Sudoku algoritmusok implementálása és tesztelése

Diplomaterv sorszáma: 662/2015

Pataki István

Budapest

2016

Tartalomjegyzék

Ábrajegyzék

1. Bevezetés

- **a játék eredete:**

  A sudoku egy érdekes játék, amely a megfejtő logikus gondolkodását teszi próbára. Az elnevezés a japán „**Sú**dzsi wa **doku**sin ni kagiru" szavakból keletkezett, de a játék nem japán eredetű. A játék a 20. század 80-as éveiben a „Number Place" („Számok behelyettesítése") néven vált ismertté egy amerikai fejtörő magazin oldalain. A japánoknál is nagyon hamar ismertté és kedveltté vált, viszont Európában csupán 2005-ben vált közkedveltté ez a látszólag egyszerű játék. Ma a sudoku játék a napilapok és magazinújságok leginkább olvasott mellékleteiben szerepel.

  http://sudokuonline.hu/info/

  Mivel ezzel Euler foglalkozott sokat, vannak, aki tőle származtatják a sudoku-t, korábbi nevén a "latin négyzetet". Euler XVIII. századi svájci matematikus volt, aki mintegy 1100 könyvet és tanulmányt írt annak ellenére, hogy élete utolsó 12 évében vak volt. Azt tartották róla, hogy úgy számol, ahogy az emberek lélegeznek, vagy ahogy a sas repül.

  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, és szintén építész!) 1984-ben a "Nikoli" nevű japán rejtvény társaság átvette a rejtvényt, és a "sudoku" elnevezést adta neki (su -szám, doku - az egyetlen lehetséges elhelyezés). 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.

  http://www.logikaifeladatok.hu/sudoku.html#tortenet

- **játékszabályok:**

  Az alapszabály az, hogy (a klasszikus esetet véve alapul) a sorok – oszlopok – blokkok, melyek a munkaterületet alkotják, csak egyszer tartalmazhatják az 1-től 9-ig tartó számok egyikét. Így persze egy sorban / oszlopban / blokkban a számok mindegyike előfordul. A játékszabályok csupán csak ennyit határoznak meg, de csak gondoljunk bele, ez elég bonyolult és szabálykövető így is. Ezért foglalkoztam e témával behatóbban, és a megoldó algoritmusok elkészítésére is ez sarkallt legfőképp, és persze a kíváncsiság.

- **a játék változatai:**

  A játékot pedig az is tovább színesíti, hogy nem csak a klasszikus 9x9-es tábla létezik, melyre az 1-től 9-ig számokat kell a szabályok szerint elhelyezni, hanem létezik a 6x6-os, 16x16-os, 25x25-ös változat is. Az utóbbiakba van hogy kétjegyű számokat kell beírni, de van, hogy betűk helyettesítik a számokat.

  Továbbá kifejlesztették már az 5x9x9-es táblát is, ahol a klasszikus 9x9-es táblákat úgy csúsztatják egymásba, hogy a táblák sarkain lévő blokkok közösek.

E programot pedig azért hasznos megalkotni, hogy mint taneszköz lehet azok számára, akik újoncként ismerkednek eme játékkal. E programot ajánlott úgy is használni, hogy a felhasználó először megfejti papíron a játékot a hagyományos módon, majd a programot ellenőrzésül használja. De ha elakadna valahol a kezdeti szakaszban, akkor a program tud ebben is segíteni. Tehát számtalan ok miatt hasznos lehet a program megírása.

- **fogalom-magyarázat:**

  cella: A tábla legalapvetőbb területi egysége, amibe kell elhelyezni az értéket.

  sor: A vízszintesen elhelyezkedő, egy vonalba eső cellák.

  oszlop: A függőlegesen elhelyezkedő, egy vonalba eső cellák.

  blokk: Egy vastagabb vonallal jelölt, a klasszikus esetben 3x3-as cellák egysége.

  aktuális cella: Az a cella, melyen épp a műveletet végezzük.

  munkaterület: Az aktuális cellához kapcsolódó más cellák, melyek eme cellával egy sorban, oszlopban, illetve blokkban van.

  szegmens: A sor illetve oszlop egy adott blokkba eső része.

  lehetőség-érték: Érték, melyet a még meg nem fejtett cella eredményül felvehet.

  megoldás-érték: Az az érték, melyet a cella eredményül felvett.

2. Feladatspecifikáció

A feladat egy grafikus felülettel rendelkező Sudoku program elkészítése.

Ehhez:

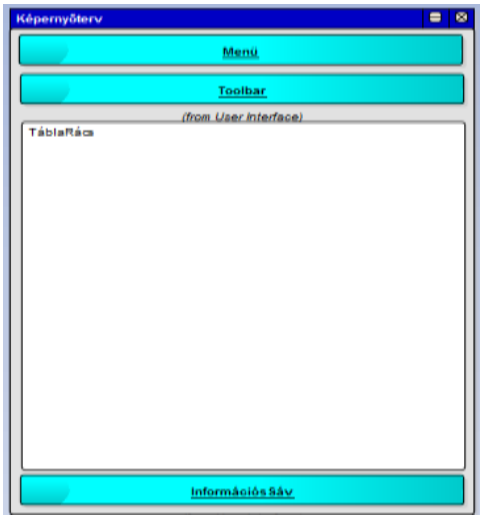
1., El kell készíteni az UML tervet és e dokumentációt.
2., Ki kell dolgozni a Rózsa-modell első 10 algoritmusa alapján a megoldó algoritmusokat. Az említett modellt a konzulens tanár biztosította.
3., Meg kell oldani az előre és visszalépkedést, miközben a program mutassa az aktuális tábla állapotát, és a talált megoldási eljárás nevét és helyét.
4., A folyamat végén jelezze a rejtvényfejtés sikerességét, és a lépések számát. Illetve jelezze, amennyiben az általam applikálandó 10 algoritmus alapján nem talált megoldást.
5., A táblákat le lehessen menteni és be lehessen tölteni.

2.1. Megvalósíthatósági vizsgálatok

A bevezetőben már leírt folyamatot elemezve azt kell látnunk, hogy a feladat több apróbb részre bontása után a feladat megoldását képező algoritmusok a megadott határidőre megvalósíthatóak. Ehhez azt is figyelembe kellett venni, hogy szétbontva a feladatot, először is el kell készíteni a grafikus felület tervét, de előtte szétnéztem a piacon, ki mit csinált már előttem e témakörben. Hosszú kutatások után arra jöttem rá, miután találtam sok sudoku implementációt, hogy az én általam készített programhoz még csak hasonlót sem találtam. Így önállóan kell létrehoznom a projectet.
Valamint van még egy fontos megállapodási tényező, a határidő. Abban állapodtunk meg, hogy ez az időpont **2016-os év vége**. Ekkorra a szoftvernek és a hozzá tartozó dokumentációnak el kel készülnie. Ez lesz az alkalmazás 1.0-ás változata. A határidő pedig tartható.

3. Tervezés

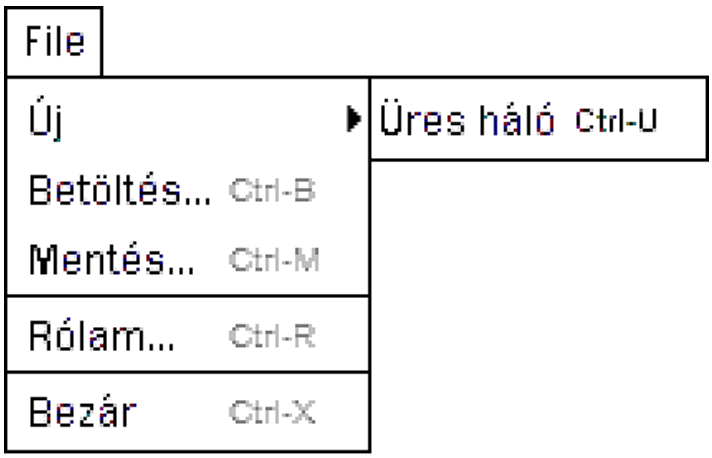
A megrendelővel egyeztetettek alapján az alkalmazás kezelőfelületének alábbi sematikus ábrája jött létre:



4. ábra: A felhasználói felület sematikus terve (forrás: Saját készítés)

Ezen az ábrán az látszik, hogy a képernyő majd 3 fő részből fog összeállni, úgy mint fej, törzs és láb. Úgy képzelem el, hogy ezeket majd egy-egy konténer fogja alkotni. A fej fogja továbbá tartalmazni a menüt és az eszközsort.

A megrendelővel lefixáltuk egymás közt azt is, hogy a szoftver menüje és toolbar-ja milyen elemeket tartalmazzon. így az alábbi képernyőterv részlet jött létre, mely más windows programmal való hasonlóság nem a véletlen műve, mivel előtte tanulmányoztam más winows-os alkalmazásokat.



5. ábra: A főmenü terve (forrás: Saját készítés)

Ezen kívül lefixáltuk, hogy milyen gombok legyenek a toolbar-on. Ezek sorrendben: töröl gomb, szám gombok 1-től 9-ig, új üres tábla, visszalépés és előrelépés gomb:

6. ábra: Az eszközsáv terve (forrás: Saját készítés)

Valamint az is rögzítve lett, hogy az információs sávban jelenjen meg a találat, és a hátralévő megoldatlan cellák száma.

Előzetesen ennyi volt lefixálva a megrendelővel, valamint a tábla tárolásával kapcsolatos kívánságok. Ezek alapján létrejött az alábbi használati-eset diagram:

4. ábra: Főbb használati esetek (forrás: Saját készítés)

Az ábra mutatja, hogy milyen metódusok fogják elvégezni majd a főbb műveleteket.

Most pedig nézzük, milyen algoritmusok végzik a megoldáskeresést...

3.1. Megoldó algoritmusok és matematikai alapjai

A Rózsa-modellt alapul véve, annak első 10 algoritmusát használom, valamint van egy úgynevezett takarító szabály, mely a Naked Single által fellelt eredményhez kapcsolódó cellákból kitörli az ott lévő, a szabály által talált értéket.
Az algoritmusok sorrendben:

- Naked single (tiszta egyes): Ennél a szabálynál azt kell megvizsgálni, hogy a cella egy értéket tartalmaz-e. Ha csak egyet, akkor ez lesz a megoldás.
  Ez végtelenül egyszerű algoritmus, ahol egy beágyazott dupla ciklussal a cellákat sorról-sorra, azon belül a sort oszlopról-oszlopra végigjárva megnézzük, hogy melyik cellában szerepel csak egyetlen lehetőség-érték. Ezt az értéket ezután le kell fixálni, majd ezt követi a takarítási folyamat, ahol a hozzá kapcsolódó cellákból ki kell törölni a megoldásértékkel megegyező lehetőség-értékeket. Ez pedig úgy történik majd, hogy megvizsgáljuk a sorokat, oszlopokat, majd blokkokat, hogy valamelyik cellában szerepel e ez a lehetőségérték. Ha nem szerepel az sem baj, hisz megoldás így is keletkezett.

- Hidden single (rejtett egyes): Olyan érték szolgál megoldásul, mely az adott munkaterület csak egy cellájában szerepel. Itt a sort, oszlopot, és blokkot megvizsgálva 1-től 9-ig az értékeket megnézve keresni kell olyan lehetőség-értéket, amiből az adott munkaterületen csak 1 pozíción található ez az érték. Így az érték megoldásul szolgál.
  Az algoritmus magja úgy néz ki, hogy az adott munkaterületen összeszámoljuk, hogy az adott értékből hány szerepel. Ha az eredmény 1, akkor abból a cellából, ahol megtalálta, a többi értéket törölni kell.

- Intersection: Ennek a szabálynak két esete van. Az első alapján megvizsgálandók a blokkok olyan szegmenseket keresve, ahol a blokk többi részében nem szerepel a jelölt, de az adott szegmenshez tartozó sorban illetve oszlopban található még az adott lehetőség-érték. Ekkor onnan ez az érték törölhető.
  A második esetben a sorok illetve oszlopból kiindulva vizsgáljuk azok szegmenseihez tartozó blokkokat, és ha a blokk további részében található még a lehetőség-érték, akkor az onnan törölhető.

Itt is a matematika halmazelméleti területén kell keresni a megoldást. Az adott sor és blokk, illetve oszlop és blokk metszetét kell vizsgálni ahhoz, hogy eljussunk a megoldáshoz. Ezt a metszetet neveztem el szegmensnek.

