalmazza. Az ablak felső nagyjából 5/6-od részén középre igazítva egy 9x9-es táblázat jelenjen meg, ami az aktuális állapotát jeleníti meg a játéknak. A táblázat azon sorai és oszlopai előtt és után vékony kék vonal legyen, ami nem blokk határ, illetve nem az egész táblázat széle, összes többi esetben vastag fekete vonal legyen. A generált számok minden esetben feketével jelenjenek meg a táblázat megfelelő pozíciójában. A táblázat többi mezőjével a következő részben foglalkozok. Az alsó részben a vezérlésnek megfelelően vagy egy 2x5-ös táblázat legyen, mellette azzal a felirattal, hogy „válassz egy számot", vagy üres legyen,

vagy esetleg üzenetet is tartalmazhat. A 2x5-ös táblázat minden sora és oszlopa előtt vékony kék vonal legyen. Az első sor első oszlopa üres legyen (törlést szimbolizál), a többi 1-től 9-ig tartalmazza a számokat, melyek fekete színűek legyenek. A bal alsó sarokban itt is legyen egy kilépés gomb, viszont jobb oldalt legyen egy lista, ami a következőket tartalmazza: új játék, megoldás, segítség, validálás, cheat mód ki/be. Ezen elemek funkcionális leírásáról már esett szó, viszont vizuális hatásukról szintén a következő részben foglalkozom.



5. ábra. Játékmenet közben látható ablak, az előző ablak menüje

3.3. A vezérlés és a hozzá tartozó vizuális változás

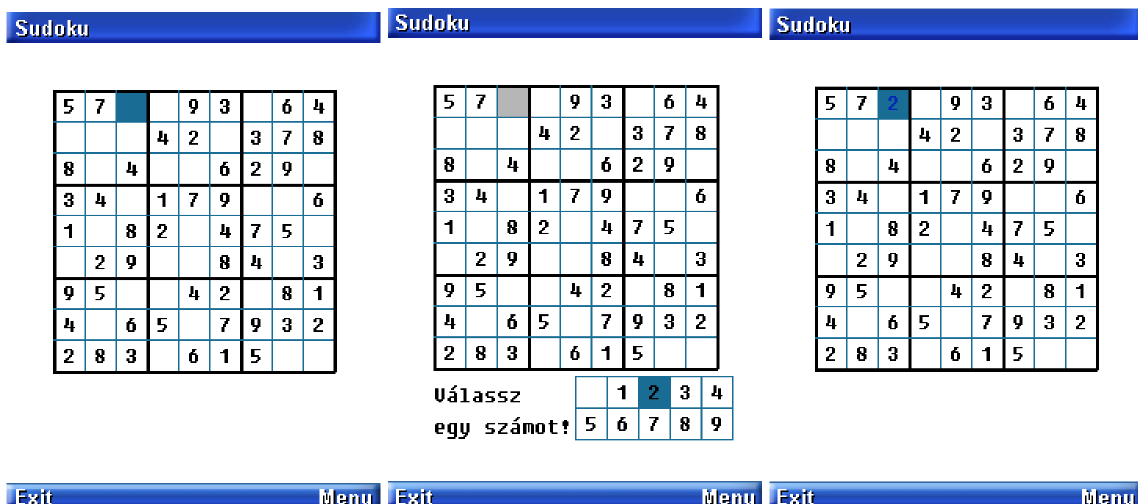
Játékmenet közben használható gombok: Ok gomb, irányító gombok, továbbá a jobb és bal oldali menü gombok. A táblázatban alapértelmezetten az első üres mezője sötétkék legyen ezzel jelezve, hogy az adott mezőn állok. Továbbá alapértelmezetten nem látszik a 2x5-ös mátrix. Az irányító gombok hatására a megfelelő irány felé lévő legközelebbi olyan mező legyen kijelölve (sötétkék), ahol ugyanabban a pozícióban lévő generált feladvány mezője üres. A lépkedés során soha ne léphessünk ki a táblázatból. Ha jobbra akarok lépni, de nincs adott sorban megfelelő mező, ahova léphetek, akkor a következő sorban keresek, ha ott sincs, akkor a következőben, és így tovább, ha egyáltalán nincs elem egyik következő sorban se, akkor marad az épp kijelölt elem kijelölve. A többi irányba is ugyanez a helyzet azzal a kivétellel, hogy balra gombra az előző sorokban keres, lefele gombra következő oszlopokban, felfele gombra pedig az előző oszlopokban.

Az Ok gomb kétféle funkcióval működik:

- amennyiben a vezérlés a 9x9-es táblázaton van: Ez az alapértelmezett. Ebben az esetben az épp kijelölt mező sötétkék színről szürkére változik, ezzel jelezve, hogy a 2x5-ös táblázatre esett a vezérlés. A 2x5-ös mátrix megjelenik az ablak alsó részén, majd az első sor első eleme sötétkék lesz, azaz aktív. A vezérlő gombok itt is ugyanúgy működnek, mint a 9x9-es mátrix esetén.

- ha a vezérlés a 2x5-ös táblázaton van: Akkor az épp sötétkéken kijelölt mező értéke másolódik a 9x9-es táblázat szürke mezőjébe, ahol a szám kék színnel fog megjelenni. Amennyiben a legelső, azaz az üres mező aktív, akkor a szürke mező értéke törlődik. A vezérlés visszakerül a 9x9-es táblázatba: a szürke mező ismét aktívvá válik, azaz sötétkékké.

Azért kék színnel kell beszúrni a számot, hogy egyértelműen különüljön el a generált mezők értéke a felhasználó által beírtétól.



6. ábra. Az alábbi ábrasorozat bemutatja az Ok gomb működését

3.4. Az egyes menüpontok vizuális hatásai

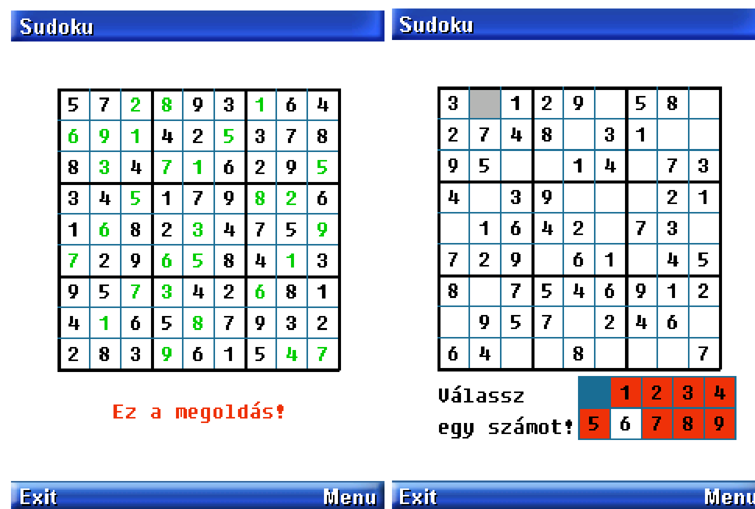
- új játék: Az első ablak válik láthatóvá, olyan mintha újraindítottuk volna az alkalmazást.

- megoldás: A 9x9-es táblázatban az összes helyről, ahol nem fekete szám van, kicseréli az ott lévő elemet a megoldásban szereplő, ugyanabban a pozícióban lévő számra, és ezt zöld színnel jeleníti meg. Az alsó részen piros színnel kapunk egy üzenetet: „Ez a megoldás!".

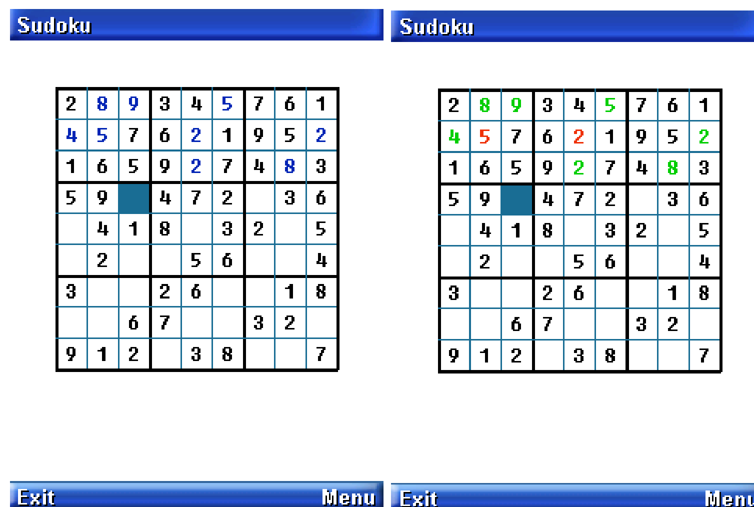
– segítség: Ennek a menüpontnak a hatása az, hogy a 9x9-es táblázat egy üres mezőjébe kék színnel beszúr a megoldásban szereplő, ugyanebben a pozícióban lévő számot.

