Mesterséges Intelligencia házi feladat

Betűket alkalmazó 9x9 sudoku variáns

Készítették:

Bodolai Dorottya és Mayer Emil

2014. ősz

A feladat leírása:

A sudoku játék 9*9-es táblát használó változata közismert, de található számos érdekes variáns is. Az adott feladatban betűkből álló sudokut kell megvalósítani (és lehetőség szerint értékelni a játékost a megoldás során). A betűkből felépített sudoku szabályai azonosak a számokból felépítettel, de a végén a főátlóban értelmes szó alakul ki. A pontosabb specifikációt az első konzultáción rögzítjük. Természetesen, ha valaki irodalomkutatással (net) talál jó megoldást, azt megérti és implementálja, az teljes értékű megoldás. (Nem kell újra feltalálni a spanyolviaszt...)

Az elkészített feladat felépítése:

A játék indulásakor a felhasználó a “New Game” gombra kattint. Ekkor a program új pályát generál.

A pálya készítése úgy működik, hogy meghívjuk a ToltPuzzle() függvényt, ami egy 80 elemű “int” típusú változókból álló tömb első kilenc elemét feltölti véletlenszerűen az 1..9 értékekkel. Az így kapott tömböt a átadjuk a makePuzzle() függvénynek, amely a továbbiakban úgy tölti ki a tomb soron következő elemeit, hogy mindig saját magát rekurzívan hívva a következő tömbelemre beírja egy 1-től 9-ig futó ciklusnak azon elemét, amely 1-től felfelé számolva először ad szintaktikailag helyesen kitöltött rejtvényt. (Ezt úgy csinálja, hogy minden egyes lépésben a checkConstraints() függvény segítségével leellenőrzi, hogy az aktuális tábla megfelel-e a Sudoku speciális szabályainak. (Esetünkben ez azt jelenti, hogy minden sorban, oszlopban, 3x3-as résztáblában, illetve a főátlóban is páronként különböző elemeknek kell szerepelniük.) Ha az összes elemet ily módon kitöltötte, akkor IGAZ értékkel tér visszaaz összes rekurzív hívás, így az eredeti függvényhívás is.

Érdekes részlet, hogy ez az algoritmus nem minden véletlenszerű 9-es számkombinációra ad kitölthető táblát, ezért a generálás felgyorsítása érdekében, ha a rekurzió mélysége meghaladja az 1000-et, akkor automatikusan HAMIS értékkel térünk vissza, és a generálás újraindul. Ez meglehetősen gyors lefutású táblagenerálást tesz lehetővé.

Eddig nem esett szó a betűkről, pedig azok szerves részét képezik a játék felépítésének. A szavakat amiket felhasználunk egy rövid szűrés után egy nyilvánosan elérhető szótár[1] szavai közül válogattuk ki, figyelve arra, hogy számunkra csak a kilenc betűből álló, ismétlődő betűket nem tartalmazó szavakra van szükségünk.

Ha kész a tábla, akkor a charpuzzle() függvény segítségével az elkészített rejtvényben adott helyen álló számokat megfeleltetjük a szólistából véletlenszerűen kiválasztott szó betűinek, így kész van az a tábla, amire a játékosnak jutnia kell majd a megoldása végén.

[1] http://extensions.openoffice.org/en/project/hungarian-dictionary-pack

A játékosnak a felhasználói felületen lehetősége van a rejtvény nehézségének beállítására. Ez azt jelenti, hogy egy 1 és 4 között változtatható érték segítségével meg tudja adni, hogy hány üres mezőt szeretne látni a táblában, ezáltal megszabja a rejtvény nehézségét.
Ez a nehézség azért fontos, mert a rejtvény egyértelmű megoldásához nélkülözhetetlen, hogy úgy takarjunk ki elemeket, hogy a kapott Sudokunak ne lehessen egynél több jó megoldása.