- Naked pair (tiszta pár):

  Olyan két cella keresése a cél, amelyben csak 2 érték szerepel és a 2 érték ugyanaz a két érték, valamint ebből a két értékből található a munkaterület e két celláján kívül eső más cellában is. Ekkor e utóbbi találat törölhető a lehetőség-értékek közül.

- Hidden pair (rejtett pár):

  Olyan két érték keresése, amelyek a munkaterület ugyanabban a két cellájában vannak, és a munkaterület további részében e két érték közül egyik sem szerepel. Valamint e két cellában található még további lehetőség-érték. Ekkor ezek a további lehetőség-értékek törölhetőek e két cellából.

- X-Wings (X-szárnyak):

  Ha teljesül, hogy egy lehetőség-érték két különböző sorban ugyanabban a két oszlopban szerepel. - pontosan két cella van, ami a jelöltet tartalmazza mindkét, ugyanolyan típusú munkaterületen, - és ezek a jelöltet tartalmazó cellák két ugyanolyan típusú munkaterületben, de az előző típustól eltérőben is benne vannak, akkor az utóbbi két munkaterületből eliminálható a jelölt többi előfordulása.

  Ezzel az általánosítással hat kombináció jön létre:

  - kiindulunk két sorból, és két oszlopból eliminájuk a jelöltet – Classic X-Wing,
  - kiindulunk két oszlopból, és két sorból eliminájuk a jelöltet – Classic X-Wing,
  - két blokkból indulunk ki, és két sorból eliminálunk,
  - két blokkból indulunk ki, és két oszlopból eliminálunk,
  - két sorból indulunk ki, és két blokkból eliminálunk,
  - két oszlopból indulunk ki és két blokkból eliminálunk.

  Valahogy az lenne az elgondolás, hogy általánosítani kellene a 6 esetet.

- Naked triple (tiszta hármas):

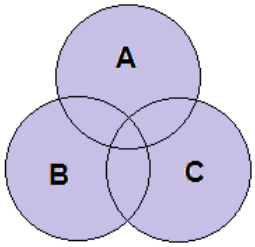
  Egy adott munkaterületet vizsgálva van e 3 olyan cella, ahol egy adott 3 elemű számhalmaz adja az értékkészletet, és legalább 2 elem szerepel az egyes cellákban.

Ekkor, ha van még a munkaterület más cellájában is ebből az értékből, akkor az onnan törölhető, így képződik megoldása az eljárásnak.

A probléma bonyolultsága abban rejlik, hogy egyes cellákban nem szükséges, hogy mind a három lehetőségérték szerepeljen, elegendő, ha legalább két érték található az adott számhalmazból.

A feladat megoldásául itt a matematika halmazelméleti fejezete szolgál. Vegyük egy olyan cella lehetőség-értékeit, ahol maximum 3 érték szerepel majd keressünk még másik 2 ilyet. Az így kapott 3 halmazt nevezzük el A, B, és C halmaznak.

Ha e 3 halmaz uniója egy 3 elemű halmazt képez, akkor kell megvizsgálni, hogy van e a munkaterület további részén további érték ezekből a lehetőség-értékekből. Ha található, akkor ezen további lehetőségértékek adják a megoldást.



A n számú, jelen esetben 3 cella összes eleméből egyet véve adja a halmazok unióját.

A besatírozott rész pedig nem más, mint az A ∪ B ∪ C, vagyis a három halmaz halmazelméleti összeadása.

- Hidden triple (rejtett hármas):

  Egy adott munkaterületet vizsgálva van e olyan 3 szám, amely csak az adott 3 cellában szerepel, és e 3 cella valamelyikében található további más jelölt-érték. Ekkor e további jelölt-érték törölhető.

  E feladat megoldásának matematikai alapjául az előző pontban alkalmazott unióképzés szolgál, csak nem a lehetőség-értékek alkotják a halmaz érékeit, hanem azok pozíciói, vagyis a munkaterületen belüli cellasorszámuk. Ez alapján vizsgálom meg azt, hogy az értékek az adott három cellába esnek e.

- Naked quad (tiszta négyes):

  Ez hasonló mint a Naked-triple, csak 4 értékre kiterjesztve.

- Hidden quad (rejtett négyes):

  Ez hasonló mint a Hidden-triple, csak 4 értékre kiterjesztve.

  És ekkor jött egy hatalmas felismerés, ami átütő sikert hozott. Nevezetesen, hogy gondoljunk bele, hogy a naked szabályok ugyanarra a sémára épülnek, csupán a keresendő értékek száma, illetve az őket tartalmazó cellák száma az amiben eltérnek.

Ugyanez igaz a hidden szabályokra is. Hisz mindezt le is írtam az egyes szabályok ismertetésénél.

- Újítás:

  Arra jöttem rá, hogy minden naked szabály, kivéve a single-t arra épül, hogy egy adott területen (sor, oszlop, vagy blokk) belül keresünk megoldást. Valamint valójában vizsgáljuk meg, hogy a sorok, oszlopok és blokkok átkonvertálhatók vektorokká. A sorok és oszlopok egyértelműek, míg a blokkok 3 sorát kell egymás mellé helyezni. Így kapunk vektorokat, és már csak a vektorban való keresésre kell koncentrálni.

  Amint látjuk, ez csak az egy-egy munkaterületformában való keresésre alkalmazható, vagyis a naked és hidden szabályokra, a többire ki kell dolgozni a saját algoritmusaikat, az összevonásra nincs lehetőség.

3.2. Adatok memóriabeli tárolása és modellje

E fejezetben írom le az aktuális állapotok tárolásának rendszerét és a kereső-algoritmusok által keletkezett eredmények adatainak tárolására szolgáló folyamatokat.
Előzetesen azt tudni kell, hogy a program azt a logikát követi, hogy nem a tábla aktuális állapotait tárolja le, hanem a lépések közötti változásokat.Ily módon, Előre lépés során a szükséges lehetőség-értékeket ki, megoldásértéket pedig bekapcsolom. Visszalépéskor pedig fordítva, a lehetőség értéket be, a megoldásértékeket pedig kikapcsolom.

Állapotok tárolása:

Itt azt a megoldást fogom alkalmazni, hogy egy osztályban (Cell) létrehozom az egy cellának a képét, vagyis elhelyezem rajta a 9 lehetőség-értéket és a megoldásértéket, valamint ez az osztály végzi el ezen értékek ki/bekapcsolását, és kijelöltségi állapotának kezelését.
Az alapelv az, hogy elhelyezek egy 3x3-as rácsra 9 szövegmezőt a jelölt-értékek, és egy továbbit az eredmény-érték megjelenítésére, és ez utóbbi érték tárolására.
Az ezt létrehozó metódus, ami voltaképp a Cell osztály konstruktora sematikusan így fog kinézni:

létrehozom
- tömb a jelölt-értékeknek
- tömb a jelölt-érték állapotának
- eredményt tároló elem
[ciklus] jelölt-értékek
létrehozom a vizuális elemet
elhelyezem az elemet
beállítom a betűtípust, méretet, ...
a jelölt-értéket megjelenítem
[ciklus vége]
elhelyezem a megoldás-értéket, ami kezdetben üres
beállítom a megoldás-érték vizuális tulajdonságait
a megoldás-értéket megjelenítem
beállítom a cellához tartozó koordináta értékeket
meghívom a színbeállítást végző metódust

Ezek után ebbe az osztályba fognak kerülni azok a metódusok, melyek ezeket az értékeket kezelik.

Ha ezzel megvagyok, ki kell alakítanom a játék háló felületét, ami értelemszerűen a már fent leírt cellából származtatódott 9x9-es hálóból tevődik össze. A cellákat pedig egy tömbhöz kell rendelni, hogy hozzáférést lehessen biztosítani minden cellához, és azon belül annak minden eleméhez. Majd az egységbe-zárás szabálya szerint a cellák közti összefüggéseket végző metódusokal egy Grid nevű osztályban heyezem el.

Eredmények tárolása:

Az osztálydiagrammban található 3 osztály, nevezetesen a Lists, LAssists, és LResults ostályok végzik el a függvények álltal talált eredmények tárolását és kezelését, amelyeket a Lists csomagban helyezek el.
Az **LAssists** osztályból származtatom az **LResults** osztályt, mely tartalmazza az alábbi adatmezőket: ruleOrder, value, comment.

Azt hamar felismertem, hogy az eredményérték és a lehetőségérték ugyanolyan formátumú adat, így azt ugyanabban a mezőben tárolhatom, viszont ahhoz hogy eldöntsem az érték melyik táborhoz tartozik, fel kellett vennem a valueType adatmezőt.
Az imént taglalt LResult osztályban is fel vannak sorolva az adatok tárolásához és visszanyeréséhez szükséges set / get metódusok, valamint a toString metódus, ami itt is

a tárolt adatok szöveges megjelenítésére szolgál, hogy olvashatóbb lehessen a tárolt eredmény abban az esetben, ha szeretnénk látni, hogy a listák milyen eredményeket tárolnak. ez például akkor hasznos, ha a programozás során valami hibás lépést tesz a szoftver, és ki szeretnénk deríteni, hogy a kereső motor hibázott, vagy az eredményeket megjelenítő algoritmusban van valami téves utasítás.

A fent említett két osztály tulajdonképpen arra szolgál, hogy a Lists osztályban létrehozzuk azokat a listákat, melyek a segéd cellák és az eredmény értékeket jelölő cellák koordinátáit és egyéb adatait tárolják és kezelik.
Itt természetesen létrehoztam két listát:
- assistList: a Coords osztályból van származtatva.
- resultList: a Result osztályból származtatott lista.
Azt pedig, hogy az egyes lépéseknek hol van a listában elfoglalt kezdő pozíciójuk, azt a stepPosition kétdimenziós tömb tárolja, melynek a nulladik oszlopa az assist pozíciót, míg az első oszlopa a result pozíciót tárolja.
Végül pedig van egy stepCount nevű adattag, amely azt tárolja, hogy hány lépést tartalmaz a tömb. Ily módon a lépések számát tárolja. Mivel a tömb private, így az ezekben tárolt adatokat a hozzájuk kapcsolt get/set metódusokkal lehet elérni.

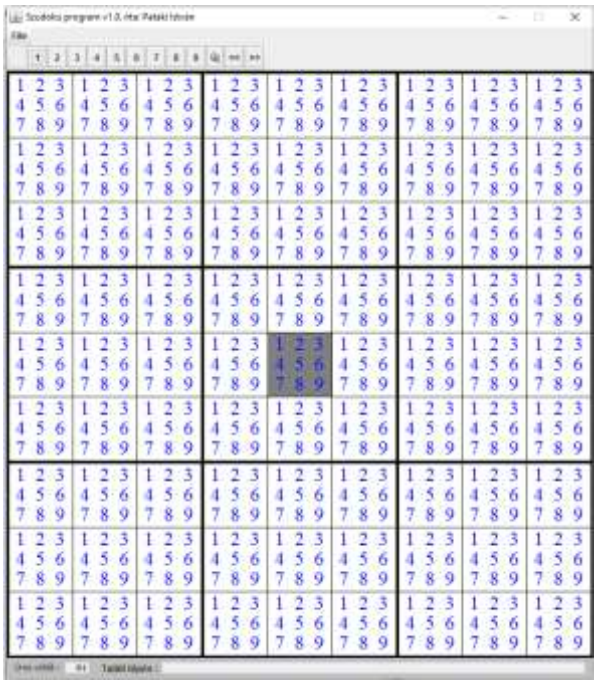
Tárolt eredmények megjelenítése:

Ha már egyszer letároltuk az adatainkat egy listába, akkor azokat meg is kell jeleníteni. A terv az, hogy mindig csak egy-egy lépést jelenít meg a szoftver. A lépéseket pedig - akár előre, akár visszafele - léptető gombokkal kell megvalósítani.

Ezen felül mód van arra is, hogy egy listába összefoglalva szövegesen mutatja a program, hogy lépésről lépésre mit talált, mindezt eredmény és segéd lista bontásban. Erre szolgál az eszközsor végén található ?? gomb.

3.3. Felhasználói felület és kezelése

Már a tervezés kezdeti szakaszában konzultáció folyt a megrendelővel arról, hogy az ő elképzelései szerint hogy nézzen ki a program kezelőfelülete. Abból kiindulva az alábbi ábra szerinti ablak keletkezett:



5. ábra: A program kezelőfelületének ábrája (forrás: Saját készítés)

A kezelőfelület felső részén található a menü, alatta pedig a toolbar, melyek képe már korai tervezési fázisban megalkotásra került. A menüben található menüpontok fedik a funkciójukat, így ez nem szorul magyarázatra.