– validálás: A 9x9-es táblázatban lévő összes kék számra megnézi, hogy a megoldásban ugyanebben a pozícióban szereplő számmal megegyezik-e, ha igen zöld színnel írja be ugyanezt a számot, ellenkező esetben pirossal.

– cheat mód ki/be: Ha ki van kapcsolva ez a mód, akkor normál működés van, ellenkező esetben, a 2x5-ös mátrixban minden olyan mező piros színű, ahol a szám a 9x9-es mátrixban kijelölt elem sorában, oszlopában vagy blokkjában szerepel. A piros mezőket lépkedés során hasonló mód átlépjük, mint a 9x9-es táblázatban a fekete számokat tartalmazó mezőket.



7. ábra. Megoldás opció választása esetén kapott ablak, példa a cheat módra



8. ábra. Validálás előtti és utáni képernyő

3.5. Rendszerkövetelmények

Ezen alkalmazás futtatására egy olyan mobil készülék szükséges, mely támogatja a CLDC 1.0 konfigurációt, illetve a MIDP 2.0 profilt. A kijelző felbontása nem korlátozott, mivel a tervezettnél kisebb felbontással rendelkező készülékeken is elindul az alkalmazás, csak nem az egész kijelző fog látszani. Az ajánlott felbontás : 320x240, erre optimalizáltam a játékot.

3.6. Fejlesztői környezet

Az alkalmazás fejlesztését windows platformon végzem, amihez az alábbi eszközöket használom :

- Java ME Software Development Kit (SDK) : Ezen SDK tartalmazza a mobilspecifikus fejlesztői eszközöket : osztályokat, interfészeket és az ezekhez tartozó API-kat. Önmagában egy SDK nem elegendő egy mobil alkalmazás fejlesztéséhez, mivel az nem tartalmaz emulátort a futtatáshoz.
- Java ME emulátor : Futási időben ezen emulátor virtualizál egy mobil készüléket.
- IDE : A fejlesztéshez a Netbeans 6.9.1-es verzióját használom, ahol a mobility plugineket telepítettem, és a projekt konfigurációjánál mind az SDK, mind az emulátor megfelelőképpen adtam meg.

4. Tervezés

Ebben a részben az alkalmazás megtervezését írom le, azt hogy hogyan szeretném kialakítani az egyes osztályokat, mely tulajdonságokat és viselkedéseket mely osztályokba kategorizálom. Fontos kérdés továbbá, amit ugyancsak itt fogok tárgyalni, hogy az egyes osztályok milyen metódusokat fognak tartalmazni, és azok mit is valósítanak meg.

4.1. A Midlet osztály

Ennek az osztálynak az ősosztálya a következő osztály: javax.microedition.midlet.MIDlet. Ennek köszönhetően, mikor a játékot futtatjuk a Midlet osztály fog belépési pontként szolgálni. A Midlet implementálja még az alábbi interfészt: javax.microedition.lcdui.CommandListener, mely interfész metódusait implementálva elérjük, hogy a felhasználó egyes interakcióira megfelelően reagáljon a program. Ez az osztály fogja egyértelműen végezni az ablakok közti váltást és azok inicializálását.

4.1.1. A Midlet osztály főbb adattagjai

Ahhoz, hogy az ablakok közti navigációt megvalósítsuk, szükségünk lesz egy Display objektumra. A Display setCurrent(GameCanvas canvas) metódusával el tudom érni a képernyőváltást. Ezen metódus hívása esetén a canvas objektum által kialakított ablakot fogjuk látni. Jelen esetben, mivel két ablakkal dolgozunk, és a belépő ablak kialakításával és vezérlésével a Midlet osztály foglalkozik, ezért csak a „játékmezőt" tartalmazó osztályt kell létrehozni, mely gyermeke a GameCanvas osztálynak és ebből kell egy példányt létrehozni a Midlet osztályba. A játékmezőt tartalmazó osztály: SudokuField

Létre kell hozni egy List objektumot, melyhez az egyes parancsokat fogom hozzáadni, annak függvényében, hogy a vezérlés épp melyik ablakban van.

Továbbá az összes ablak által használatos összes parancsot itt definiálom. Azaz létrehozom a következő Command objektumokat:

- exitCommand: A program szabályos bezárását éri el.
- generate: Előállít egy feladványt és a vezérlés a "játékmezőre" kerül.
- newGame: A vezérlés az első ablakra kerül.
- solution: A „játékmezőn" a feladvány megoldása jelenik meg.
- giveHint: A „játékmezőn" az egyik üres helyre a egy elem beszúrása történik.

– validate : A „játékmezőre" bevitt számok helyességének ellenőrzése.

– cheatOn : Cheat mód kikapcsolása.

– chaetOff : Cheat mód bekapcsolása.

4.1.2. A Midlet osztály főbb metódusai

– initMidlet() : Az üdvözlő képernyő inicializálására szolgál. A List objektumhoz az alábbi Command utasításokat adom : exitCommmand, generate. Továbbá a List objektum append(String value, Image img) metódusával felviszem a nehézségi szinteket. Majd beállítom a setCommmandListener(this) metódussal azt, hogy a utasítások feldolgozását ebben az osztályban a commandAction(Command c, Displayable d) metódus végzi. Legvégül, miután minden inicializálás megtörtént a Display objektum setCurrent(this) metódusának hívásával elérem, hogy megjelenjen az ablak.

– commandAction(Command c, Displayable d) : A Command objektumok hatását implementáló metódus. A metódus megvizsgálja, hogy mi is az aktuális utasítás, és az utasításnak megfelelő műveleteket végzi el. A SudokuField osztály is ennek az osztálynak ezen metódusát használja a CommandListener-ként. Abban az esetben, ha a Commmand a SudokuFileld osztályra vonatkozik (newGame, validate, solution, giveHint, cheatOn, cheatOff), akkor a SudokuField példányának a megfelelő metódusát hívom meg, amit a SudokuField osztály tárgyalásakor részletezek.
  Ebben a metódusban a legérdekesebb a generate utasítás feldolgozása. Itt példányosítok egy SudokuField objektumot, a megfelelő nehézségi szint függvényében. Ezek után inicializálom a „játékmező" ablakot, azaz a megfelelő Commmandokat hozzáadom a listához. Beállítom a címsort „Sudoku"-ra. És a commandListener-t ugyanennek az osztálynak, ugyanerre a metódusára. Inicializálás után beállítom azt, hogy a megjelenítés kerüljön az előbb létrehozott SudokuField objektumra.

4.2. A SudokuField osztály

Ez az osztály végzi a „játékmenet" ablak vezérlését és megjelenítését, leszámítva a menüt, amit az előbb részleteztem. Mivel a felhasználó által végzett input, azaz egy gomb lenyomása és az ezután bekövetkező változás az ablakon szorosan összefügg, ezért ezt a két funkciót egy osztályon belül valósítottam meg. Az osztály a GameCanvas leszármazottja, illetve implementálja a Runnable interfészt.

4.2.1. A SudokuField osztály főbb adattagjai

- Sudoku sudoku: Egy a későbbiekben részletezett Sudoku osztály példánya. Lényegében a feladvány összes tulajdonságát ez az objektum tárolja, és a generálást is ennek az objektumnak a példányosítása során végzi a rendszer.

- boolean isOk: A vezérlés ebben az ablakon belül is két helyen lehet. Magában a sudoku táblázatban (felső táblázat) vagy a beszúrandó számokat tartalmazó táblázatban (alsó táblázat). Amennyiben ez a változó true, akkor a vezérlés az utóbbi helyen van, ellenkező esetben a sudokut tartalmazó táblázatban.

- int currentKey: A felhasználó által éppen lenyomott billentyű kódját tárolja.

- Graphics graphics: A 2 dimenziós megjelenítéshez szükséges objektum.

- boolean cheatMod: Ez a változó tárolja azt, hogy a játék cheat módban fut-e vagy sem. Alapértelmezetten az értéke false.

- int widthInField: A felső táblázatra vonatkozóan tárolja az aktuális pozíció szélességi értékét pixelben értelmezve.