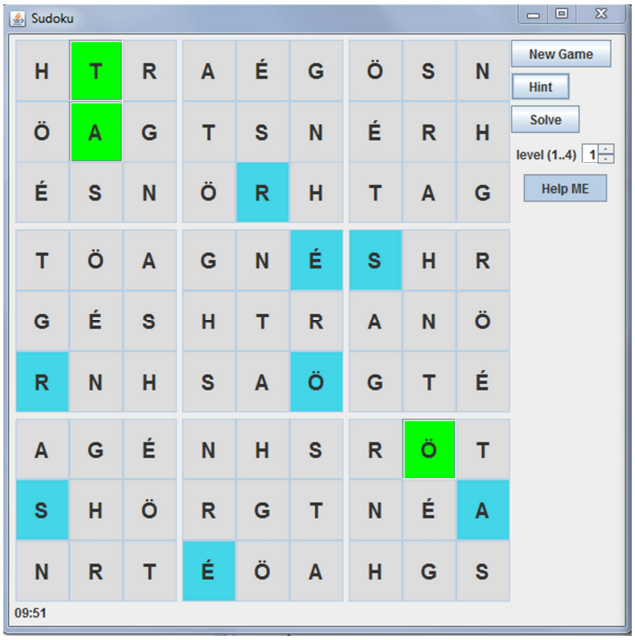
Ezt úgy érjük el, hogy a generált táblából a nehézségi szintnek megfelelően véletlenszerűen kitakarunk elemeket (egy új tömbbe másoljuk a generátumot, és a kitakart helyekre -1 értéket helyettesítünk), majd a kitakart táblát egy megoldófüggvény (solvePuzzle()) gondjaira bízzuk.
Itt jön a program gyenge pontja, ugyanis a megoldófüggvény úgy működik, hogy sorban megy végig a tömb elemein, ha üres mezőt talál, akkor megnézi, hogy mely elemek azok, amelyek elméletileg helyesek volnának még oda, az egyiket véletlenszerűen behelyettesíti, és továbblép a következő elemre (rekurzívan). Ez sajnos azt vonja maga után, hogy lehetséges próbálkozások száma igen meredeken növekszik a kitakart elemek számától függően. Körülbelül negyven kitakart mező esetén azonban még elviselhető ez a futási idő.

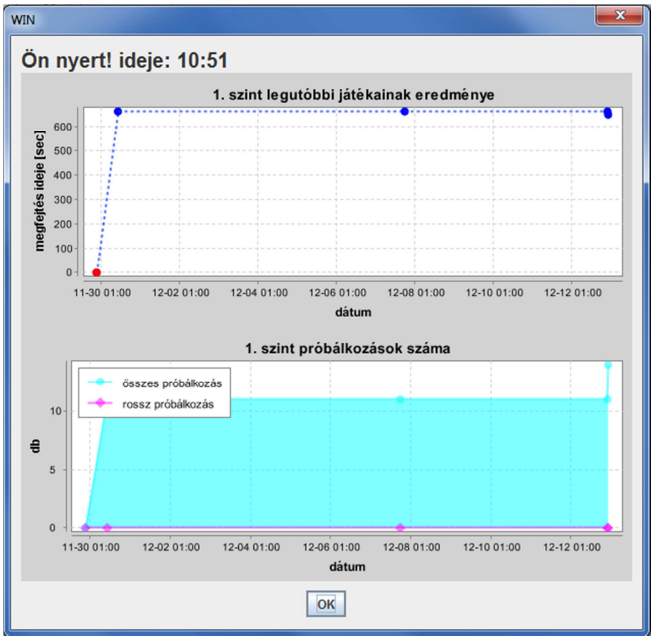
Az algoritmus azt mondja, hogy ráeresztjük a kitakart táblára a megoldófüggvényt valahány (esetünkben épp 30) egymás utáni alkalommal, amely minden iterációban igen sokszor lefut. (ciklus a ciklusban, megadjuk az esélyt a programnak, hogy találjon helyes megoldást). Minden iteráció végén összevetjük a kapott eredményt az eredetileg generált táblával, és ha egy kis eltérés is létezik, akkor abortálunk, és új kitakarást csinálunk.
A tesztek alapján a jelenleg megadott értékek biztosítják azt, hogy legmagasabb nehézségi szinten (4) se tartson számottevően sok ideig egy elfogadhatóan nagy valószínűséggel egyértelműen megoldható tábla generálása.