A menü alatt található a toolbar, mely első tíz gombja az értékeknek a játéktérbe való felvitelére szolgálnak. Továbbá itt is el tudjuk érni a menüben már látott Új feladvány funkciót. E sor végén pedig található még 2 gomb, melyek az előre / vissza léptetésért felelős.

A táblázat feltöltése három alaplépés sorozatával valósul meg. Az első lépés a cella kijelölése, a második az oda beírandó érték megadása, majd a munkaterület többi cellájának aktualizálása.

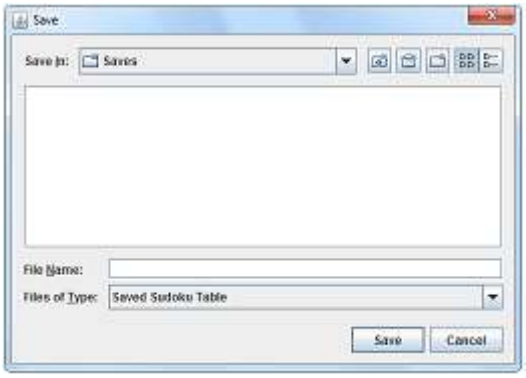
Cella kijelölése: Az aktuális cellát a program a cella szürkévé színezésével jelzi. Váltani pedig vagy az egérgomb lenyomásával, vagy a billentyűzeten lévő nyilak segítségével lehetséges. Itt szándékosan nem írtam bal illetve jobb egérgombot, mivel mindkettő

használatos, a bal egérgomb simán csak kijelöli a cellát, míg a jobb egérgombbal megjelenik egy helyi menü, de erről majd később beszélek még. Ám e mellett mód van a billentyűzeten lévő nyilak segítségével is váltani a cellák között. Ilyenkor a gombnyomásra a kijelölt cella a gombnak megfelelő irányú szomszédos cellára ugrik.

Érték beírása: Az aktuális cella tartalmát úgy tudjuk módosítani, hogy ha a cella üres, akkor oda értéket tudunk beírni, vagy ha már tartalmaz megoldás-értéket, akkor módunk van onnan kitörölni azt. Erre is használható a billentyűzet vagy az egér. Billentyűzet esetén be lehet írni az értéket, amennyiben az üres volt előzőleg, illetve Delete gombbal törölni az ott lévő értéket. A másik megoldás erre az eszközsávon kiválasztandó a kívánt érték, vagy ha ott már van érték, akkor a törlés gomb az aktív. A program vezérli, hogy mely nyomógombok legyenek megnyomhatóak a cellához tartozó lehetőség-értékek alapján.

Végezetül van egy további módszer is az értékek megadására, ez pedig a kívánt cellán jobb egérgombbal történő kattintással, illetve az Enter billentyű lenyomásával érhető el. Ilyenkor megjelenik a helyi menü, mely a beírható értékeket, illetve ha ott már van megoldás-érték, akkor a töröl menüpontot jeleníti meg. E menüpontok közül választva elérhetjük a kívánt hatást.

A cella aktualizálása után a program minden esetben elvégzi a munkaterület aktualizálását is, ami azt jelenti, hogy a munkaterület többi cellájából értékbeírás esetén kitörli az ott lévő, az adott megoldásértéknek megfelelő jelöltértéket. Törlésnél viszont fordított a helyzet, itt a munkaterület celláiból kikeresi a megoldásértéknek megfelelően azokat a jelöltértékeket, melyek esetében nincs hozzákapcsolva az adott munkaterülethez kapcsolódóan adott megoldásértékű eredményérték.
Így a felhasználó mindig a megfelelően aktualizált táblázattal tud dolgozni. A beírható értékeket a toolbar-on elhelyezett számgombok is mutatják. Ahol a szám-nyomógomb fekete színű, ott a szám egy lehetséges érték, ahol szürke, azt az értéket abba a cellába nem írhatjuk be. Ha eredményérték szerepel a cellában, akkor az üres nyomógomb az aktív, ami az eredményérték törlésére ad lehetőséget. A folyamat addig folytatódik, míg a feladványt a program számára nem adjuk meg.
Amikor végzett a felhasználó a feladvány megadásával, ajánlott a feladvány mentése, amit a menü ehhez tartozó menüpontjával érhetünk el. Ekkor megjelenik ez az ablak:



6. ábra: Mentés dialógusablak (forrás: Java dialogbox)

Az ablak File-name mezőjébe meg kell adni a file nevét, majd a save gombbal el lehet érni, hogy a feladvány eltárolódjon. Alapértelmezetten a program az aktuális időpontot kínálja fel, amelyet természetesen átírhatunk beszédesebb fájlnévre, sőt ajánlott.

Mód van a már eltárolt táblák betöltésére, ehhez a Betöltést segítő párbeszédablak szolgál.

A főablak alján folyamatosan látható, hogy hány cella nincs még megfejtve.

Ezután következhet a megoldáskeresés folyamata. A jobbra nyilat megnyomva el is indul a megoldáskeresés, ami az eddigi mérések alapján maximum 1-2 másodpercet vesz igénybe. A keresés sikeréről vagy sikertelenségéről egy üzenet tájékoztat minket, mely egy OK gomb megnyomására eltűnik, és kijelölésre kerülnek a találatot jelző értékek. A program azt is mutatja, hogy mely értékek vagy cellák voltak hatással az eredmény eléréséhez. A lépések közti navigálásra a toolbar-on elhelyezett jobbra illetve a balra nyíl szolgál. Ha a lépkedés során elértük a lépések határát, azaz ha sikerült a kereső algoritmusoknak megfejteni a feladványt, akkor az előre gomb elszürkül, azaz használhatatlanná válik. Az első lépés elérésekor pedig a vissza gomb szürkül el.

Az állapotsorban, ami a képernyő alján foglal helyet megjelenik az éppen aktuálisan még nem megfejtett cellák száma. Mellette pedig a találat szövege, vagyis az, hogy melyik algoritmus melyik részfolyamata talált rá a megoldásra.

A menüben található még továbbá egy Rólam menüpont, mely a program névjegyét jeleníti meg. Ezen található a program készítőjének az adatai, az alkalmazás készítésének éve, továbbá a program verziószáma, ami a mostani állapotot tekintve v1.0. A programot a Bezárás menüponttal, illetve a főablak jobb felső sarkában lévő piros X gombbal lehet bezárni, a windows hagyományoknak megfelelően.

3.4. Tárolási mechanizmus

E fejezetben azt szeretném ismertetni, hogy a tábla állapotát későbbi használat céljából háttértárra elmentési folyamat, hogy zajlik le, és milyen formátumban vannak eltárolva a rács adatai.

Az előző fejezetben említett Mentés és Betöltés menüpontok felelősek a tábla tárolási funkcióinak elvégzéséért.

A mentés során keletkezik egy szöveges állomány, amit egy DOS alapú karakteres szövegszerkesztő programmal, pl Notepad, meg tudunk nyitni.

Ez a szöveges állomány tartalmaz 9 sort, ami a tábla 9 sorának felel meg. Egy sort kiemelve azt láthatjuk, hogy számokat és kapcsos zárójeleket tartalmaz.

Mit is jelentenek ezek?

Két eset fordul elő:

1. A cella meg van fejtve, ilyenkor csupán a megoldás értéket tartalmazza a szöveg,
2. A cella nincs megfejtve, ilyenkor kapcsos zárójelek között fel vannak sorolva a lehetőségértékek.

Ezt a tárolási eljárást meg mondom őszintén, hogy loptam egy interneten található diplomamunkából, ám ott a cellákat határoló keretet egy space karakter jelöli, míg az én verziómban ez ki van hagyva, mivel felesleges.

Ezek az adatok úgy képződnek, hogy a Cell osztály toString metódusát hívom meg, mivel ott van applikálva az az algoritmus, mely megadja a cella tartalmát szöveges formátumban. A metódust meghívva minden egyes cellára, és az így kapott stringeket összeadva képződnek a sorok, majd a sorokat háttértárra kiírva az egész szöveges állomány.

Íme egy példa, mely bemutatja, hogy is néz ez ki a valóságban:

{8}29{1358}{158}764{15}
{4}{1347}5{1369}{1469}{1349}{179}28
6{1478}{47}{1589}{124589}{124589}{1579}{579}3
{24589}{3489}{234}{1589}{12589}6{1459}{359}7
{2459}{49}1{579}3{259}8{569}{2569}
7{389}{236}4{12589}{12589}{159}{3569}{12569}
1{79}{27}{356789}{56789}{3589}{579}{56789}4
35{47}{16789}{146789}{1489}2{6789}{69}
{49}682{4579}{459}31{59}

Ez a 01 Naked single.SST file tartalma is egyben. Itt látható, hogy a (0,0) cella nincs megfejtve, egy lehetőség-értéke maradt, a 8-as, a következő cella viszont már meg van fejtve, és a 2-es érték a megoldás. Az azt követő cella szintén megfejtett, és a 9-es érték a megoldás. Még egy eset, ha a kapcsos-zárójelek között több érték szerepel, úgy nyilván azok mind a cella lehetőség-értékei.

Hogy lássuk mindezt szemléletesebben, íme ugyanaz a tábla rácsokban megjelenítve, úgy ahogy a felhasználó is látja:

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

7. ábra: A táblázat grafikus megjelenítése (forrás: Saját készítés)

Egy file névkonvenciót követtem az így keletkezett állományok tárolásánál, hogy beírtam az algoritmus sorszámát, nevét, majd ha több példát is letároltam egy algoritmus teszteléséhez, akkor a végére elhelyeztem egy újabb sorszámot, ami ezáltal megkülönbözteti az ugyanahhoz az algoritmushoz tartozó másik példától. A táblák file-kiterjesztéséül az SST-t határoztam meg, mely egy általam kitalált elnevezés kezdőbetűiből áll össze: **S**aved **S**udoku **T**able.

A keletkezett állományokat pedig egy saves nevű almappában tárolom mely ha nem létezik, akkor az első mentéskor létrehozza a program. Ám hogy ne okozzon problémát az az eset, hogy mi van akkor, ha az alkalmazás első indításánál az az ötlete támad a felhasználónak, hogy meg szeretne nyitni egy tárolt file-t, ami persze nem létezik, ilyenkor is létre fogom hozni e mappát, így nyilván nem akad ki a program.

Mivel pedig nem szeretnék sok időt elpazarolni egy saját dialógus-ablak megírására, így a Java saját beépített ablakát használom majd fel. Ennek csak egy szépséghibája van, hogy túl általános, és a felhasználó el tud navigálni az általam megadott mappából.
Ezt kiküszöbölendően a továbbfejlesztési javaslatok közé fel lehet sorolni ennek a dialógusablaknak a megvalósítására szolgáló osztályt is.

4. Implementáció

E fejezetben felsorolásra kerülnek a program osztályainak attribútumai és metódusai az osztályok logikai kacsolódásának sorrendjében, ám először be kell mutatnom azt, hogy milyen folyamatosságot alátámasztó rendszert alakítottam ki.
A programírás során nagy hangsúlyt fektettem a Tiszta kód c. könyvben leírtak betartására. A könyv az objektum-orientált programozás elveinek gyakorlati aspektusait tárgyalja.
- 1. Az osztályok a lehető legszűkebb funkciót lássák el. Tehát az osztály a lehető legkevesebb feladatért legyen felelős.
- 2. Minél kevesebb paraméterezettséggel rendelkezzenek a főprogram metódusai.
- 3. Egy-egy osztály ne legyen hosszabb, mint 500 sor. No ezt nem mindenhol tartottam be, de igyekeztem figyelembe venni.
- 4. A metódusok lehetőleg maximum 25-30 sorosak legyenek, legalábbis törekedni rá. Első körben nagyon hosszú metódusokat készítettem, aztán többszöri átgondolás és átalakítás után sikerült elérnem a kívánt hosszakat.
- 5. A sorok hossza lehetőleg maximum 80 karakter legyen, mivel így vízszintes görgetés nélkül lehet olvasni a kódot. Bár ezt sem mindig tudtam betartani, de legalább igyekeztem.
- 6. Az osztályok adattagjai privátak, és az elérésükhöz az objektum-orientált alapelveknek megfelelően a gettereket / settereket használtam.

Mindezen szabályokat figyelembe véve sok kis osztályra, azon belül kis metódusokra bontottam a programot.