- int heigthInField: A felső táblázatra vonatkozóan tárolja az aktuális pozíció magassági értékét pixelben értelmezve.

- int widthInNumpad: Az alsó táblázatra vonatkozóan tárolja az aktuális pozíció szélességi értékét pixelben értelmezve.

- int heigthInNumpad: Az alsó táblázatra vonatkozóan tárolja az aktuális pozíció magassági értékét pixelben értelmezve.

4.2.2. A SudokuField osztály főbb metódusai

- SudokuField(int level): A sudoku objektumot példányosítja a megadott nehézségi fok függvényében. Inicializálja az alábbi változókat: widthInField, heigthInField. Legvégül meghívja az ebben az osztályban található drawTable() metódust.

- drawTable(): A felső táblázatot rajzolja meg. A számokat beszúrja a megfelelő színnel, attól függően, hogy az adott szám a generálás pillanatában jött létre (fekete), vagy a felhasználó szúrta be. Amennyiben a felhasználó szúrta be, további 3 opció lehetséges: validált szám és valid (zöld), validált szám és invalid (piros), vagy nem validált (kék). Az alábbi két változó függvényében a kijelölt mezőt színezi be: widthInField, heigthInField.

Ha isOk false, akkor sötétkék színnel, egyébként szürke színnel és meghívja a drawNumpad() metódusnak ugyancsak ezen osztálynak.

- drawNumpad(): Az alsó táblázatot rajzolja meg. A kijelölt mezőt a widthInNumpad, heigthInNumpad változók függvényében sötétkék színnel színezi. Amennyiben a cheat mód inaktív más nem történik, ellenkező esetben azok az értékek, amik a felső táblázatban kijelölt pozícióba nem kerülhetnek, azok piros háttérrel jelennek meg.
- clearNumpad(): Törli az alsó táblázatot.
- clearAll(): Törli az egész ablakot.
- fillNumer(int number): A paraméterként megadott számot kék színnel szúrja be a felső táblázat alábbi két változója által meghatározott pozíciójába: widthInField, heigthInField.
- giveHint(): A sudoku objektum increaseGenerated() metódusát meghívja, majd újrarajzolja az ablakot.
- solution(): A sudoku objektum tárolja a megoldást is. Az egész ablakot törli, majd a felső táblázatot a megoldással tölti ki. Az alsó részre a következő üzenetet írja piros színnel: „Ez a megoldás!".
- validate(): A sudoku objektum getValidMatrix() és getWrongMatrix() metódusával frissíti a felső táblázatot.
- run(): A felhasználó által lenyomott jobbra, balra, fel, le és ok gombok hatására végbemenő vizuális és adatreprezentációs változást végzi. Ennek a metódusnak a futása külön szálon megy végbe, mivel ez egy végtelen ciklus, ami az inputot figyeli. Ennek a metódusnak a részletezése inkább implementációs kérdés, mint tervezési. Részletezni később fogom.

4.3. A Sudoku osztály

Az előző két osztály a vezérlést és a megjelenítést szolgálja. Effektíven az adatokat (szám értékeket) ez az osztály biztosítja. Ennek az osztálynak olyan a felépítése, hogy kívülről le lehessen kérni, illetve beállítani a mezők értékeit, továbbá még arra is választ kell adnia, hogy az adott helyen lévő elemet a felhasználó írta be vagy előre generált. Továbbá tárolni kell a megoldás mátrixot, illetve azt, hogy a felhasználó által beírt érték helyes vagy sem.

4.3.1. A Sudoku osztály főbb adattagjai

- int[][] generatedMatrix : Ez a mátrix reprezentálja az előállított feladvány kezdeti állapotát. Minden mátrixban a 0 jelöli azt, hogy az adott hely üres.

- int[][] userMatrix : Ez a mátrix reprezentálja a felhasználó által beírt értékeket. Természetesen, csak ott lehet 0-tól különböző szám, ahol a generatedMatrix ugyanazon pozíciójában 0 áll.

- int[][] validMatrix : Azokon a pozíciókban ahol a userMatrix megegyezik a solution mátrixxal, ott ebben a mátrixban is ugyanez az érték szerepel, minden más helyen 0. Tehát a helyes kitöltéseket reprezentálja.

- int[][] wrongMatrix : Azokon a pozíciókban, ahol a userMatrix nem nulla és nem egyezik meg a solution mátrix ugyanazon pozíciójában lévő elemmel, ott a wrongMatrix értéke megegyezik a userMatrix ugyanazon pozíciójában lévő elemmel. Tehát a hibás kitöltéseket reprezentálja.

- int[][] solution : Ez a megoldást reprezentáló mátrix. Nyílván 0-át sehol sem tartalmazhat.

- MySet notAvailableNums : Ez az általam írt MySet osztálynak egy példánya. A MySet egy Integereket tartalmazó lista, de mivel a Java Me API nem tartalmaz listát, ezért írtam sajátot.

- int[] levels = {0, 570, 800, 1200, 5000} : Ez a tömb egy nehézségi sort reprezentál : könnyű fokozat : 570-800, közepes fokozat : 800-1200, nehéz fokozat : 1200-500, ahol a számok egy heurisztikán alapuló értékek.

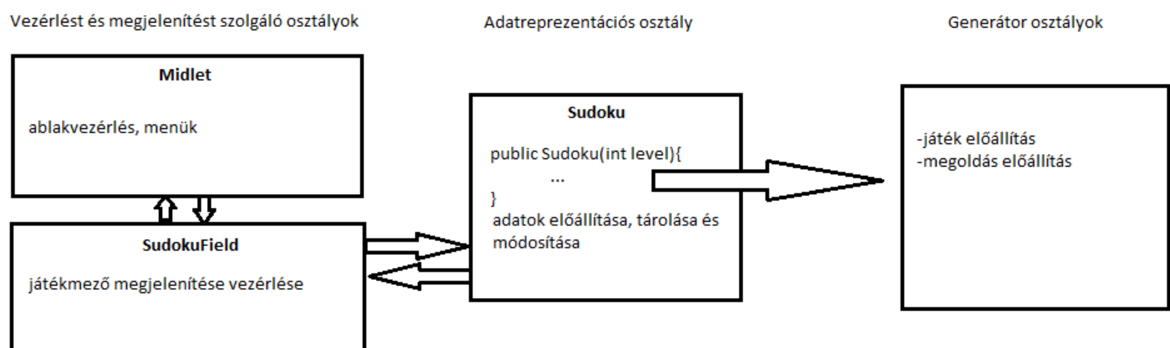
4.3.2. A Sudoku osztály főbb metódusai

- Sudoku(int levelIndex) : Maga a sudoku feladvány generálásához szükséges osztályokból való példánykészítés, azok megfelelő metódusainak hívása, illetve az adattagok inicializálása mindezek alapján. Maga a generálás folyamata implementációs kérdés, későbbiekben részletezem.

- boolean isSolved : True a visszatérési értéke, ha minden mezőben van szám, egyébként false.

- boolean isValidSolved : Ha ki van töltve a feladvány és az helyes is, akkor true a visszatérési érték, egyébként false.

- int[][] getXxxMatrix : Az adott mátrixok más osztályokban történő hivatkozásához szükséges metódusok. Továbbá ezen metódusok hívásakor az adott mátrix értékei frissülnek a userMatrix függvényében.
- void setUserMatrix(int i, int j, int value) : Ezzel a metódussal érhető el a beszúrás vagy törlés adatreprezentációs szinten történő változtatása. A userMatrix i-edik sorának j-edik oszlopában lévő értéket cseré le a value értékre. Amennyiben a value értéke 0, akkor törlésről van szó.
- MySet getNotAvailableNums(int x, int y) : Egy integereket tartalmazó listát ad vissza, ami azokat a számokat tartalmazza, amiket az x-edik sorban és y-odik oszlopban lévő helyre nem lehet beszúrni. Azaz adott sorban, oszlopban vagy blokkban már van ilyen szám. Ezt a metódust a cheat módhoz használom fel.



9. ábra. Az osztályok kapcsolata

Összegzésképp elmondható, hogy a tervezési fázis teljes. Megvannak az osztályok és azok metódusainak a váza. A metódusokról részletes leírással rendelkezünk, kivéve azokat, melyek bonyolultabb implementációs kérdést vetnek fel, amiket a továbbiakban részletesen elemzek.