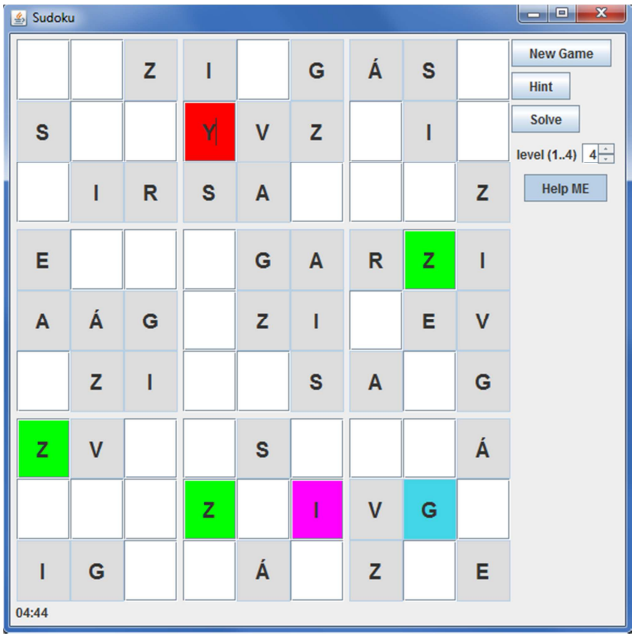
Ha kész a tábla, akkor már csak a felhasználón múlik a helyes megoldás megtalálása.
Választhatja a nehéz utat, vagyis addig próbálkozik a tábla kitöltésével, amíg az helyesnek nem bizonyul, azonban rendelkezésére állnak segítségek is. Fontos tudni, hogy a játék időre megy, a játék indulásakor elindul egy stopper, és ha segítséget veszünk igénybe, akkor időbüntetést kapunk.
A „Hint” gomb az egyik még kitöltetlen mező helyére beírja az oda való értéket. (60 másodperc időbüntetés)
A „Solve” gomb megnyomása a játék feladását, megoldás megmutatását eredményezi, azonban ilyenkor a végelszámolásban az időeredményünk egyáltalán nem számít.
A „Help ME” feliratú gomb pedig annyit tesz, hogy ilyenkor minden beírásról közli a program, hogy az oda illő értéket írtuk-e be a mezőbe. Ha igen, akkor zöldre színezi a hátteret, ha nem, pirosra. Vigyázat, ilyenkor az idő kétszer olyan gyorsan ketyeg!
Ezen kívül fontos tudni, hogy ha olyan hibát vétünk, amely egyszerűen elkerülhető lett volna, vagyis olyan értéket írunk be az egyik mezőbe, amely az aktuálisan látható táblában is ellene megy a szabályoknak, vagyis szerepel már az adott sorban/oszlopban/területen, akkor jutalmunk egy lilára színezett háttér, és 60 másodperc időbüntetés.

Ha megoldottuk a táblát, elérkeztünk az aktuális játék végéhez, akkor felugrik egy ablak, amelyben két grafikon[2] látható. Az alsó jelöli az elmúlt öt játék összes próbálkozásainak számát, illetve azt, hogy ezek közül hány volt rossz, a felső grafikonon pedig az elmúlt öt játék megoldásához szükséges idő látható másodpercben. Felül kék a pötty, ha szabályosan eljutottunk a játék végére, illetve piros, ha feladtuk, vagyis 0 érték szerepel a képen.