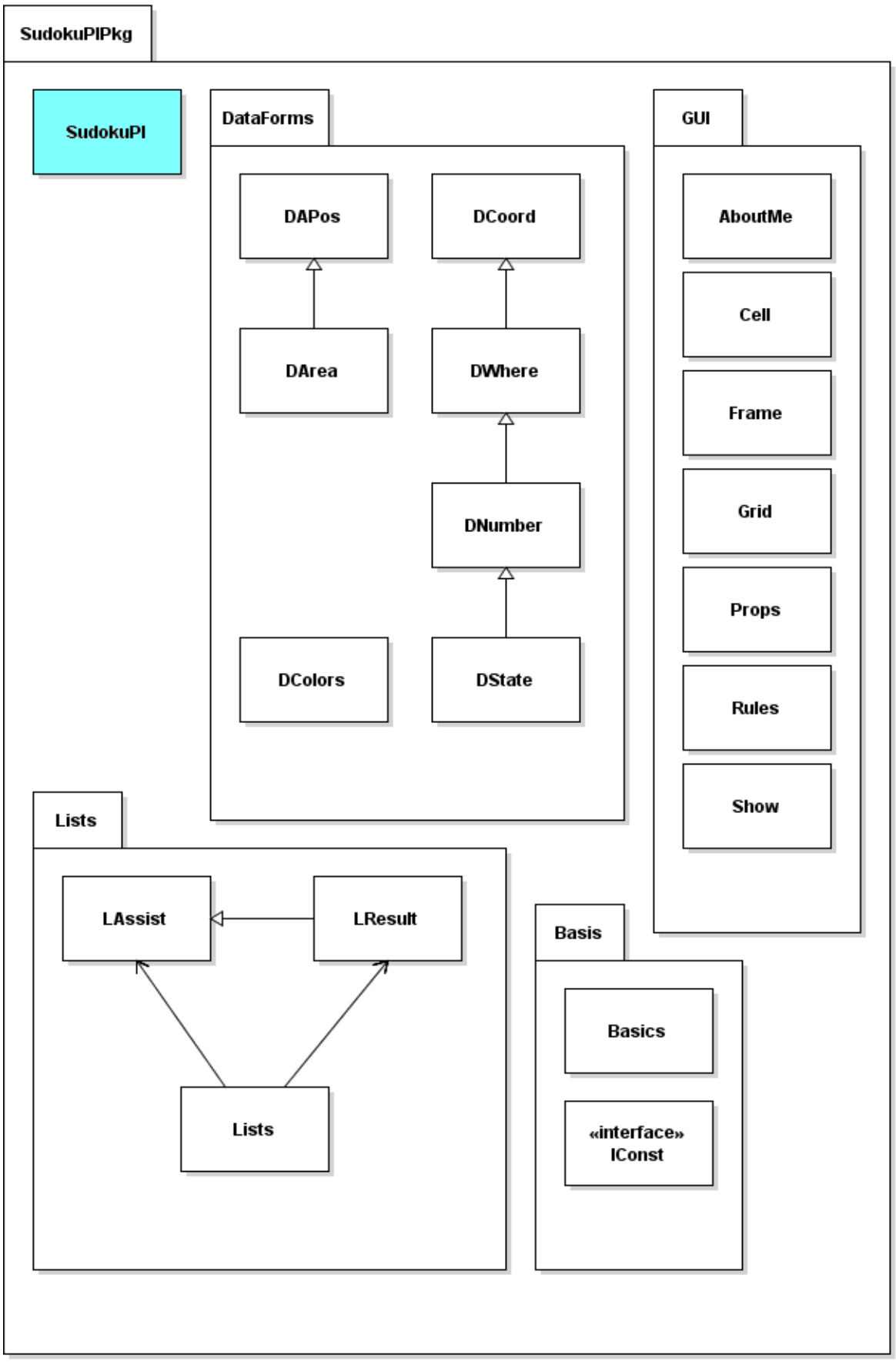
Konvenciók:

Kialakítottam egy olyan rendszert, hogy az azonos céllal használt változóneveknek mindenhol egységes elnevezéseket használtam, és ezt az implementálás kezdetekor lefektettem, mivel hamar rájöttem, hogy ezzel javul az átláthatóság és az értelmezhetőség. Ezek:

- row, row1, row2, ... : sor
- col, col1, col2, ... : oszlop
- c, c1, c2 : egy virtuális index
- blockNumber : a blokk sorszáma
- pos1, pos2, pos3, ... : talált pozíciók
- offset, offset1, offset2, ... : a blokkon belüli cellasorszám
- boY, boX : a blokk origójának koordinátái
- offsetY, offsetX : a blokkon belüli cellapozíció
- candidate, cand1, cand2, ... : lehetőség érték(ek)
- result, result1, result2, ... : eredmény érték
- where : művelet kapcsolata: Screen, vagy Memory

Az osztályok funkciói, attribútumai és metódusai:

Ezek alapján a programot az alábbi osztályokra bontottam szét azon elv szerint, hogy egy-egy egység kezelését külön osztály végezze el. Elnevezésként olyan szabályokat alakítottam ki magamnak az átláthatóság érdekében, hogy az osztályok neveit csoportosítottam, és egyedi elnevezéseket helyeztem el a neveik előtt. Ily módon az adattagokat megvalósító osztályok D betűvel, a listákat megvalósító osztályok L betűvel kezdődnek. Sőt, hogy ne egy mappába legyen ömlesztve mind, így külön mappákba helyeztem el az adathierarchiát (DataForms-ba), és a listákat (Lists-be) kezelő osztályokat.
Így jöttek létre az alábbi csoportok és osztályok:



8. ábra: A programot alkotó csomagok (forrás: Saját készítés)

DataForms:

- **DArea**: Munkaterület kezelését reprezentáló osztály.
- **DAPos**: Munkaterületen belüli pozíciót tároló osztály.
- **DCoord**: Az Y, X koordináták tárolására szolgál.
- **DWhere**: A rácsadatok helyének meghatározásához szükséges (Screen - Memory).
- **DNumber**: Adott cella értékének kezelését végző osztály.
- **DState**: Adott cella-érték állapotának kezelését végző osztály.
- **DColors**: Adott színséma színeinek tárolását ellátó osztály.

Lists:

- **LAssist:** A jelölt cellák tárolására használandó, ebből származik az LResult osztály.
- **LResult:** A keresési eredményeket tároló és kezelő osztály.
- **Lists**: A táblát megoldó folyamat lépéseit tároló osztály.

Basis:

- **Basics**: Alapfüggvényeket tároló osztály
- **IConst** - A programban használandó konstansokat tároló interface.

GUI:

- **AboutMe**: A névjegyet megjelenítő ablak osztálya
- **Cell**: Egy-egy cella értékeinek kezelését végző osztály
- **Grid**: A cellákat összefogó, és a köztük lévő kapcsolatok kezelését látja el.
- **Frame** - A grafikus felületet kezelő osztály.
- **Rules** - A megoldó algoritmusokat tároló osztály.
- **Show**: A megoldáskeresés folyamatát és az eredmények bemutatását végző osztály.
- **Props**: A beállításokat kezelő osztály.

fő mappa:

- **SudokuPI**: A szoftver belépési pontja, a fő-frame létrehozása és részeinek kialakítása.

Most pedig következhet a program részletes osztály-diagrammja, mely a csomagokba elhelyezett osztályok kapcsolódási rendszerét ábrázolja.

ide kerül az A3-as ábra ...

A metódusok megírásánál alkalmaztam a **túlterhelés** szabályt, ami azt jelenti, hogy a metódust nem csak a neve azonosít, hanem a paraméterek típusa, azok sorrendje és a paraméterek száma. Ha a paraméterek típusának eltérését alkalmazom, akkor ezt a paraméterlistában a / jellel elválasztott paramétertípusok felsorolásával jelzem.

Az osztályok csoportosított felsorolásában látottak szerint íme azok részletes leírása:

DArea osztály

Adott munkaterület reprezentációja.

Attribútumai:

- where: A képernyőre vagy a háttértömbre vonatkozik a művelet.
- areaForm: A területformát tárolja, azaz sor, oszlop, vagy blokkra vonatkozik e.
- area: Az adott területformájú munkaterület sorszáma.

Metódusai:

- DArea ()
  Paraméterek nélküli konstruktor.

- DArea (byte where, byte areaForm, byte area)
  Paraméteres konstruktor.

- setArea (byte where, byte areaForm, byte area)
  Alapértékek beállítása részletes paraméterezéssel.

- setArea (DArea area)
  Alapértékek beállítása összevont paraméterezéssel.

- setWhere (byte where)
  A where attribútum értékének beállítása.

- setAreaForm (byte areaForm)
  Az areaForm attribútum értékének beállítása.

- setArea (byte area)

  Az area attribútum értékének beállítása.

- getWhere () : byte

  A where attribútum értékének lekérdezése.

- getAreaForm () : byte

  Az areaForm attribútum értékének lekérdezése.

- getArea () : byte

  Az area attribútum értékének lekérdezése.

- toString () : String

  Az objektumban tárolt értékek szöveges megjelenítése az adott formátumban.

DAPos osztály

Adott munkaterületen belüli pozíció reprezentációja.

Attribútumai:

- offset: Az adott területformájú munkaterületen belüli pozíció.

Metódusai:

- DAPos ()

  Paraméterek nélküli konstruktor.

- DAPos (byte where, byte areaForm, byte area, byte offset)

  Paraméteres konstruktor részletes paraméterezéssel.

- DAPos (DArea area, byte offset)

  Paraméteres konstruktor összevont paraméterezéssel.

- setValues (DArea area, byte offset)

  Alapértékek beállítása összevont paraméterezéssel.

- setOffset (byte offset)

  Az offset attribútum értékének beállítása.

- getOffset () : byte

  Az offset attribútum értékének lekérdezése.

- toString () : String

  Az objektumban tárolt értékek szöveges megjelenítése az adott formátumban.

DCoord osztály

Adott munkaterület reprezentációja.

Attribútumai:

- row: Adott sor.
- col: Adott oszlop.

Metódusai:

- DCoord()

  Paraméterek nélküli konstruktor.

- DCoord (byte row, byte col)

  Paraméteres konstruktor, byte értékekkel.

- DCoord (float yx)

  Paraméteres konstruktor, lebegőpontos értékkel.

- DCoord (DCoord coord)

  Paraméteres konstruktor, összevont értékkel.

- setCoord (byte row, byte col)

  Értékek beállítása részletes paraméterezéssel.

- setCoord (DCoord coord)

  Értékek beállítása összevont paraméterezéssel.

- getCoord () : DCoord

  Értékek lekérdezése összevont paraméterezéssel.

- getRow () : byte

  A row attribútum értékének lekérdezése.

- getCol () : byte

  A col attribútum értékének lekérdezése.

- setRow (byte row)

  A row attribútum értékének beállítása.

- setCol (byte col)

  A col attribútum értékének beállítása.

- equals (DCoord coord) : boolean

  Adott koordináta értékek összehasonlítása az objektumban tárolttal.

- toString () : String

  Az objektumban tárolt értékek szöveges megjelenítése az adott formátumban.

DWhere osztály

Azt tárolja, hogy hol végzi a műveletet az adott eljárás.

Attribútumai:

- where: A képernyőre vagy a háttértömbre vonatkozik a művelet.

Metódusai:

- DWhere ()

  Paraméterek nélküli konstruktor.

- DWhere (byte where, DCoord coord)

  Paraméteres konstruktor.

- setValues (byte where, DCoord coord)

  Értékek beállítása részletes paraméterezéssel.

- getWhere () : byte

  A where attribútum értékének lekérdezése.

- setWhere (DWhere where)

  A where struktúra értékeinek beállítása.

- setWhere (byte where)

  A where attribútum értékének beállítása.

- toString() : String

  Az objektumban tárolt értékek szöveges megjelenítése az adott formátumban.

DNumber osztály

Az eredmény-értéket vagy a jelölt-értéket tárolja az osztály.

Attribútumai:

- number: Az eredmény-értéket vagy a jelölt-érték.

Metódusai:

- DNumber ()

  Paraméterek nélküli konstruktor.

- DNumber (DWhere where, byte value)

  Paraméteres konstruktor.

- setValues (DWhere where, byte value)

  Értékek beállítása részletes paraméterezéssel.

- getNumber () : byte

  A number attribútum értékének lekérdezése.