5. Implementálás

Ebben a részben a számomra legérdekesebb implementációs problémákkal fogok foglalkozni, amikkel a fejlesztés során találkoztam. Konkrét algoritmusokat fogok bemutatni, amiket használtam, illetve összevetem más algoritmusokkal. Két nagyobb egységre osztom ezt a részt: generálás, amit a Sudoku osztály konstruktoránál hajtok végre, illetve vezérlés, a SudokuField osztályon belül.

5.1. A generálás

Mielőtt hozzákezdenénk a generáló algoritmus írásához mindenképpen figyelembe kell venni az alábbi tényezőket:

- Kevés memória és lassú processzor áll rendelkezésünkre egy mobil készülék esetén. Olyan algoritmus kell választani, ami a legkevesebb memória használat mellett elég gyors is.
- Mivel a követelménybeli funkciók igénylik azt, hogy rendelkezésre álljon az eredmény is, ezért olyan algoritmust kell keresni, ami a legrövidebb idő alatt állítja elő a megoldást, vagy már alapból azt állítja elő.

A generálás során mindenképpen rendelkezni kell egy olyan metódussal, ami egy adott játékállásról eldönti, hogy az megoldható, illetve ha megoldható, akkor határozzon meg egy bizonyos metrika alapján egy nehézségi szintet. Az egyszerűség kedvéért, amennyiben nem egyértelműen megoldható a játékállás, akkor az algoritmus 0-át ad vissza, egyébként egy pozitív számot, ami minél nagyobb annál nehezebb a játék. A generálást alapjában véve két irányból lehet megközelíteni. Mind a két eset hasonlít abban, hogy a helyességet ellenőrző algoritmus ugyanaz, de lényegi különbség abban mutatkozik meg, hogy hol és hányszor alkalmazzuk.

Az építő módszer: Ez a módszer, ahogy nevéből is ered lépésenként építi fel a feladványt. Az algoritmus a következő:

- Első körben üres a tábla.
- Második lépésben véletlenszerűen beszúrok egy számot. Majd az adott játékállásra megvizsgálom, hogy megoldható vagy sem.
- Ha megoldható, akkor megnézem, hogy a nehézségi szint elég kicsi-e. Amennyiben elég kicsi a nehézségi szint, akkor készen vagyunk, ha nem, akkor a 2. lépéstől kell folytatni

az algoritmust. Ha meg túl kicsi a nehézségi szint, akkor törölni kell az utoljára beszúrt elemet és a 2. lépéstől folytatni az algoritmust.

– Amennyiben nem megoldható a játék, akkor ugyancsak a 2. lépéstől kell folytatnunk az algoritmust.

A redukciós módszer: Ennél az algoritmusnál egy már helyes megoldásból indulunk ki és azt redukáljuk, míg a megfelelő lesz a megoldás. A módszer algoritmusa:

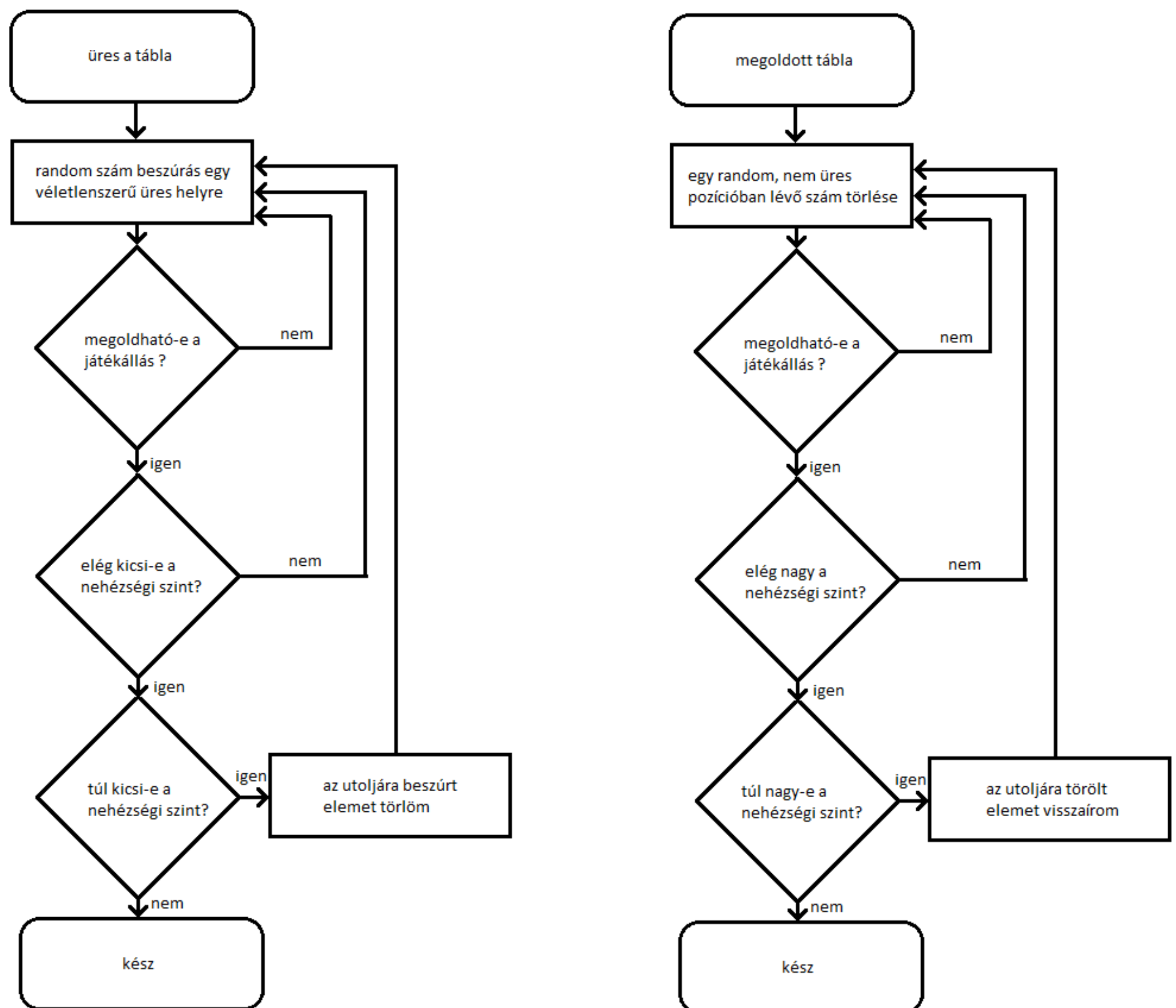
– Első lépésnél a tábla maga egy helyesen kitöltött sudoku feladvány.

– Második lépésnél véletlenszerűen kitörlök egy számot a táblából. Majd a játékállásra megnézem, hogy megoldható vagy nem.

– Ha egyértelműen megoldható, akkor megnézem, hogy a nehézségi szint elég nagy-e, ha igen, akkor készen vagyunk, ha nem akkor a 2. lépésnél kell folytatni. Ha túl nagy a nehézségi szint, akkor vissza kell írni a legutóbb törölt számot és a 2. lépéstől folytatni az algoritmust.

– Ha nem oldható meg egyértelműen a játék, akkor vissza kell írni a legutóbb törölt számot és a 2. lépésnél kell folytatni az algoritmust.

A redukciós módszer hátránya az, hogy szükség van egy plusz algoritmusra, ami legenerál egy megoldott táblát, viszont erre a generálásra nagyon gyors és hatékony algoritmus írható. Az építő módszer viszont sokkal több idő alatt épít fel egy feladványt az üres táblából kiindulva, ráadásul gyakran előfordulhat az, hogy az egyik lépésben még nem megoldható a feladvány, a következőben meg már túl könnyű, ez sok visszalépéshez vezethet. Mivel a legköltségesebb művelet a megoldás keresése és a redukciós módszernél jelentősen kevesebb műveletet igényel ez az algoritmus, így összességében elmondható, hogy a redukciós módszer lényegesen gyorsabb. Mindösszesen a programozónak jelent egy kis extra munkát a kész feladványt előállító metódus megírása. Jelen alkalmazás kereteiben éppen ezért a redukciós módszert fogom alkalmazni.