Ábrák:



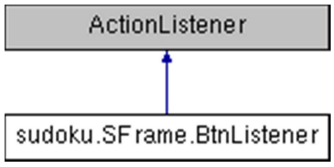




[2] http://xeiam.com/xchart/

SFrame.BtnListener osztály:

Inheritance diagram for sudoku.SFrame.BtnListener:



Public Member Functions

- void **actionPerformed** (ActionEvent e)

Detailed Description

A gombok (New Game, Hint, Solve) esemeny kezeloje.

Member Function Documentation

void sudoku.SFrame.BtnListener.actionPerformed (ActionEvent *e*)

New Game - Visszaallitja a tablat, beallitja a nehezseget generaltat egy uj tabla es elinditja a stoppert
Hint - Ha meg nem ert veget a jatek, akkor kitolt egy mezot Solve - Kitolti a tablat es az idot 0-ra allitja

---

Controller osztály:

Public Member Functions

- **Controller** ()
- void **initCharset** ()
- void **makeTabla** ()
- void **kitakarTabla** ()
- boolean **solveTabla** ()
- void **setNehezseg** (int **nehezseg**)
- int **getNehezseg** ()
- **Tabla getT** ()
- void **resetTabla** ()
- String **printINT** (int[] **t**)

Private Attributes

- **Tabla t**
- int **nehezseg** = 1

Detailed Description

Controller osztaly, a jatektabla kezelesere

Constructor & Destructor Documentation

sudoku.Controller.Controller ()

Konstruktor, letrehozza a tablat

Member Function Documentation

int sudoku.Controller.getNehezseg ()

Visszaadja a nehezseget

Tabla sudoku.Controller.getT ()

Visszaadja magat a tablat

void sudoku.Controller.initCharset ()

A szolistabol random valaszt egy sort es ennek a karaktereibol allitja elo a tabla karakterkeszletet

void sudoku.Controller.kitakarTabla ()

Nehezsegnek megfelelo szamu mezot kitakar. Valaszt egy random sor-oszlop parost, ha az erteke mar 0, akkor addig uj paros valaszt, kinullazza az erteket es a kitakart mezok szamat eggyel noveli.

void sudoku.Controller.makeTabla ()

Tablat letrehozo fuggveny. Feltolti a tabla elso sorat veletlenszeruen, majd feltolti a tobbit is ennek fuggvenyeben, amig nem kap szabalyos tablat, addig ismetli ezeket a lepeseket. A kesz tabla elemeit elmenti a **Tabla** nyolcvanegy tombjebe.

String sudoku.Controller.printINT (int[] *t*)

A parameterben kapott int tombot visszaadja emesztheto formatumban

Parameters:

t	stringge alakitando tomb

Returns:

a tomb emesztheto formatumban

void sudoku.Controller.resetTabla ()

Visszaallitja a tablat az alapallapotaba

void sudoku.Controller.setNehezseg (int *nehezseg*)

Beallitja a nehezseget

Parameters:

nehezseg	a beallitando nehezseg

boolean sudoku.Controller.solveTabla ()

Megoldo fuggveny. Betolti a generalt tablat es kitakartatja az elemeket. Megoldja ezt, ha nem jar sikerrel, akkor uj kitakarast alkalmaz.

Returns:

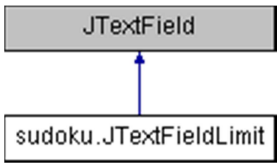
Member Data Documentation

int sudoku.Controller.nehezseg = 1`[private]`

Tabla sudoku.Controller.t`[private]`

JTextFieldLimit osztály:

Inheritance diagram for sudoku.JTextFieldLimit:



Classes

- class **LimitDocument**

Public Member Functions

- **JTextFieldLimit** (int **limit**)

Protected Member Functions

- Document **createDefaultModel** ()

Private Attributes

- int **limit**
  A megadott limit.