- setNumber (DNumber number)

  A number struktúra értékeinek beállítása.

- setNumber (byte number)

  A number attribútum értékének beállítása.

- toString() : String

  Az objektumban tárolt értékek szöveges megjelenítése az adott formátumban.

DState osztály

A jelölt illetve megoldás-érték állapotát tárolja.

Attribútumai:

- state: A képernyőre vagy a háttértömbre vonatkozik a művelet.

Metódusai:

- DState ()

  Paraméterek nélküli konstruktor.

- DState (DNumber number, boolean state)
  Paraméteres konstruktor.

- setValues (DNumber number, boolean state)
  Értékek beállítása részletes paraméterezéssel.

- getState () : boolean
  A state attribútum értékének lekérdezése.

- setState (DState state)
  A state struktúra értékeinek beállítása.

- setState (boolean state)
  A state attribútum értékének beállítása.

- toString() : String
  Az objektumban tárolt értékek szöveges megjelenítése az adott formátumban.

DColors osztály

Adott színsémához rendelt szineket tárolja és kezeli.

Attribútumai:

- actColors[]: Adott színsémához tartozó színkészletet tárolja.

Metódusai:

- setColors (Color[] colors)
  Bemásolja az aktuális színséma színeit.

- getColors () : Color[]
  Az aktuális színséma színeinek lekérdezése.

- getColorElement (byte element)
  Visszaadja az aktuális színséma adott szinét.

- setColorElement (byte element, Color color)
  Beállítja az aktuális színséma adott színét.

- toString() : String
  Az objektumban tárolt színek RGB értékeinek megjelenítésére szolgál.

Basics osztály

Ebbe az osztályba helyeztem el pár olyan általános függvény-metódust, melyek általános alap funkciókat látnak el.

Metódusai:

- deleteChar(String sText, String sChar) : String
  sText string-ből kitörli az összes sChar-ban tárolt karaktert, és az így kapott szöveg a metódus visszatérési értéke.

- getBlocknumberFromPosition(DCoord coord) : byte
  A függvény visszaadja, hogy az adott cella hányadik blokkba esik.

- getOffsetFromPosition(DCoord coord) : byte
  A függvény visszaadja, hogy az adott cella a blokk hányadik cellájába esik.

- getWhichContact(DCoord coord1, DCoord coord2) : byte
  Összehasonlít két koordinátát, és megállapítja viszonyukat, azaz hogy azonos-e, egy sorban, oszlopban vagy blokkban van-e.

- getCoordFromBlockPosition(byte blockNumber, byte position) : DCoord
  Visszaadja a blokk és pozíció alapján a koordinátát.

- insertChar(String sWord, String sChar) : String

  Karakter beillesztése a szövegbe, majd a szöveg karaktereinek rendezése.

  Az így kapott szöveg a metódus visszatérési értéke.

- isPositionInBlock(DCoord coord, byte blockNumber) : boolean

  Megállapítja, hogy a megadott cellapozíció az adott sorszámú blokkba esik e.

- OtherNumber(String numbers, byte number) : byte

  A numbers stringben tárolt 2 szám közül visszaadja azt ami nem a number.

- sortStringchars(String sText) : String

  A karaktertáblázatban elfoglalt helye szerint sorba rendezi az sText változóban átadott szöveg karaktereit.

- unionSets(String st1, ..., String st5) : String

  Az st1...st5 stringekben tárolt karakterek unióját képezi, azaz úgy egyesíti a stringeket, hogy a bennük lévő karakterek csak egyszer szerepelnek az eredmény-stringben.

- subtractionSets(String from, String what) : String

  A from stringből eltávolítja a what karaktert. Ha a what stringben több karakter van, akkor is csak az első karaktert kezeli.

- padL(String what, String extendChar, int length) : String

  A what szöveget balra igazítja úgy, hogy kiegészíti a szöveget az extendChar-ban megadott karakterrel a length hosszra.

- padR(String what, String extendChar, int length) : String

  A what szöveget jobbra igazítja úgy, hogy kiegészíti a szöveget az extendChar-ban megadott karakterrel a length hosszra.

- space(int length) : String

  Adott hosszúságú space karakterláncot állít elő.

- repeat(String what, int times) : String

  Megismétli az adott karakterláncot adott számszor, majd azt adja vissza eredményül.

IConst interface

A program olvashatóságának érdekében konstansokat vezettem be, amiket ebbe az interfészbe helyeztem el. Íme a szerkezete:

```
public interface IConst {

// ------------------------------------------------------- SWITCH or STATE
    /**
     * Konstans a bekapcsolt állapot jelöléséhez
     */
    public final boolean S_ON = true;

    /**
     * Konstans a kikapcsolt állapot jelöléséhez
     */
    public final boolean S_OFF = false;

...


// ------------------------------------------------------- COLORS AND SCHEMES

  int SCHEME_OCEAN = 0, SCHEME_NATURE = 1, SCHEME_TWILIGHT = 2, SCHEME_VAN_GOGH = 3;

  String[] SCHEME_NAME = {"Óceán", "Természet", "Estefelé", "Van Gogh"};

  Color INDIGO = new Color(76, 122, 179);
  Color VIOLET = new Color(128, 0, 128);
  Color BROWN = new Color(128, 64, 0);

// ------------------------------------------------------- ARRAYS

  String[] ELEMENTS = {"Eredmény cella:", "Segéd cella:", "Lehetőség-érték alap:",
    "Lehetőség-érték jelölt:  ", "Rács:"};

  String[] OP_MODE = {"Képernyő", "Memória"};

  String[] RULESSHOW = {"", "Naked Single", "Hidden Single", "Intersection",
    "Naked Pair", "Hidden Pair", "X-Wings", "Naked Triple", "Hidden Triple",
```

```
    "Naked Quad", "Hidden Quad"};

  String[] RULESNAMES = {"", "NakedSingle", "HiddenSingle", "Intersection",
    "NakedPair", "HiddenPair", "XWings", "NakedTriple", "HiddenTriple",
    "NakedQuad", "HiddenQuad"};

  String[] TOOLTIPS_TEXT = {"",
    "Ez a szabály azt vizsgálja meg, hogy van e olyan cella, melyben csak egy lehetséges érték található.",
    "Megvizsgálja a cellát, hogy a munkaterületen csak ott van e az adott érték.",
...

}
```

Az áttekinthetőség érdekében, mint látható felosztottam blokkokra, és ezeket jól elkülönítettem egymástól. Ezt jelöli a - karakterekkel jelölt vonal és a mögötte elhelyezett kategórianév.

LAssist osztály

Ez a tárolási mechanizmus eredendő osztálya. Így épül fel:

private short stepOrder; - Aktuális lépés sorszáma.

private DCoord coord; - Rácson belüli pozíció.

private boolean type; - A cella illetve érték típusa.

Az osztály 3 konstruktorral rendelkezik:

- 3 paraméteres: ilyenkor az adott értékekre állítja be a tárolandó értékeket.
- 2 paraméteres: ekkor a lépés sorszámát 0-ra állítja be.
- nincs paraméter: ekkor pedig az alapértelmezett null értékeket veszi fel, vagyis 3 darab 0-át és egy false-t.

Ezen kívül a 3 érték beállítására szolgáló set, illetve a lekérdezésükre szolgáló get metódusokat tartalmazza az osztály.

Végezetül a toString override metódust is alkalmazom, mely a Java hagyományoknak megfelelően visszaadja a tárolt értékeket

[stepOrder]. [row=Y, col=X]

alakban, ahol az Y és az X változók értelemszerűen felveszik az éppen aktuális értéket.

Attribútumai:

- stepOrder : lépés-sorszám
- coord : sor-oszlop pozíció
- type : Az érték illetve cella típusa

Metódusai:

Az adattagok és a konstruktoron kívül csak a megszokott set / get metódusok szerepelnek, valamint a toString metódus.

- LAssist (short stepOrder, DCoord coord, boolean type)
  Minden adattagot adott értékre beállító konstruktor.

- LAssist (DCoord coord, boolean type)
  A lépés sorszámát nélkülöző konstruktor.

- LAssist ()
  Konstruktor alapértékekkel.

- getCoord () : DCoord
  A cella rácson belüli pozícióját adja vissza.

- getStepOrder () : short
  A lépés sorszáma.

- setCoord (DCoord coord)
  beállítja a koordináta értékeket.

- setStepOrder (short stepOrder)
  Beállítja a lépéssorszám mezőt.

- toString () : String
  Visszaadja az adattagokat mondatba szerkesztett formában.

LResult osztály

A keresőfüggvények általi találatok tárolására szolgál.
Ennek az osztálynak is, a Coords listáknál leírtak szerinti konstruktora van.
Valamint az osztály tartalmazza a get/set metódusokat, amiken keresztül az osztályból létrehozott objektum mezőinek értékét beállítom, vagy lekérdezem.

Attribútumai: (private adattagok)

- ruleOrder: A keresőmetódus által talált szabály sorszámát tárolja.
- value: A talált érték.
- comment: Megjegyzés, ide az Intersection és az X-Wings ágait helyeztem el.

Metódusai:

- LResult (short stepOrder, byte ruleOrder, DNumber value, boolean valType, String comment)
  Minden adattagot adott értékre beállító konstruktor.

- LResult (byte ruleOrder, DNumber value, boolean valType, String comment)
  A stepOrder-t 0-ra, a többi adattagot az adott értékre beállító konstruktor.

- LResult ()
  Minden értéket az alapértékre beállító konstruktor.

- getResultText() : String
  Visszaadja a megoldás szövegét.

- getValue() : byte
  Visszaadja a talált értéket.

- getValueType() : byte
  Visszaadja azt az adattagot, mely megmondja, hogy az érték megoldás-, vagy lehetőség-érték.

- getWhere() : byte

  Megmondja, hogy a megoldás sorban, oszlopban vagy blokkban található e meg.

- toString() : String

  Olvasható formában visszaadja a rekordban tárolt értékeket.

Lists osztály

A keresési algoritmusok eredményeit tároló és kezelő osztály.

Attribútumai: (private adattagok)

- ArrayList<DCoords> assistList: E lista a megoldást segítő cellák pozícióit tárolja.
- ArrayList<Result> resultList: Ebben a listában vannak letárolva a megoldások.
- short[800][2] stepPosition: Azt tárolja, hogy a fenti két listában (assistList és resultList) melyik pozíciótól kezdődnek az adott lépéshez kapcsolódó adatok. E tömb 0. oszlopa jelöli az Assist pozíciót, míg az 1. oszlop a Result pozíciót.
- stepCount: a keresőfüggvények által talált lépések száma.
- stepOrder: Az aktuális lépés sorszáma

Metódusai:

- addAssist (short stepOrder, DCoord coord, boolean cellType)

  Az adott adatokkal feltöltött új adatsort illeszt a AssistList végéhez.

- addResult (short stepOrder, byte ruleOrder, DNumber value, boolean valueType, String comment)

  Az adott adatokkal feltöltött új adatsort illeszt a resultList végéhez.

- clearAssistList ()

  Kitörli az assistList lista sorait.

- clearResultList ()

  Kitörli a resultList lista sorait.

- clearAssistAndResultLists ()

  Kiüríti a két listát.

- incStepCount ()

  Növeli a lépések számát.

- decStepOrder ()

  Csökkenti az aktuális lépés sorszámát, ha még nem érte el a lépéslista elejét.

- incStepOrder ()

  Növeli az aktuális lépés sorszámát, amennyiben ez lehetséges.

- setAssistPosition (short stepOrder, short position)

  Beállítja a tömb stepOrder-edik sor, 0. oszlopába a position értéket, ami a kezdőpozíciót jelöli az assist listában.

- setResultPosition (short stepOrder, short position)

  Beállítja a tömb stepOrder-edik sor, 1. oszlopába a position értéket, ami a kezdőpozíciót jelöli a result listában.

- getAssistPosition (short stepOrder) : short

  Visszaadja, hogy az assistList-ben melyik sortól kezdődik az adott lépés.

- getResultPosition (short stepOrder) : short

  Visszaadja, hogy a resultList-ben melyik sortól kezdődik az adott lépés.

- getAssistSize () : short

  Visszaadja az assistList sorainak a számát.

- getResultSize () : short

  Visszaadja az resultList sorainak a számát.

- setStepCount (short value)

  Az adott értékre beállítja lépésszámlálót.

- setStepOrder (short stepOrder)

  Az adott értékre beállítja az aktuális lépést.

- getStepCount () : short

  Visszaadja a lépések számát.