10. ábra. Az építő módszer és a redukciós módszer folyamatábrája

A kész játékot előállító algoritmus:

- Minden egyes soron végighaladva megpróbálja azt kitölteni.
- Amennyiben sikeres a kitöltés, továbbmegy a következő sorra.
- Ha nem sikerült kitölteni az adott sort, megvizsgálja, hogy hányszor próbálkoztunk a kitöltéssel.
- Amíg kevesebb próbálkozásunk volt, mint 54, addig az adott sort megpróbálja újra kitölteni.
- Az 54. próbálkozásnál az adott sort tartalmazó blokkok első sorától kezdi újra a kitöltést.

Egy sor kitöltése:

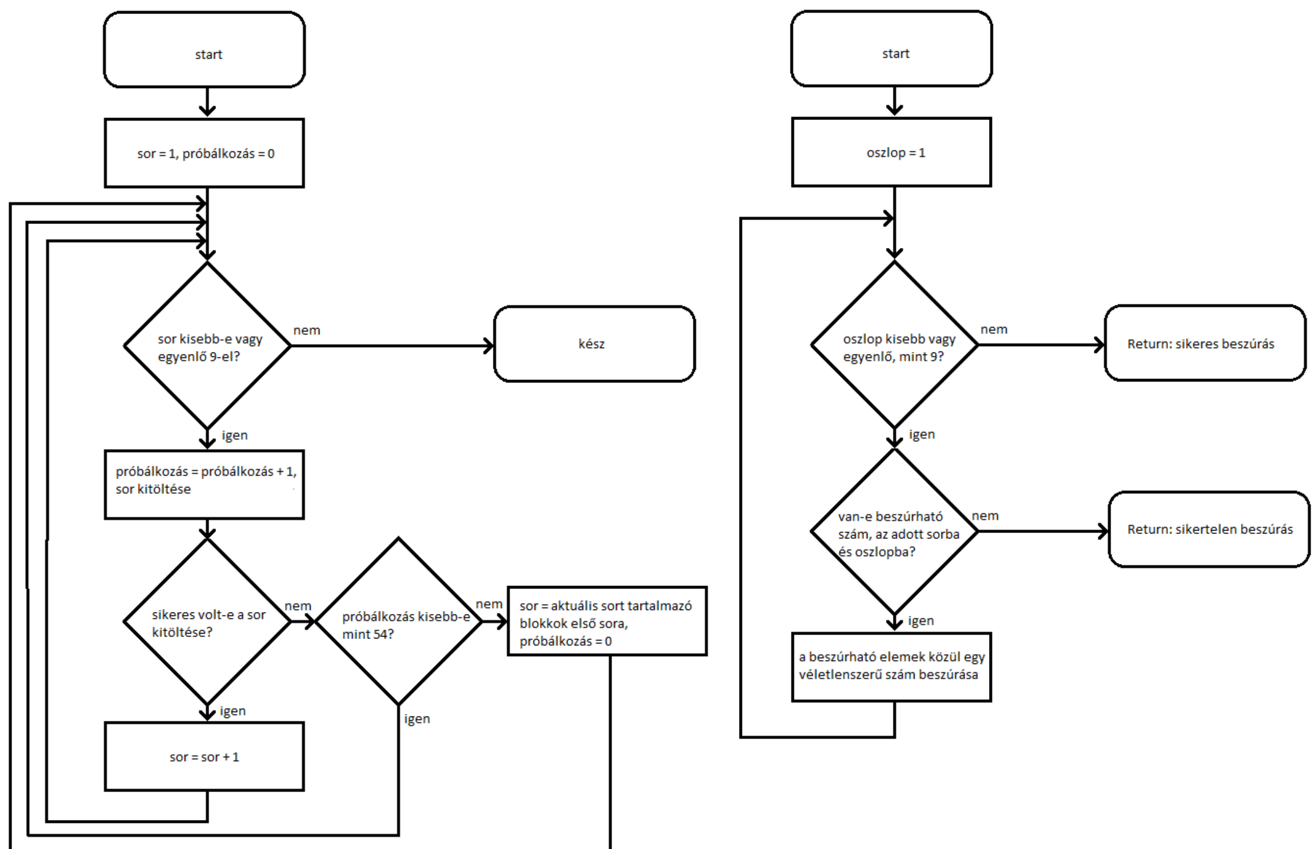
– Az adott sor minden elemét a következőképpen próbáljuk beszúrni:

– Megnézzük, hogy az adott pozícióban milyen értékeket lehet beszúrni (azaz mely számok nem szerepelek az adott sorban, oszlopban és blokkban).

– Ha nincs beszúrható szám, akkor maga a sor kitöltése sikertelen.

– Ellenkező esetben a beszúrható értékek közül véletlenszerűen egyet beír.

– A sor kitöltése sikeres, ha annak minden elemét kitöltötte az algoritmus.



11. ábra. Az előbb említett két algoritmus folyamatábrája

5.1.1. Adott játékállás megoldhatóságát eldöntő algoritmus

Egy játékállásról eldönti az alábbi algoritmus, hogy megoldható-e, ha megoldható, akkor visszaad egy nehézségi számot. Az algoritmus az alábbi:

- Az alábbi lépéseket addig ismételjük, amíg az egész játékállás nem tartalmaz üres mezőt vagy az előző lépésben is ugyanannyi volt a kitöltött mezők száma, mint most.
- 1. lépés: Minden mezőre megnézzük, hogy hány értéket lehet oda beszúrni úgy, hogy a Sudoku alapszabályainak az ne mondjon ellent. Amelyik pozícióba csak 1-et lehet, oda azt beszúrjuk és a nehézségi szintet szimbolizáló változót 3-al növeljük.
- 2. lépés: Minden egyes soron végigmegyünk és az adott soron belül megnézzük 1-től 9-ig mindegyik számra, hogy az adott szám az adott soron belül hány helyre szúrható be. Ha csak 1 helyre, akkor beszúrjuk azt és növeljük a nehézségi szintet szimbolizáló változót 3-al. Ha több, mint egy helyre akkor csak a nehézségi szintet szimbolizáló változót növelem 1-el.
- 3. lépés: Minden egyes oszlopon végigmegyünk és az adott oszlopon belül megnézzük 1-től 9-ig mindegyik számra, hogy az adott szám az adott oszlopon belül hány helyre szúrható be. Ha csak 1 helyre, akkor beszúrjuk azt és növeljük a nehézségi szintet szimbolizáló változót 3-al. Ha több, mint egy helyre akkor csak a nehézségi szintet szimbolizáló változót növelem 1-el.
- 4. lépés: Minden egyes blokkon végigmegyünk és az adott blokkon belül megnézzük 1-től 9-ig mindegyik számra, hogy az adott szám az adott blokkon belül hány helyre szúrható be. Ha csak 1 helyre, akkor beszúrjuk azt és növeljük a nehézségi szintet szimbolizáló változót 3-al. Ha több, mint egy helyre akkor csak a nehézségi szintet szimbolizáló változót növelem 1-el.
- Amennyiben véget ért a ciklus, azt kell megvizsgálni, hogy azért ért-e véget, mert minden mező ki lett töltve, vagy azért mert az utolsó és utolsó előtti ciklus közt nem volt beszúrás.
- Ha az előző miatt ért véget, akkor egyértelműen megoldható az alábbi játékállás, és a nehézségi szintet szimbolizáló változó ad egy metrikát a feladvány nehézségére vonatkozóan.
- Ha az utóbbi miatt ért véget, akkor a játékállás nem oldható meg egyértelműen, és a nehézségi szintet 0-ra állítjuk be, ami ezt szimbolizálja.

5.2. A SudokuField run() metódusa

Ez a metódus kezeli a felhasználó által kiadott alábbi inputokat: fel, le, jobbra, balra, és ok gomb megnyomása. Alapjában véve két részre lehet osztani ezt a metódust: a gomb nyomása a felső táblázatra van hatással, vagy az alsóra.

Ha az osztály isOk változója false értékű: Akkor a vezérlés a felső táblázaton van, és az alábbi gombok, a következő hatást érik el az alkalmazáson:

– Ok gomb: A felső táblázatot újrarajzolja úgy, hogy az eddigi sötétkék háttérrel kijelölt mezőt most szürke hátérrel jeleníti meg és az alsó táblázatot is megrajzolja. Mindeközben az isOk változó értékét true-ra állítja.