Static Private Attributes

- static final long **serialVersionUID** = -6538640453215003146L

Detailed Description

Sajat textField olsztaly, amelyben csak limitalt szamu karaktert lehet beirni

---

Constructor & Destructor Documentation

sudoku.JTextFieldLimit.JTextFieldLimit (int *limit*)

Konstruktor

Parameters:

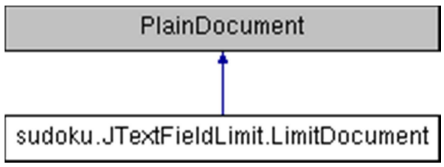
limit	az elvart limit

Member Function Documentation

Document sudoku.JTextFieldLimit.createDefaultModel ()[protected]

JTextFieldLimit.LimitDocument osztály:

Inheritance diagram for sudoku.JTextFieldLimit.LimitDocument:



Public Member Functions

- void **insertString** (int offset, String str, AttributeSet attr) throws BadLocationException

Static Private Attributes

- static final long **serialVersionUID** = 1361183952901627398L

Member Function Documentation

void sudoku.JTextFieldLimit.LimitDocument.insertString (int *offset*, String *str*, AttributeSet *attr*) throws BadLocationException

---

Member Data Documentation

final long sudoku.JTextFieldLimit.LimitDocument.serialVersionUID = 1361183952901627398L`[static]`, `[private]`

Main osztály:

Static Public Member Functions

- static void **main** (String[] args)

Detailed Description

Fo osztaly

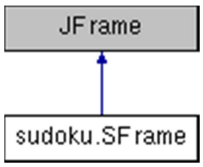
Member Function Documentation

static void sudoku.Main.main (String[] *args*) `[static]`

Letrehoz egy Controllert es egy SFrame-et

SFrame osztály:

Inheritance diagram for sudoku.SFrame:



Classes

- class **BtnListener**
- class **TextListener**
- class **TimeListener**

Public Member Functions

- **SFrame** (**Controller** c)
- void **setBoard** ()
- void **solveBoard** ()
- void **hintBoard** ()
- void **solved** ()
- JPanel **drawChart** (File **f**)

Private Attributes

- JPanel **contentPane**
- **JTextFieldLimit f** [][] = new **JTextFieldLimit**[9][9]
  - a 81 mezo
- JPanel **p** [][] = new JPanel[3][3]
  - 9 panel a 3x3-as tabla reszere
- **Controller controller**
  - a tablakat kezelo objektum
- boolean **helpmode** = false
  - a segito mod kapcsoloja
- JLabel **lblTime**
  - a stopper cimkeje
- Timer **timer**
  - az eltelt ido kezelesehez
- int **time**
  - az eltelt ido
- boolean **ended** = true
  - a jatek veget jezo flag
- int **nehezseg** = 1
  - a kovetjezo jatek nehezsegi szintje
- int **rosszTipp** = 0
  - rosszul beirt karakterek szama
- int **tipp** = 0
  - az osszes beirt karakter

Static Private Attributes

- static final long **serialVersionUID** = 650261138298404165L

Constructor & Destructor Documentation

sudoku.SFrame.SFrame (Controller *c*)

Az ablak, mely tartalmazza a jatektablat

Parameters:

c	a jatek soran hasznalt tablat kezelo controller

Member Function Documentation

JPanel sudoku.SFrame.drawChart (File *f*)

A parameterkent kapott filet beolvassa, az utolso 5 adatbol diagramot general. Az elso diagram a datumok es az adott szint megoldasi idejei (masodpercben) alapjan keszul, a solve gomb segitsegevel megoldottak 0 értékkel és piros jelolovel kerulnek kirajzolasra. A masodik diagram a datumokat a probalkozasok szamat abrazolja.