- getStepOrder () : short

  Visszaadja az aktuális lépés sorszámát.

- getAssistStepOrder (short index) : short

  Visszaadja az assistList adott sorának stepOrder mezőjében tárolt értéket.

- getResultStepOrder (short index) : short

  Visszaadja a resultList adott sorának stepOrder mezőjében tárolt értéket.

- isActualStepLast () : boolean

  Az aktuális lépés elérte e már az utolsó lépést.

- getAssistCoord (short index) : DCoord

  Visszaadja az assistList adott sorának coord mezőjében tárolt értéket.

- isAssistTypeResult (short index) : boolean

  A cella típusa.

- getResultCoord (short index) : DCoord

  Visszaadja a resultList adott sorának coord mezőjében tárolt értéket.

- getRuleNameByOrder (short ruleOrder) : String

  Lekérdezi a szabály nevét a sorszáma alapján.

- getResultRuleOrder (short index) : byte

  Visszaadja a resultList adott sorának ruleOrder mezőjében tárolt értéket.

- getResultValue (short index) : byte

  Visszaadja a resultList adott sorának value mezőjében tárolt értéket.

- isResultTypeResult (short index) : boolean

  Visszaadja a resultList adott sorának valueType mezőjében tárolt értéket.

- getResultShowComment (short index) : String

  Visszaadja a resultList adott sorának kiírandó megjegyzésének értékét.

- toString ():String

  Az assistList és resultList tartalmát összeállítja egy stringbe.

AboutMe osztály

Az osztály szerepe annyi, hogy megjeleníti a szoftverről és annak készítőjéről szóló adatokat. A képernyőn megjelenik az alábbi ablak:

10. ábra: Az "AboutMe" ablak (forrás: Saját készítés)

Cell osztály

Ez az osztály tartalmazza az egy cellának a képét, és a benne lévő értékek állapotát kezeli.

A candidates vektorban (egydimenziós tömb) vannak láthatóan a lehetőség-értékek. Konstruktorának tartalmát a 3.2-es fejezetben már részben elemeztem.
Ám található még itt egy fontos elem, az x és y koordináta. Ehhez felvettem a két mezőt összefogó DCoord típusú attribútumot, mely tárolja a cella panelrácsban elfoglalt pozícióját.

A koordináta ahhoz kell, hogy majd a cella-kattintás eseménykezelője meg tudja mondani, hogy melyik cellán lett megnyomva az egér bal illetve jobb gombja.

```
private JLabel[] candidates = new JLabel[10]; // 1..9
private JLabel result = new JLabel("");

 public Cell(DCoord coord, DColors colors) {
  byte row, col;
  setLayout(null);
  for (byte cand = 1; cand < 10; cand++) {
   row = (byte) (((cand - 1) / 3) * 20 + 2);
   col = (byte) (((cand - 1) % 3) * 20 + 2);
   candidate[cand] = new JLabel("" + cand);
   candidate[cand].setBounds(col, row, 20, 20);
   candidate[cand].setHorizontalAlignment(SwingConstants.CENTER);
   candidate[cand].setFont(new Font("Times New Roman", Font.PLAIN, 18));
   add(candidate[cand]);
  }
  result.setHorizontalAlignment(SwingConstants.CENTER);
  result.setFont(new Font("Times New Roman", Font.PLAIN, 40));
  result.setBounds(10, 10, 40, 40);
  result.setVisible(false);
  add(result);
  setCoord(coord);
  repaintColors(colors);
 }
```

Attribútumai:

- candidatesCounter: A cellában lévő még élő lehetőségértékek számát tárolja.
- coord: a cellának a rácson belüli koordinátája.

Metódusai:

- Cell ()

  E paramétermentes konstruktor ahhoz kell, hogy kialakítsuk az objektum pointereinek tárhelyét.

- Cell (DCoord coord, DColors colors)

  Paraméteres konstruktor, a pozíció és a színkészlet átvételéhez.

- init (DCoord coord, DColors colors)

  A cella tartalmának kialakítása. Ezt tartalmazta a fentebbi kódrészlet.

- repaintColors (DColors colors)

  Beállítja a cella elemeinek színét.

- clearResult ()

  Kitörli az eredményt a cellából, majd bekapcsol minden lehetőségértéket, így ezután az aktuális sor, oszlop és blokk alapján át kell frissíteni. A frissítési folyamat mivel több cellát érint, így az ezt végző metódust a MyGUI osztály tartalmazza.

- isCandidateStateOn (byte/String candidate) : boolean

  Visszaadja, hogy a lehetőségérték be van-e kapcsolva.

- getCandidatesCounter () : byte

  Lekérdezi, hogy hány lehetőségérték van bekapcsolva.

- isResultStateOn () : boolean

  Visszaadja, hogy a cella meg van-e fejtve.

- getResult () : String

  A cella eredményértékét adja vissza. Ha üres a viszaadott string, akkor a cella nincs megfejtve.

- getValidCandidates () : String

  Visszaadja a cella lehetőség-értékeit.

- recountCandidates ()

  Újraszámolja a cella lehetőségértékeit.

- setCandidateMark (byte/String cand, boolean state)

  Beállítja az adott lehetőségérték jelöltségi állapotát.

- setCandidateState (byte/String cand, boolean state)

  Beállítja az adott lehetőségérték láthatósági állapotát.

- setCandidatesCounter (byte counter)
  Felülírja a bekapcsolt állapotban lévő lehetőségértékek számát.

- setCell (String cell)
  Beállítja a cella tartalmát.

- setResult (byte/String candidate)
  Az adott számot beállítja megoldásértéknek, majd a bekapcsolt állapotban lévő lehetőségeket kikapcsolja.

- decCandidatesCounter ()
  1-el csökkenti a cellában bekapcsolt lehetőségértékek számlálóját.

- incCandidatesCounter ()
  1-el növeli a cellában bekapcsolt lehetőségértékek számlálóját.

- getCoord () : DCoord
  Lekérdezi a cellának a rácson belüli pozícióját.

- setCoord (DCoord coord)
  Beállítja a cellának a rácson belüli pozícióját.

- toString () : String
  Visszaadja a cella értékkészletének kivonatát. A visszaadott érték formátumát a 3.4-es fejezet már tárgyalta.

Grid osztály

Az osztály a cella-rács képét alakítja ki, továbbá a cellák közti összefüggéseket kérdezi le. Ez utóbbi szolgál alapul a keresési szabályok algoritmusainak.
Létrejöttének pillanatában először az az ötlet rajzolódott meg a fejemben, hogy vonalakat fogok húzni, hol keskenyebbet, hol meg vastagabbat. Majd eszembe jutott, hogy jobb lenne, ha az objektum-orientáltság elveinek bemutatása céljából a jelenlegi modell született meg.

Egy kis trükköt alkalmaztam, nevezetesen nem rajzoltam ki egy vonalat se a képernyőre. A pBody panel színét, mely a rács konténere feketére változtattam, ami majd a rács színe lett, és a cella lapjai között réseket hagyva megalkottam a rácsot. Valamint a blokkokat határoló keretet megjelenítendően további 3 pontnyi helyet hagytam ki a cellák között.
Az egyes cellákat tartalmazó panel hozzá van kapcsolva egy 2 dimenziós tömbhöz, melyen keresztül ily formában a cellákat kezelni lehet.
A tömb deklarációja: private Cell[][] screenCells = new Cell[9][9];. Ezt csak azért említem itt meg, mivel a screenCells tömb is tárol adatokat. Hisz az üres tábla feltöltése, azaz a feladvány megadása a screenCells-hez kapcsolódó folyamat. A bekapcsolt lehetőség-érték a panelen az egyenlő azzal, hogy a cellában azt az értéket a cella még felveheti, továbbá ha a cella adott paneljén a lehetőséhgértékek ki vannak kapcsolva, és csak a megoldás-érték látható, akkor az azt jelenti, hogy az adott cella meg van fejtve. Így tárolja a tábla értékeit, csak e közben persze látjuk is közvetlenül a tartalmat, míg a memoryCells-ben tárolt érték közvetlenül nem látható. Ennek deklarációját mutatja e kódrészlet:

```
private String[][] memoryCells = new String[9][9];,
```

mely a screenCells toString metódusának megfelelően tárolja le a cellák lehetőség-értékeit és eredmény értékeit.
Végezetül már csak 1 dologra volt szükségem, hogy beszámozzam a rács paneljeit annak érdekében, hogy az eseménykezelő vissza bírja adni, hogy melyik cellát választottam ki. Ezért felvettem a Cell osztályba két mezőt, gridRow és gridCol, melyek a rácsban elfoglalt pozíciójukat jelölik, és a létrehozásukkor állítom be, a cellában lévő értékek színeivel együtt.
Természetesen a rács kezeléséhez is tartozik sok metódus, mely a rács cellái közötti összefüggéseket kezeli, de ezzel majd a későbbi fejezetek foglalkoznak.
Ám a képernyőn lévő táblán kívül van egy két dimenziós tömb, mely a paneles-cellás rácshoz hasonlóan a cellák értékeit tárolja. A különbség pedig az, hogy ebben a tömbben csak egy string van, melynek tartalmát a 3.4-es fejezetben részletezem.

Attribútumai:

- Cell[][] screenCells: A képernyőre kikerülő cellarács tárolója
- String[][] memoryCells: A memóriában lévő cellarács tárolója
- screenEmptyCells: A képernyőrács üres celláinak a száma
- memoryEmptyCells: A memóriarács üres celláinak a száma
- editMode: Azt tárolja, hogy a rács szerkeszthető e

- candidatesCounter: A cellában lévő még élő lehetőségértékek számát tárolja.
- actCell: a cellának a rácson belüli aktuális koordinátája.
- prevCell: cellaváltás előtti pozíció.

Metódusai:

- Grid (Frame owner)
  Konstruktor, mely meghívja az init metódust..

- init ()
  Kialakítja a játékteret, beállítja a színeit, és hozzárendeli az eseménykezelőt.

- myPopup ()
  A cellákhoz tartozó popup menüt hozza létre.

- repaintAllColors ()
  Átszínezi a játékteret

- setupAndShowPopup (MouseEvent me)
  A popup menü menüpontjait állítja be a toolbar gombjai alapján és megjeleníti.

- clearResult (DWhere where)
  Az adott cella Result értékét törli.

- isCandidateStateOn (DNumber number):boolean
  Az adott cella adott lehetőség-értékének állapotát adja vissza.

- getCandidatesCounterInCell (DWhere where):int
  Visszaadja az adott cella lehetőség-értékeinek számát.

- getCandidatePositionsInArea (DArea area, byte candidate):String
  Visszaadja, hogy az adott munkaterületen az adott jelöltek mely pozíciókon találhatóak.

- getCell (DWhere where) : String
  Visszaadja egy adott cella tartalmát

- getCoordFromPosition (DAPos aPos) : DCoord
  A területen belüli pozícióból valódi koordinátát állít elő.

- getCountCandidateInArea (DArea area, byte candidate) : byte
  Visszaadja, hogy az adott munkaterületen az adott jelöltek hányszor szerepelnek.

- recountEmptyCells (byte where) : short
  Összeszámolja az üres cellákat.

- isCellResolved (DWhere where) : boolean
  Lekérdezi, hogy az adott cella meg van e fejtve.

- getResult (DWhere where) : String
  Visszaadja az adott cella eredmény-értékét. Ha nincs megfejtve a cella, akkor a visszaadott érték egy üres string.

- getResultPositionsInArea (DArea area, String result) : String
  Visszaadja, hogy az adott eredményérték mely pozíciókon szerepel az adott területen.

- getValidCandidates (DWhere where) : String
  Visszaadja egy sztringben az adott cella lehetőség-értékeit

- getPositionInArea (DArea area, byte offsetOrd, byte candidate) : byte
  Egy jelölt offsetOrd-adik előfordulásának pozícióját adja vissza az adott területen.

- isTableResolved (byte where) : boolean
  Visszaadja, hogy a tábla megvan-e fejtve.

- isTableCorrect (byte where) : boolean
  Visszaadja, hogy a tábla láthatólag helyes-e.

- resetTable (byte where, boolean stateCandidates)

  Reszeteli a képernyő illetve a memóriatáblát.

- changeCell (DCoord coord)

  Cellaváltásnál beállítja az előző lépés cellapozícióját, majd vált az új cellára. Cellaváltás utáni folyamatoknak, és a toolbar gombok engedélyezésének vezérlése.