– Jobbra gomb: Töröli az egész képernyőt, majd a widthInField nevű változót növeli az alábbi értékkel: (mező szélesség) * (adott pozíciótól csigavonalban jobbra és le legközelebb eső módosítható mező között lévő mezők száma). Abban az esetben, ha az adott útvonalon nincs több módosítható mező, akkor a widthInField értéke nem változik. Következő körben a heightInField változó értékét növelem az alábbi értékkel: (widthInField — első mező távolsága a kijelző bal szélétől) egész osztás (egy mező szélessége * 9). A widthInField értékét a következő értékkel csökkentem: (widthInField — első mező távolsága a kijelző szélétől) maradékosztás (egy mező szélessége * 9). Ezek után a felső táblázatot újrarajzolom a frissített widthInField és heightInField értékekkel. Ezzel az algoritmussal érem el, hogy a jobbra gomb lenyomásának hatására a csigavonalban jobbra és le irányban a következő szerkeszthető mezőt tudjam kijelölni.

– Balra gomb: Töröli az egész képernyőt, majd a widthInField nevű változót csökkent az alábbi értékkel: (mező szélesség) * (adott pozíciótól csigavonalban balra és fel legközelebb eső módosítható mező között lévő mezők száma). Abban az esetben, ha az adott útvonalon nincs több módosítható mező, akkor a widthInField értéke nem változik. Következő körben a heightInField változó értékét csökkentem az alábbi értékkel: (widthInField — első mező távolsága a kijelző bal szélétől) egész osztás (egy mező szélessége * 9). A widthInField értékét a következő értékkel növelem: (widthInField — első mező távolsága a kijelző szélétől) maradékosztás (egy mező szélessége * 9). Ezek után a felső táblázatot újrarajzolom a frissített widthInField és heightInField értékekkel. Ezzel az algoritmussal érem el, hogy a balra gomb lenyomásának hatására a csigavonalban balra és fel irányban a következő szerkeszthető mezőt tudjam kijelölni.

– Lefele gomb: Töröli az egész képernyőt, majd a heigthInField nevű változót növeli az alábbi értékkel: (mező magasság) * (adott pozíciótól csigavonalban le és jobbra legköze-

lebb eső módosítható mező között lévő mezők száma). Abban az esetben, ha az adott útvonalon nincs több módosítható mező, akkor a heightInField értéke nem változik. Következő körben a widthInField változó értékét növelem az alábbi értékkel: (heightInField — első mező távolsága a kijelző tetejétől) egész osztás (egy mező magassága * 9). A heightInField értékét a következő értékkel csökkentem: (heightInField — első mező távolsága a kijelző tetejétől) maradékosztás (egy mező magassága * 9). Ezek után a felső táblázatot újrarajzolom a frissített widthInField és heightInField értékekkel. Ezzel az algoritmussal érem el, hogy a lefele gomb lenyomásának hatására a csigavonalban le és jobbra irányban a következő szerkeszthető mezőt tudjam kijelölni.

- Felfele gomb: Töröli az egész képernyőt, majd a heigthInField nevű változót csökkenti az alábbi értékkel: (mező magasság) * (adott pozíciótól csigavonalban fel és balra legközelebb eső módosítható mező között lévő mezők száma). Abban az esetben, ha az adott útvonalon nincs több módosítható mező, akkor a heightInField értéke nem változik. Következő körben a widthInField változó értékét csökkentem az alábbi értékkel: (heightInField — első mező távolsága a kijelző tetejétől) egész osztás (egy mező magassága * 9). A heightInField értékét a következő értékkel növelem: (heightInField — első mező távolsága a kijelző tetejétől) maradékosztás (egy mező magassága * 9). Ezek után a felső táblázatot újrarajzolom a frissített widthInField és heightInField értékekkel. Ezzel az algoritmussal érem el, hogy a felfele gomb lenyomásának hatására a csigavonalban fel és balra irányban a következő szerkeszthető mezőt tudjam kijelölni.

Ha az osztály isOk változója true értékű: Akkor a vezérlés az alsó táblázaton van. Természetesen bármilyen értéket 1-től 9-ig be lehet szúrni bárhova, ahol az adott mező szerkeszthető, illetve törölni onnan, amennyiben a cheat mód inaktív. Ha a cheat mód aktív, akkor van értelme beszúrható értéket tartalmazó mezőről beszélni. Az alábbi gombok, a következő hatást érik el az alkalmazáson:

- Ok gomb: A felső táblázatot újrarajzolja úgy, hogy az eddigi szürke háttérrel kijelölt mezőt most kék hátérrel jeleníti meg, ahova kék színnel az alsó táblázatban kijelölt számot kék színnel beszúrja. Az alsó táblázatot törli. Mindeközben az isOk változó értékét false-ra állítja.

- Jobbra gomb: Töröli az alsó táblázatot, majd a widthInNumpad nevű változót növeli az alábbi értékkel: (mező szélesség) * (adott pozíciótól csigavonalban jobbra és le legközelebb eső, beszúrható értéket tartalmazó mező között lévő mezők száma). Abban az esetben, ha a widthInNumpad értéke nagyobb mint (5 * mező szélesség), akkor a

heightInNumpad értékét növelem egy mező magasságnyival, a widthInNumpad-ot csökkentem az alábbi értékkel: widthInNumpad — (5 * mező szélessége). A módosított értékekkel újrarajzolom az alsó táblázatot. Ezzel az algoritmussal érem el, hogy a jobbra gomb lenyomásának hatására a csigavonalban jobbra és le irányban a következő beszúrható számot tartalmazó mezőt tudjam kijelölni.

- Balra gomb: Törli az alsó táblázatot, majd a widthInNumpad nevű változót csökkenti az alábbi értékkel: (mező szélesség) * (adott pozíciótól csigavonalban balra és fel legközelebb eső, beszúrható értéket tartalmazó mező között lévő mezők száma). Abban az esetben ha a widthInNumpad értéke negatív, akkor a heightInNumpad értékét csökkentem egy mező magasságnyival, a widthInNumpad-ot növelem az alábbi értékkel: widthInNumpad + (5 * mező szélessége). A módosított értékekkel újrarajzolom az alsó táblázatot. Ezzel az algoritmussal érem el, hogy a balra gomb lenyomásának hatására a csigavonalban balra és fel irányban a következő beszúrható számot tartalmazó mezőt tudjam kijelölni.

- Lefele gomb: Törli az alsó táblázatot, majd a heigthInNumpad nevű változót növeli az alábbi értékkel: (mező magasság) * (adott pozíciótól csigavonalban le és jobbra legközelebb eső beszúrható értéket tartalmazó mező között lévő mezők száma). Abban az esetben, ha az adott útvonalon nincs több beszúrható értéket tartalmazó mező, akkor a heightInNumpad értéke nem változik. Következő körben a widthInNumpad változó értékét növelem az alábbi értékkel: (heightInNumpad — első mező távolsága a kijelző tetejétől) egész osztás (egy mező magassága * 5). A heightInNumpad értékét a következő értékkel csökkentem: (heightInNumpad — első mező távolsága a kijelző tetejétől) maradékosztás (egy mező magassága * 5). Ezek után az alsó táblázatot újrarajzolom a frissített widthInNumpad és heightInNumpad értékekkel. Ezzel az algoritmussal érem el, hogy a lefele gomb lenyomásának hatására a csigavonalban le és jobbra irányban a következő beszúrható számot tartalmazó mezőt tudjam kijelölni.

- Felfelé gomb: Törli az alsó táblázatot, majd a heigthInNumpad nevű változót csökkenti az alábbi értékkel: (mező magasság) * (adott pozíciótól csigavonalban fel és balra legközelebb eső beszúrható értéket tartalmazó mező között lévő mezők száma). Abban az esetben, ha az adott útvonalon nincs több beszúrható értéket tartalmazó mező, akkor a heightInNumpad értéke nem változik. Következő körben a widthInNumpad változó értékét csökkentem az alábbi értékkel: (heightInNumpad — első mező távolsága a kijelző tetejétől) egész osztás (egy mező magassága * 5). A heightInNumpad értékét a következő értékkel növelem: (heightInNumpad — első mező távolsága a kijelző tetejétől) maradék-

osztás (egy mező magassága * 5). Ezek után az alsó táblázatot újrarajzolom a frissített widthInNumpad és heightInNumpad értékekkel. Ezzel az algoritmussal érem el, hogy a felfele gomb lenyomásának hatására a csigavonalban fel és balra irányban a következő beszúrható számot tartalmazó mezőt tudjam kijelölni.

Ennél a pontnál állva a fejlesztés során elmondhatjuk, hogy rendelkezünk egy lefordítható kóddal, illetve futtatható alkalmazással. Természetesen a játék nem feltétlen tökéletes, ezért lesz szükség a tesztelési fázisra.