Parameters:

f	az adatokat tartalmazo fajl

Returns:

a kirajzolando diagramokat tartalmazo JPanel

void sudoku.SFrame.hintBoard ()

hint gomb megnyomasa utan egy veletlenszeru kitoltetlen mezot kitolt

void sudoku.SFrame.setBoard ()

uj jatek eseten a tabla elemeit beallitja a kitakarasnak megfeleloen

void sudoku.SFrame.solveBoard ()

solve gomb megnyomasa utan a tabla meg kitoltetlen elemeit kitolti

void sudoku.SFrame.solved ()

ha a jatek vegetert, minden mezot kitoltottunk, a scores fajlba beleirja az aktualis eredmeny es a generalt diagramokatkirajzolja egy uj ablakba (JOptionPane)

Tabla osztaly:

Public Member Functions

- **Tabla** ()
- void **save81** ()
- void **load81** ()
- boolean **makePuzzle** (int[] **puzzle**, int i)
- boolean **solvePuzzle** (int[] **puzzle**, int i)
- boolean **checkSquare** (int[] **puzzle**, int i)
- boolean **checkConstraints** (int[] **puzzle**)
- void **charpuzzle** ()
- char[] **charpuzzle** (int[] tomb)
- char **getChar** (int c)
- int **getINT** (char c)
- String **toString** ()
- boolean **checkBlack** (int[] **puzzle**, int i, int kapott)
- void **ToltPuzzle** (int[] **puzzle**)
- void **reset** ()

Public Attributes

- int[] **puzzle**
- int[] **kitakart**
- int[] **nyolcvanegy**
- int **szamlalo** = 0
- int **szam** = 0
- int **szam_2** = 0
- char[] **chpuzzle**
- char[] **konvert**
- int **kitakartNum** =0

Static Public Attributes

- static char[] **karakterkeszlet**

Constructor & Destructor Documentation

sudoku.Tabla.Tabla ()

Member Function Documentation

void sudoku.Tabla.charpuzzle ()

Lemasolja a lemasolni valo puzzle tablat olyanra, hogy az char elemekbol alljon, igy a jatekfeluletre kiirhato legyen

char [] sudoku.Tabla.charpuzzle (int[] *tomb*)

Ugyanazt csinalja, mint a **charpuzzle()**, csak kapott tomb alapjan visszateresi ertekkel

boolean sudoku.Tabla.checkBlack (int[] *puzzle*, int *i*, int *kapott*)

Megnezi, hogy a kapott mezobe a kapott ertek helyesen beirhato-e.

A "sor" nevu valtozo az aktualis mezo 9-cel valo egesz osztasanak erteke.

Az "oszlop" nevu valtozo az aktualis mezo 9-es maradeka.

Kivesszuk az 1,2...9 ertekekkel feltoltott tombbol azokat az elemeket, amik szerepelnek az aktualis mezo soraban.

Kivesszuk az 1,2...9 ertekekkel feltoltott tombbol azokat az elemeket, amik szerepelnek az aktualis mezo oszlopaban.

Ez a switch-case szerkezet pedig az aktualis mezohoz tartozo 3x3-as teruletet vizsgalja a tablaban. Szinten kiveszi a "kilenc" tombbol azokat az elemeket, amik szerepelnek a 3x3-as resztablaban. Ezt a reszt lehetne szebben is csinalni, azonban azt hiszem, hogy igy sokkal konnyebben atlathato.

A kovetkezo resz pedig megvizsgalja, hogy maradt-e olyan eleme a kilences tombnek ami egyenlo a kapott ertekkel.

Ha pedig a kapott (user altal beirt) ertek nincs benne a lehetseges ertekek tombjeben, akkor false lesz a valasz.

boolean sudoku.Tabla.checkConstraints (int[] *puzzle*)

Leellenorzi, hogy a kapott tabla a szabalyoknak megfelelo-e.

A mi jatekunk sajatossaga, hogy raadasul a foatloban is egyedi elemeknek kell szerepelniuk, ha ez nem teljesul, akkor mar lephetunk is vissza.

A kovetkezo ket ciklus leellenorzi, hogy az oszlopokban es a sorokban egyedi ertekek kerultek-e.