- setCellBackground (DCoord coord, Color color)

  Az adott cella színét állítja be az adott színre.

- setCandidateMark (DState state)

  A képernyőn beállítja az adott lehetőség-érték kijelöltségi állapotát.

- setCandidateState (DState state)

  Be/kikapcsolja az adott jelöltet.

- setCell (DWhere where, String values)

  Egy cella értékeinek beállítása.

- setCellMark (DCoord coord, boolean markType, boolean state)

  Az adott cella jelölését kapcsolja ki/be.

- showTable (byte toWhere)

  A memóriából áttölti a képernyőre a táblát, vagy viszont. A toWhere érték a cél jelölésére szolgál.

- fillIndexTable ()

  Indextábla feltöltése.

- setGridResult (DNumber number)

  Beállítja az adott cella megoldás-értékét.

- setListResult (short stepOrder, byte rule, DNumber number)
  Az eredményértéket hozzáadja a result listához.

- getActCell() : DCoord
  Visszaadja az aktuális cella koordinátáját.

- getPrevCell() : DCoord
  Visszaadja az előző cella koordinátáját.

- setActCell (DCoord coord)
  Beállítja az aktuális cella koordinátáját.

- setPrevCell (DCoord coord)
  Beállítja az előző cella koordinátáját.

- getEmptyCellsCounter (byte where) : short
  Visszaadja a memória/képernyő-rács üres celláinak a számát.

- getEmptyCellsCounterString (byte where) : String
  Visszaadja a memória/képernyő-rács üres celláinak a számát.

- setEmptyCellsCounter (byte where, short counter)
  Beállítja a memória/képernyő-rács üres celláinak a számát.

- setEmptyCellsCounterByAdding (byte where, short adding)
  Növeli / csökkenti a memória/képernyő-rács üres celláinak a számát.

- isEditable () : boolean
  Szerkesztő üzemmód állapotának lekérdezése.

- setEditable (boolean state)
  A szerkesztőmód be/kikapcsolása.

Frame osztály

Ide helyeztem el az ablak felső és alsó részét kezelő metódusokat és az azok állapotát tároló adattagokat.

Adattagok:

- String tableFileName : Az aktuálisan betöltött file nevét tárolja.

Metódusok:

- Frame ()
  Meghívom az init metódust.

- init ()
  A fő ablak tartalmát feltöltöm a különböző elemekkel. A jobb olvashatóság érdekében a részeket külön metódusok hozzák létre, amiket szintén itt hívok meg. Végül beállítom a szerkesztési üzemmódot bekapcsolt állapotra.

- myMenu ()
  Felépíti a menüt, azaz létrehozza a menüpontokat, majd hozzájuk rendeli a forró billentyűket. Végül pedig összekapcsolja a menüpontokat az azokat lekezelő eseménykezelővel.

- myToolbar ()
  Felépíti a toolbart, azaz beállítja a nyomógombok paneljének tulajdonságait, majd felrakom a panelre a gombokat, és összekapcsolom az eseménykezelővel.

- addJButton (JButton comp, String text, String comm, String toolTipText)
  A toolbar-hoz hozzáadja az adott gombot, és beállítja a paramétereit.

- myFoot ()
  A lábléc elemeit állítja össze.

Most következzenek a menü és lábléc kezelő metódusok. A toolbar-t is voltaképp egyfajta menüként tekintem, mivel az is a program funkcióinak elérését valósítja meg.

- actionPerformed (ActionEvent ae)
  Ez a metódus kezeli le a menü és a toolbar egyes elemeinek kiválasztását. Az egyes elemek álnevei alapján el tudom dönteni, mit választott ki a felhasználó.

- showResults ()
  Megjeleníti a bemutatóhoz tartozó eredménylistát.

- loadTable ()
  Megjeleníti a fájl kiválasztó dialógus ablakot, majd betölti a kiválasztott fájlt.

- convertRowToCells (String fRow)
  Egy sort tartalmazó stringet cellákra bontja.

- saveTable ()
  Megjeleníti a fájl kiválasztó dialógus ablakot, majd az adott néven elmenti a tábla tartalmát.

- setToolbarButtonEnabled (byte ord, boolean state)
  A toolbar egy-egy nyomógombjának engedélyezési státusát állítja be.

- setupToolbarNumButtons (String candidates)
  Az adott karaktersor alapján beállítja a nyomógombok engedélyezettségi állapotát.

- showFileName ()
  Beállítja a megjelenő file nevét a kezelőfelületen.

- getTableFileName () : String
  Visszaadja a mentett tábla file nevét.

- setTableFileName (String fileName)
  Beállítja a mentett tábla file nevét.

- showHitsToolTipText (byte rule)
  A talált lépés mezőjének tooltip szövegét állítja be.

- showEmptyCells (byte where, String number)
  Megjeleníti az üres cellák számát. A where szabályozza, hogy az ablak állapotsorának erre a célra létrehozott mezőjébe (SCREEN), vagy a konzol képernyőre (MEMORY) helyezze e el az eredményt.

- showResultText (byte where, String text)
  Megjeleníti az eredmény szövegét a showEmptyCells-nél leírtak alapján.

- setupFrame ()
  Frissíti a kezelőfelületet.

Rules osztály

Ez az osztály tartalmazza a program által használt szabályokat összevont formában.

Metódusok:

- Rules (Frame parent)
  Az osztály konstruktora, mely a Frame osztály elérhetőségét biztosítja.

- nakedSingle (byte pWhere) : boolean
  A Naked Single nevű szabály adaptációja, mely soronként, azon belül oszloponként végignézi az egyes cellákat olyan cella után kutatva, melyben csak egy érték van, majd ezt beállítja eredményértékül.

- naked (byte pWhere, byte ruleOrder) : boolean
  Az összes további Naked szabály adaptációja, ideértve azt is, amit nem tartalmaz e diplomamunka, gondolok itt a tiszta 5-ösre.

- hidden (byte pWhere, byte ruleOrder) : boolean

  Az összes Hidden szabály adaptációja, ideértve azt is, amit nem tartalmaz e diplomamunka, gondolok itt a rejtett 5-ösre.

- intersection (byte pWhere) : boolean

  Az Intersection nevű szabály adaptációja, mely sorok/oszlopok és blokkok kapcsolatán alapuló szabályrendszer.

- xWings (byte pWhere) : boolean

  Az X-Wings nevű szabály adaptációja.

- getExtent (byte ruleOrder) : byte

  Visszaadja, hogy az adott szabály esetében hány értékig keressen.

Show osztály

Attribútumai:

- showStarted : Azt tárolja, hogy zajlik e a bemutató
- actRulesChain : Pillanatnyi szabálybejárási lánc
- baseRulesChain : Alap szabálybejárási lánc

Metódusai:

- Show (Frame parent)

  Az osztály konstruktora, mely a Frame osztály elérhetőségét biztosítja.

- assistsSetShow (boolean state)

  Aktuális lépéshez tartozó bejelölendő cellák jelöltségi állapotának beállítása.

- resultsSetShow (boolean res, boolean markThem, boolean doIt)

  A resultList aktuális lépésének tartalmát jeleníti meg a táblán.

Paraméterei:

- state: A fellelt találatok értékeit be vagy ki kapcsolja e.
- markThem: A fellelt találatok értékeit jelölje e be.
- doIt: Végezze e el a talált lépéshez tartozó értékek kapcsolgatását.

- stepBack ()

Lépés visszafelé.

- stepForward (byte where)

Lépés előre.

- stepForwardScreen (byte where)

Lépés előre a képernyőn.

- stepForwardMemory (byte where)

Lépés előre a memóriában lévő eredmény-listában.

- createList ()

Felépíti az eredmény-listát.

- fixIt ()

A listakészítés közbeni találat végrehajtását végzi el, hogy a következő lépés onnan induljon.

- oneStep (byte where, byte stepOrder)

Meghívogatja a kereső metódusokat sorrendben addig, míg nem talál megoldást. Ha a 10. algoritmus sem talál megoldást, akkor a keresés sikertelen megfejtés üzenettel leáll.

Ám ez a metódus tartalmaz még pár találati-lista szervezéséhez kapcsolódó folyamatot. Gondolok itt a Beállítások dialógusablak Használandó szabályok fülének jobb oldalán lévő elemekre, és aminek a leírását a StorageProps osztály leírása tartalmazza.

- setStepPosition ()

  Beállítja az aktuális lépés pozícióját a lépéskezdeteket tartalmazó tömbben.

- rule_Cleaning (short stepOrder, DNumber number) : boolean

  Ez voltaképp a 0. szabály, ami azon alapszik, hogy ha van egy fix értékünk, akkor az ehhez kapcsolható cellákból ez az érték törölhető a lehetőség-értékek közül.

- turnOffCandidates (short stepOrder, DNumber number)

  Az adott jelölt-érték törlése az adott cellához kapcsolódó munkaterületekről.

- isShowStarted () : boolean

  Azt adja vissza, hogy tart e a bemutató.

- setShowStarted (boolean state)

  A bemutató folyamatának létét állítja be.

- getRuleOrder (byte index) : byte

  Visszaadja az index érték szerinti szabály sorszámát.

- setupRulesChain () : String

  Összeállítja a lépések listáját, majd átadja a baseRulesChain változónak.

- getBaseRulesChain () : String

  Visszaadja az alap szabályláncot.

- setBaseRulesChain (String chain)

  Beállítja az alap szabályláncot.

- getActRulesChain () : String

  Visszaadja az aktuális szabályláncot.

- changeChain (short order)

  Az adott pozíció előtti részt áthelyezi a lánc végére.

Props osztály

Ez az osztály tartalmazza a program által használandó állapotértékek tárolását és kezelését. Az osztályban találhatók a használandó szabályokra vonatkozó tárolók (actRulesChain, baseRulesChain).

Ezen kívül az osztály végzi el a Beállítások ablak és annak eredményeihez tartozó folyamatok kezelését is. Itt a folyamat úgy zajlik, hogy az objektum létrehozásakor beolvasásra kerül a fileIni-ben tárolt fájlnevű ini, a beállításokat tároló állomány, valamint a dialógusablak a tárolt értékekhez igazítva.

A dialógusablak első lapján található a program szabályrendszerének vezérlésére vonatkozó beállítások, amely így néz ki:

21. ábra: Szabályrendszer beállításainak ablaka (forrás: Saját készítés)

Itt az látszik, hogy a bal oldalon kapcsolgatni tudjuk, hogy az alkalmazás mely szabályokat használja fel a kereséshez. A Naked Single szabály el van szürkítve, hogy ne lehessen kikapcsolni, mivel erre a szabályra minden kereséskor szükség van, mert csak ez a szabály hoz végső eredményt egy cellára vonatkoztatva.

A jobb oldalon a keresés folyamatára vonatkozó beállítások találhatóak.

- Szabály kikerülés: Kikapcsolt állapotban minden alkalmazandó szabályon megtörténik a keresés, majd ha nincs eredmény, akkor leáll a keresés. Bekapcsolt állapotnál viszont ha első körben nem talál megoldást, akkor elkezdi újból a keresést úgy, hogy figyelmen kívül hagy egy-egy szabályt.
- Szabály betokozás: Kikapcsolt állapot esetén nem állítja át a keresési listát. Bekapcsolt állapotban viszont az történik, hogyha a keresési folyamat során valamely szabály esetén megoldás van, akkor a következő ciklusban a keresési lánc megváltozik úgy, hogy a talált pozíció a Naked Single helyéhez képest az

első illetve a második helyre kerül. Ezáltal a keresés ilyenkor arról a pozícióról folytatódik, ahol az előző körben maradt.

- Naked Single helye: Itt választani lehet, hogy a Naked Single szabály mindig az első helyen legyen e, vagy a listával együtt vándoroljon.
- Haladási irány: Itt beállítható, hogy a keresés milyen irányba menjen végbe. Ezt kissé torzítja az előző pontban ismertetett Naked Single pozíciójára vonatkozó opció.

Attribútumai:

- actRulesChain: A megoldás-keresés pillanatnyi listájának tárolója.
- baseRulesChain: A megoldás-keresés eredeti listájának tárolója.
- prop: A módosítható értékek tárolója
- prevProp: Az előzőleg elmentett értékek tárolója.
- defaultProp: Az alapértelmezett értékek tárolója
- fileIni: Az ini file állománypointere

Metódusai:

- StorageProps (Frame parent, boolean modal)
  Konstruktor.

- init ()
  Beállítja az osztály kezdőértékeit.