6. Tesztelés

Ez a fázis két részből áll. Az első részben egy teszttervet adok meg, ami a tesztelő számára írja le azokat a lépéseket, amiket végre kell hajtania. A második részben az alkalmazás gyorsaságát fogom tesztelni.

6.1. Teszterv

Ez a dokumentum tartalmazza azokat a dolgokat, amiket a tesztelőnek ellenőriznie kell, továbbá azt is, hogy hogyan is tegye mindezt. Az előírttól eltérő működés esetén a tesztelőnek értesíteni kell erről a fejlesztőt, akinek javítania kell a hibákat. A hibák javítása során újabb tesztelésekre lesz szükség.

– Mind a 3 nehézségi szint mellett történő generálásnál ellenőrizni kell azt, hogy az előállítás sikeresen végbement, azaz a generálást végrehajtó algoritmusok egyike se tartalmaz végtelen ciklust, illetve generálás után a vezérlés továbbadódik a következő ablakra.

– Mind a 3 nehézségi szint mellett ellenőrizni kell, hogy az előállított játék érvényes-e, azaz minden sorban, oszlopban és blokkban azt kell leellenőrizni, hogy egy szám legfeljebb egyszer szerepel-e.

– Az előállított feladvány egyértelműen megoldható-e? Ezt a következő módon lehet tesztelni: Tetszőleges úton bejárom az egész pályát, majd az olyan mezőkbe, ahova csak egy számot lehet beírni, oda beírom azt. Ezt a lépést egészen addig ismétlem, amíg tudok egyértelműen számot beírni. Ha már nem tudok egyértelműen számot beírni és még maradt üres mező, akkor egy olyan mezőt kell kiválasztani, ahol a beszúrható számok száma a legkisebb. Ekkor a játékból annyi változatot kell készíteni, ahány értéket a kiválasztott mezőbe be lehet szúrni, és minden változatnak a kiválasztott mezőjébe beszúrok a többitől különböző, oda beszúrható értéket. Minden változatra ismétlem az algoritmust az 1. lépéstől egészen addig, amíg vagy nem marad üres mező, vagy van olyan üres mező, ahová nem lehet beszúrni egyetlen értéket sem. Utóbbi esetben az adott változat nem megoldható, ezért elvetjük. Ha a változat nem tartalmaz üres mezőt, akkor azt megoldásnak tekintjük. Akkor mondható egyértelműen megoldhatónak egy feladvány, ha egyetlen egy olyan változat létezik, ami megoldáshoz vezet.

– Tesztelni kell azt is, hogy egy nehezebb nehézségi szinttel előállított játékot megoldani a gyakorlatban is nehezebb-e megoldani. Ezt a problémát csak a játék különböző szinteken történő futtatása mellett lehet tesztelni.

- Különböző támogatott készülékeken kell tesztelni, hogy az ablak teljes része látható-e, azaz bizonyos elemek nem csúsznak-e ki a látótérből.
- Ellenőrizendő az, hogy a mezőt szimbolizáló vonalak határain belül esik-e a mezőben lévő szám.
- A táblázatokban történő koordináció során ellenőrizni kell, hogy a vezérlés soha ne kerüljön a táblázaton kívülre. A felső táblázatban minden nem módosítható mezőt át kell ugrani, az alsóban pedig cheat mód mellet a nem beszúrható értékeket. Tesztelésnél olyan szélsőséges esteket kell megvizsgálni, ahol az utolsó sorban, az első sorban, utolsó oszlopban, első oszlopban nincs módosítható érték. A teszteléshez fix játék megadását a következőképpen tudjuk megadni: A Sudoku osztály konstruktorába a generálást követően felül kell írni a generatedMatrix nevű mátrixot.
- Új játék menüpont kiválasztásánál tesztelni kell, hogy a generálás folyamata ugyanúgy működik-e, mint a játék elindításánál.
- Megoldás opció választása esetén azt kell ellenőrizni, hogy a menüpont kiválasztása előtti játékállás minden nem módosítható értéke megegyezik a megoldás ugyanazon pozíciójában lévő számmal, illetve, hogy maga a megoldás ténylegesen megoldás-e. Akkor tekintünk egy játékállást megoldásnak, ha nem tartalmaz üres mezőt, illetve egyik sorában, oszlopában és blokkjában sincs sorismétlődés.
- Segítség opció választása esetén azt kell tesztelni, hogy a menüpont kiválasztása előtti és utáni játékállás közti különbség annyi, hogy az utóbbi egy zöld színű, nem módosítható számmal többet tartalmaz egy olyan pozícióban, ahol az előző játékállásban üres mező volt. Meg kell nézni, hogy ezt a számot fogja-e tartalmazni a megoldás is ugyanebben a pozíciójában, melyet a megoldás opcióval tudunk lekérni.
- Validálás opció választása esetén azt kell ellenőriznie a tesztelőnek, hogy az előző játékállásban szereplő kék színű számok mindegyike az aktuális játékállásban vagy zöld vagy kék színűek lesznek és értékük nem változik. Ellenőrizni kell, hogy ahol az aktuális játékállásban zöld szám található, abban a pozícióban a megoldásban is ugyanaz a szám szerepel, illetve ahol piros szám, ott a megoldásban attól különböző szám van. Meg kell vizsgálni, hogy a zöld számokat tartalmazó mezőket lépegetésnél átugorjuk-e, illetve piros szám esetén azt, hogy tudjuk-e módosítani a mezőt.
- Cheat mód be opció választása esetén a tesztelőnek azt kell megnéznie, hogy a második táblázatban azokat a számokat, amiket a kijelölt mezőbe nem lehet beszúrni piros háttérrel

jelenjenek meg, illetve ezeket a számokat a lépkedés során átugorjuk. Tesztelendő az is, hogy amennyiben a cheat mód aktív, a menüben a „cheat mód ki" menüpont legyen látható.

- Cheat mód ki opció választása esetén azt kell tesztelni, hogy a program normál működés mellett fut, azaz bármilyen módosítható mezőbe bármilyen szám beszúrható. Tesztelendő tovább az is, hogy amennyiben a cheat mód inaktív, a menüben a „cheat mód be" menüpont legyen látható.
- A tesztelőnek ellenőriznie kell a rendszer reakcióját arra az esetre, mikor már nincs üres mező. Meg kell figyelnie, hogy kapunk-e valamilyen információt akkor, mikor az utolsó üres mezőt kitöltjük, és azt, hogy az helyes-e. Abban az esetben ha az adott játékállás megegyezik a megoldással, akkor a „Helyes megoldás" feliratot kell kapnunk, és onnantól kezdve nem lehet több számot beszúrni. Abban az esetben ha van legalább egy számismétlődés is valamelyik sorban, oszlopban vagy blokkban, akkor a „Hibás megoldás" üzenetet kellene kapnunk.

Amennyiben a program többszöri tesztelés és javítás után már több hibát nem tartalmaz, akkor elmondhatjuk, hogy előállt egy, az összes követelménynek eleget tevő, hibamentes verziója a játéknak. Ez a stabil verzió kiadásra kész.

Következő lépésben egy sebességtesztet hajtottam végre a játék leginkább időköltséges funkcióján, a generáláson. A követelmények közt szerepelt az is, hogy a játék élvezhető sebesség mellett működjön. Ez az érték eléggé relatív, azaz nehéz behatárolni egy maximumot, amit még elviselhetőnek mondunk, de azért már a tesztelés során tudunk véleményt nyilvánítani arra vonatkozóan, hogy elég gyors-e a játék. A sebesség tesztelése során 10 generálás sebességét mértem le mind a 3 nehézségi szinten. Az alábbi táblázat tartalmazza ezeket az információkat. A generálás sebességének a mérését a következőképpen végeztem el: Mielőtt meghívtam a generáló metódust, a rendszeridőt lekértem és eltároltam egy változóban. Végrehajtottam a generálást, majd a jelenlegi rendszeridőből kivontam a generálás előtti rendszeridőt. A kapott érték pontosan a generáláshoz szükséges idő. Ahogy a lenti táblázatból kiderül, maga a sebesség ezredmásodpercben mérhető, azaz elmondható, hogy az előállítás annyira gyors, hogy a felhasználó szinte észre se veszi azt, hogy időbe kerül a generálás. Megfigyelhető az is, hogy egy nehezebb játék előállítása átlagosan több időt vesz igénybe, ami a redukáló algoritmus alkalmazásának köszönhető.

	könnyű	**közepes**	**nehéz**
1. teszt	75 ms	115 ms	85 ms
2. teszt	50 ms	210 ms	121 ms
3. teszt	65 ms	97 ms	133 ms
4. teszt	62 ms	142 ms	122 ms
5. teszt	80 ms	29 ms	117 ms
6.teszt	84 ms	94 ms	131 ms
7.teszt	72 ms	40 ms	125 ms
8. teszt	102 ms	47 ms	93 ms
9. teszt	67 ms	57 ms	85 ms
10. teszt	87 ms	43 ms	109 ms
Átlagos sebesség	**74,4 ms**	**87,4 ms**	**112,1 ms**

12. ábra. Sebesség tesztelésének eredménye

7. Evolúció

A játék használata közben a felhasználó részéről egyéb új követelmények is felmerülhetnek. Azt a folyamatot, amikor a programozó az új követelmények függvényében módosítja az alkalmazást a megfelelő fejlesztési lépéseket betartva, evolúciónak nevezzük. Egy-egy új követelmény hatással lehet egy már meglévőre. Ebben az esetben legtöbbször optimalizálásról van szó. A következőkben néhány követelmény ajánlást fogok felvetni, amelyeknek az esetleges későbbi verziók eleget tehetnek.

– A jelenlegi 3 nehézségi szint mellé hozzá lehetne adni még kettőt: „nagyon könnyű" illetve „nagyon nehéz". A játék értékét feltehetően emelné az, hogy a különböző képességű és gyakorlattal rendelkező emberek tudásuknak megfelelő szinten tudnak játszani. Bármely játék esetében a nehézségi szint választásának lehetősége leginkább egy üzleti jellegű kérdés. A szoftverfejlesztő cégek ezzel az opcióval teszik populárisabbá terméküket, és természetesen nagyobb számú eladásokra is tesznek ezáltal szert.

– Talán a legnagyobb értéknövelő hatást a vizuális elemek javításával érhetnénk el. Legtöbbször ha a felhasználónak maga a grafikai megjelenés nem tetszik, akkor feltehetően sokkal ritkábban használja azt, függetlenül a játék összes többi funkcióitól. Mivel ez a játék logikai típusú, ezért alapból a felhasználó nem fog elvárni 3 dimenziós effekteket, ennek ellenére azért léteznek már 3D-s pályával rendelkező sakkprogramok. Jelen alkalmazás esetében olyasmi vizuális változtatásokat tudnék elképzelni, mint például az egyes mezőkhöz tartozó árnyékok megrajzolása, háttér megjelenítése, gazdagabb színhasználat.

– Az eddigi megoldás funkciót az alábbi módon lehetne javítani: Ne egyből a megoldást jelenítse meg, hanem egyszerre csak egy értéket szúrjon be a program, és az egyes beszúrások közt legyen egy maximum egy másodperces eltérés. Ez a változtatás ismét csak vizuális javítást szolgálna, a játék funkcionalitása, használhatósága rovására.

– A segítség funkciót úgy lehetne módosítani, hogy ne egy random helyre szúrja be az odaillő számot, hanem pontosan azt a mezőt töltse ki, amelyiket kijelöltük. Ezen funkció változtatása ismét nem biztos, hogy célszerű. Itt megoldás lehetne az, hogy mind a két funkció létezne egymás mellett, különböző névvel.

– A kijelző egy szabad területén meg lehetne jeleníteni néhány információt az aktuális játékállással kapcsolatban. A megjelenítendő adatok a következőek: üres mezők száma, generált mezők száma, esetlegesen lépés számláló.

– Annak függvényében, hogy mennyi idő alatt és hány lépésből oldotta meg a feladványt a játékos, a program adhatna egy pontszámot, amit a rendszer tárolna. Egy menüpontból

elérhetővé lehetne tenni a legjobb eredményeket. Ez a funkció arra ösztönözné a játékost, hogy a saját vagy más pontszámát megdöntse. Nyilván ezen ötlet alkalmazása inspirálna egy felhasználót arra, hogy többet használja ezt a játékot, illetve mások körében is terjessze azt.

- Tartalmazhatna a játék egy mentési funkciót, ami az aktuális játékállást mentené el, amit a játékos a készülék újraindítása után is be tudna tölteni. Ez a funkció akkor lehet hasznos, ha a játékosnak nincs annyi ideje, hogy kirakja az aktuális feladványt, de a későbbiekben még szeretné folytatni. Másrészt ezzel a funkcióval el tud menteni egy kezdeti feladványt, és tudja tesztelni önmagát a játékos, hogy mellyek az ő általa használt megoldás előállítását segítő algoritmusok közül a leghatékonyabbak.

- A program által használt megoldáskereső algoritmusok listáját lehetne bővíteni. Ezzel a bővítéssel azt érnénk el, hogy egyre bonyolultabb feladványt tudna készíteni a rendszer. A több nehézségi szinttel rendelkező verzióhoz érdemes lenne ezt a megoldást használni.

Sajnos a szakdolgozatom keretei közt az előbb említett funkciók megvalósítására nem volt elegendő időm, de amennyiben a későbbiek során lehetőségem lesz, akkor szívesen foglalkozok ennek az alkalmazásnak a továbbfejlesztésével.

8. Telepítési útmutató

A NetBeans lehetőséget nyújt arra, hogy build-eljünk egy Java ME projektet. Ezen folyamat sikeres lefutása után előáll egy jar kiterjesztésű fájl, ami a Javás környezetben egy tömörített fájl, ami lefordított java osztályokat és konfigurációs állományokat tartalmaz. Ahhoz, hogy egy mobilkészülékre feltelepítsük az alkalmazást, ezt a jar fájlt kell átmásolni a készülék adattároló egységére. A készüléken ezt a fájlt futtatható állományként kell indítani. Amennyiben az eszközön van Java futtató környezet, akkor az fogja végezni az alkalmazás telepítését. Amennyiben a telepítés hiba nélkül végbement, akkor a telepített Java alkalmazások listájából kiválasztva indítható a játék.

9. Összegzés

Szakdolgozatom teljes leírást nyújt az összes fejlesztési lépéshez, melyek a sudoku játék előállításához voltak szükségesek. Ezen dokumentáció elég pontos és részletes ahhoz, hogy egy programozó ez alapján megértse az általam megírt kódokat, sőt képes lenne egy következő verzió elkészítéséhez is.

Első körben egy rövid leírást tartalmaz a dolgozat arról, hogy mit is értünk mobilos környezetben történő fejlesztésről, hogyan megy végbe, milyen szabványok vannak.

Következő körben arról ejtek néhány szót, hogy mi késztetett arra, hogy szakdolgozatomat egy mobil készülékekre szánt sudoku játék megvalósításáról írjam.

Az ezt követő fejezetben feltárom az összes általam megfogalmazott követelményt, melyeket az elkészítendő alkalmazásommal szemeben állítottam fel.

A tervezési résznél leírtam a rendszer felépítését, a fontosabb osztályokat, illetve azok lényegesebb metódusait és adattagjait.

Az implementáció alatt elsősorban a feladványt előállító algoritmusokat írtam le, illetve azt az algoritmust, ami a táblázatban való lépkedést valósítja meg.

Tesztelési fejezetben azokat a dolgokat írtam le, amiket az alkalmazás tesztelése során mindenképp ellenőrizni kell. Ugyanennél a résznél készítettem egy táblázatot is, ami 10 különböző teszt esetében a generálás sebességét reprezentálja.

Az utolsó fejezet az evolúció, ahol olyan ajánlásokat írtam le, amik arra vonatkoznak, hogy egy esetleges későbbi verziója ennek a játéknak miket tartalmazhatna.

Köszönetnyilvánítás

Ezúton szeretném megköszönni azon tanáraimnak a közreműködését, akik lehetővé tették nekem azt, hogy ezt a játékot el tudjam készíteni. Külön szeretném kiemelni Dr. Kósa Márk Szabolcs tanár urat, aki koordinálta a dolgozat elkészítésének folyamatát, illetve hasznos tanácsokkal látott el.

10. Irodalomjegyzék

1. Pekk Roland: Alkalmazásfejlesztés mobiltelefonokra 2006.

2. Forstner Bertalan - Ekler Péter - Kelényi Imre: Bevezetés a mobilprogramozásba 2008.

3. Sudoku megoldás generátor: http://www.dreamincode.net/code/snippet3447.htm

4. Webes sudoku játék: http://mathijs.jurresip.nl/2006/11/26/java-sudoku-generator/