- setupDefaultProperties ()
  Meghatározza az alapértékeket.

- showPropsFrame ()
  Megjeleníti a beállításokat végző ablakot.

- settingsGUI ()
  A módosítható értékek beállítását végző ablak kialakítását végző metódus.

- pages ()
  Az ablak felső részének kialakítása.

- page1 ()

  Használandó szabályok lap kialakítása.

- page2 ()

  Színbeállító lap kialakítása.

- page3 ()

  Üzemmód kiválasztó lap kialakítása.

- addGuiElement (int page, DCoord coord, JComponent comp, String comm, boolean mustTurn)

  Elemet helyez el az adott lap adott pozícióján, hozzárendeli az eseménykezelőt, végül ha szükséges, hozzáadja a kikapcsolandó elemek listájához.

- addListener (AbstractButton ab, String comm, ActionListener e)

  Az adott gombhoz eseménykezelőt rendel.

- actionPerformed (ActionEvent ae)

  A beállítások dialógusablakának eseménykezelője.

- setupPages ()

  Beállítja, hogy mely lapok legyenek aktívak, ami attól függ, hogy a bemutató elkezdődött e már.

- resetDialog ()

  Beállítja a kezelőfelületen lévő elemek állapotát a prop tárolóban lévő értékek alapján.

- resetRadioButtons ()

  Beállítja a kezelőfelületen lévő Radio-gombok állapotát.

- activeRulesCount () : byte

  Összeszámlálja a használandó szabályokat.

- closeDialog ()

  "Bezárja" a beállításokat végző ablakot. A macskaköröm arra utal, hogy csak elrejti.

- saveProps ()

  Elmenti a beállított értékeket.

- getActualColor (byte what) : Color

  Az előre beállított színséma alapján visszaadja az adott elemhez tartozó színt.

- isColorSchemeChanged () : boolean

  Megvizsgálja, hogy a séma megváltozott e az előző mentési állapothoz képest.

- setColorSchemeToProp (String scheme)

  Az aktuális színséma sorszámát állítja be.

- getColorSchemeFromProp () : byte

  Az aktuális színséma sorszámát kérdezi le.

- getColorSchemeFromDialog () : byte

  Visszaadja a kiválasztott színséma sorszámát.

- getPrevColorScheme () : byte

  Visszaadja a mentett színséma sorszámát.

- getActualColors () : DColors

  Visszaadja az aktuális sémához tartozó szineket.

- showColors ()

  Megjeleníti az aktuális színeket, vagyis átszínezi a színsávokat.

- setOperatingModeToProp (byte opMode)

  Az aktuális üzemmód sorszámát menti el.

- getOperatingModeFromProp () : byte
  Visszaadja a műveletvégzés módját.

- getActOperatingModeFromDialog () : byte
  Visszaadja a kiválasztott üzemmódot.

- setRuleStateToProp (byte rule, boolean state)
  Az adott szabály állapotát állítja be a propertyben.

- getRuleStateFromProp (byte rule) : boolean
  Kiolvassa az adott szabály használhatósági állapotát a propertyből.

- getActRuleStateFromDialog (byte rule) : boolean
  Kiolvassa a kiválasztott szabály állapotát a dialógusablakból.

- isRuleUsable (byte rule) : boolean
  Visszaadja, hogy az adott szabály bekapcsolt állapotban van e.

- setShowStarted () boolean
  A bemutató folyamatának létét állítja be.

- getRulesGetOutFromProp () : boolean
  Szabály kikerülési kapcsolóhoz tartozó változó állapotának lekérdezése.

- setRulesGetOutToProp () : boolean
  Szabály kikerülési kapcsolóhoz tartozó változó állapotának beállítása.

- getRulesGetOutFromDialog () : boolean
  A szabály kikerülési kapcsoló állapotának kiolvasása a dialógus-ablakból.

- getEncapsulateFromProp () : boolean
  Szabály betokozási kapcsolóhoz tartozó változó állapotának lekérdezése.

- getEncapsulateToProp () : boolean

  Szabály betokozási kapcsolóhoz tartozó változó állapotának beállítása.

- getEncapsulateFromDialog () : boolean

  A szabály betokozási kapcsoló állapotának kiolvasása a dialógus-ablakból.

- getNakedSinglePosFromProp () : boolean

  A Naked Single szabály első pozíciójára vonatkozó kapcsolóhoz tartozó változó állapotának lekérdezése.

- getNakedSinglePosToProp () : boolean

  A Naked Single szabály első pozíciójára vonatkozó kapcsolóhoz tartozó változó állapotának beállítása.

- getNakedSinglePosFromDialog () : boolean

  A Naked Single szabály első pozíciójára vonatkozó kapcsoló állapotának kiolvasása a dialógus-ablakból.

- getSearchWayFromProp () : boolean

  Keresési irány kapcsoló állapotának lekérdezése.

- getSearchWayToProp () : boolean

  Keresési irány kapcsoló állapotának beállítása.

- getSearchWayFromDialog () : boolean

  Keresési irány kapcsoló állapotának kiolvasása a dialógus-ablakból.

- resetElements (boolean way)

  Beállítja a dialógus elemeinek engedélyezési státuszát.

- getExtention (boolean needDot) : String

  A fájlkiterjesztést adja vissza. A needDot kapcsoló mondja meg, hogy pontot is tegyen e a kiterjesztés elé.

- copyProperty (Properties sour, Properties dest)
  átmásolja a property bejegyzéseket a sour által megadott propertyből a dest által megadottba.

SudokuPI osztály

Ez az osztály a program belépési pontja. Csak pár elemet tartalmaz, úgy mint egy pSudoku nevű JPanelt, mely a teljes felület gyűjtő konténere, majd az elrendezés-kezelőjét beállítom borderLayout-ra. A tervezés korai szakaszában, e dokumentáció 3. fejezetének elején láthattuk a képernyőtervet. A terven az látszik, hogy 3 fő részből tevődik össze, fejrész (menü és eszközsor), lábléc, és játéktér. Ezeknek egy-egy panel felel meg, melynek elnevezése utal a tartalmára és az ablakban elfoglalt helyére. Ezek elhelyezését az alábbi sorok végzik el:

```
pSudoku.add(BorderLayout.NORTH, myGUI.pHead);
pSudoku.add(BorderLayout.SOUTH, myGUI.pFoot);
pSudoku.add(BorderLayout.CENTER, myGUI.pBody);.
```

Attribútumai:

- JPanel pSudoku : Egy pointer, mely a pSudoku panelre mutat annak érdekében, hogy a főfelületről megnyíló dialógus-ablakokat az ablak közepére helyezhessem. Valamint erre a panelre helyezem el a 3 fő ablakkonténert.

Metódusai:

- SudokuPI () : Paramétermentes konstruktor, mely az ablak 3 fő elemét elhelyezi az ablakban, úgymint pHead, pBody, pFoot.
- JPanel getpSudoku() : A főkonténerre mutató pointert adja vissza értékül.

Végül pedig a pSudoku panelt hozzáadjuk a JFrame-hez, ami az alkalmazás fő formja.
Az osztály tartalmazza még a main statikus metódust, melyben beállítom az ablak paramétereit, majd megjelenítem az ablakot.

5. Tesztelés

Az interneten rákeresve a szabály nevére találtam példákat. Ezek jó indulási alapok voltak, hogy leteszteljem a program helyes működését. Mivel több szabálynál több változat is előfordult, úgy mint sor – oszlop – blokk alapú, így mindegyikhez a megfelelő változatot bevittem a programba, hogy minden esetre felkészüljön a program. Az egyes feladványokat a Saves almappába mentettem a 3.4-es, Tárolási mechanizmus című fejezetben leírtak alapján. Ezek a példák fedik az összes algoritmus variánsokat, amik bekövetkezhetnek, mivel ezek segítségével szerettem volna a program minden ágát letesztelni, hogy ne érjen meglepetés se engem, se a tisztelt felhasználót.
Ezeket a példákat lefuttattam a különböző beállítások alkalmazásával, és nem meglepő módon, más és más eredményre jutott.
A legérdekesebb az volt, mikor a keresés sorrendjét megfordítottam és a Naked Single első pozíciójának fixálását kikapcsoltam. Ily módon a program teljesen fordított logikával, és háromszor annyi lépésben talált a megoldásra.
Ami még érdekes volt, hogy a különböző keresési opciók állítgatásával a keresési idő nem nagyban változott, bár ez csak szubjektív vélemény. Lehet, hogy érdemes lenne egy időzítő beépítése a programba, amely ezred másodperc, vagy még nagyobb pontossággal meghatározhatná a megfejtéshez szükséges időt.
A fent említett példákon lefuttatva az alkalmazás kereső funkcióját a programról bátran merem állítani, hogy helyesen működik.

6. Továbbfejlesztési javaslatok

- a további 30-32 algoritmus kidolgozása. (Bár már a tiszta és rejtett 5-ös szabály kész)
- a táblaméret változtatása, tehát ne csak a 9x9-es tábla legyen az alap, hanem a 16x16-os, és 25x25-ös.
- ablakméret változtatásának lehetősége.
- saját mentési / betöltési dialógusablak megírása, mely magyar nyelvű és kissé lebutított.
- nyomtatási funkciók megírása.
- Feladvány generátor megalkotása.
- A mentett tábla hibakódolása. Ezzel védve a hibás beolvasástól, vagy a trolloktól a mentett adatokat.

- Egy teszter program vagy metódus megírása, mely betölti a mentett táblákat egyesével, majd megoldaná azokat, összekapcsolva a más pontoknál leírt lehetőségekkel.
- Statisztikai mérések gyűjtése annak méréséhez például, hogy melyik függvényt használja többször / ritkábban az alkalmazás. Ennek eredményét arra lehetne használni, hogy a keresési sorrendet erre lehetne optimalizálni.
- Időzítő beépítése a keresés méréséhez.

Ezeket persze én is el fogom még végezni a programon, ami majd ezáltal új verziószámot fog kapni. Biztos hogy még sok ötletem lesz, amik jönnek folyamatosan, hisz ezt a listát is úgy állítottam össze.

7. Összegzés

E szoftver elkészítése közben rengeteget tanultam a programozás és programtervezés gyakorlat közbeni rejtelmeiről.
Első fontos tanulság, hogy alaposan meg kell tervezni a készítendő programot. Az én esetemben először ki kellett volna dolgoznom a kereső-algoritmusokat, így hamarabb rájöttem volna arra, hogy ráhúzhatóak a sudoku szabályok 2-3 kategóriára.
Ezek után, aki programozásba fog először gondolja át a lépéseket, majd bontsa fel több területre, és a területeket dolgozza ki, papíron. Amíg ez nincs meg, egy sort se írjon meg a készítendő szoftverből.
A program tervezéséhez használható az SSADM, az UML, vagy egy már elavult régi módszertan, a Jackson módszer, hisz bár elavult, de a cél szentesíti az eszközt.
Aztán az se egy elhanyagolható szempont, hogy sokszor a legegyszerűbb megoldások a legjobbak, így kár túlbonyolítani a dolgokat. Bár az is igaz, hogy ezt se a végtelenségig csináljuk, hisz vannak helyzetek, mikor épp a komplexitásban vannak a szépségek. Ezért is készült el a naked és a hidden metódus, melyek több keresési algoritmus feladatát végzik el.

8. Irodalomjegyzék

[1] Wimmer Ágnes, Juhász Péter, Jeney Johanna - Hogyan írjunk...? 101 tanács (szak)dolgozatíróknak, Alinea kiadó, Budapest, 2009, ISBN 978-963-9659-40-7

[2] Ian Sommerville – Szoftverrendszerek fejlesztése, Panem könyvkiadó, Budapest, 2002, ISBN 963 545 311 6

[3] Robert C. Martin - Tiszta kód, az agilis szoftverfejlesztés kézikönyve, Kiskapu kiadó, Budapest, 2010, ISBN 978 963 9637 69 6

[4] Angster Erzsébet - Az objektumorientált tervezés és programozás alapjai, Magánkiadás, Budapest, 1999, ISBN 963 6508 18 6

[5] Kövesdán Gábor – Szoftverfejlesztés JAVA SE platformon, SZAK Kiadó, Bicske, 2014, ISBN 978-963-9863-35-4

[6] Kathy Sierra, Bert Bates – Agyhullám: Java Frissített kiadás: Java 5.0, kiadó: Kiskapu kft., Budapest, 2011, ISBN 978 963 9637 79 5

[7] http://www.sudokuwiki.org/