Az alabbi igen bonyolultnak tuno switch-case szerkezet csupan megvizsgalja a tabla egyes 3x3-as komponenseire, hogy azok is szabalyosak-e. Lehetne rovidebben is, azonban igy szepen attekintheto.

boolean sudoku.Tabla.checkSquare (int[] *puzzle*, int *i*)

Feltolti az aktualis mezot egy lehetseges ertekkel.

A "sor" nevu valtozo az aktualis mezo 9-cel valo egesz osztasanak erteke.

Az "oszlop" nevu valtozo az aktualis mezo 9-es maradeka.

Kivesszuk az 1,2...9 ertekekkel feltoltott tombbol azokat az elemeket, amik szerepelnek az aktualis mezo soraban.

Kivesszuk az 1,2...9 ertekekkel feltoltott tombbol azokat az elemeket, amik szerepelnek az aktualis mezo oszlopaban.

Ez a switch-case szerkezet pedig az aktualis mezohoz tartozo 3x3-as teruletet vizsgalja a tablaban. Szinten kiveszi a "kilenc" tombbol azokat az elemeket, amik szerepelnek a 3x3-as resztablaban. Ezt a reszt lehetne szebben is csinalni, azonban azt hiszem, hogy igy sokkal konnyebben atlathato.

Ez a resz pedig megvizsgalja, hogy maradt-e olyan eleme a kilences tombnek amit meg be lehet irni.

Ha nem maradt a vizsgalt tombben ertek, akkor HAMIS ertekkel terunk vissza.

Valasztunk egy veletlenszeru erteket 1..9 intervallumon.

Ha ott -1 van, akkor ujat valasztunk.

A vizsgalt helyre beirjuk a leheteseges ertekek kozul veletlenszeruen valasztott erteket.

Es IGAZ ertekkel visszaterunk.

char sudoku.Tabla.getChar (int *c*)

Visszaadja a kapott szamhoz tartozo karaktert.

int sudoku.Tabla.getINT (char *c*)

Visszaadja a kapott karakterhez tartozo szamot, ha nincs, akkor -1-et.

void sudoku.Tabla.load81 ()

Lemasolja az egyik tombot, hogy ujra az eredeti allas keruljon a puzzle tombbe

boolean sudoku.Tabla.makePuzzle (int[] *puzzle*, int *i*)

Letrehoz egy szabalyos tablat szamokkal rekurziv modon.
Ha tobb mint 1000-szer hivtuk a rekurziot, akkor kilepunk, es ujrahivjuk, hogy gyorsitsuk a generalast.
Eloszor belepunk egy ciklusba mely majd a feltoltesben segedkezik.
Ha a kapott indexu elem nem nulla, akkor tovabblepunk a kovetkezo elemre.
Ha nulla az i. helyen levo elem, akkor ertekul adjuk neki az aktualis elemet a ciklusnak.
Megnezzuk, hogy az adott helyre illesztett elem szabalyos-e ott.
Ha igen, es az index 79-nel nagyobb, akkor keszen vagyunk.
Szabadon legeneralhatjuk a betuket tartalmazo tombot.
Es visszaterunk IGAZ ertekkel.
Ha meg nem ertuk el a 80. indexet, akkor tovabblepunk a kovetkezo elemre.
Ha pedig az ellenorzes soran kiderult, hogy a beirt ertek nem jo, kitoroljuk es probaljuk a kovetkezot.

void sudoku.Tabla.reset ()

Visszaallitja az osszes valtozot alapallapotba

void sudoku.Tabla.save81 ()

Elmenti a legeneralt puzzle tomb allasat egy masik tombbe

boolean sudoku.Tabla.solvePuzzle (int[] *puzzle*, int *i*)

Megprobal megoldani egy szabalyos tablat kitakart elemekkel.
Ha szaz iteracional tartunk mar, akkor ujrafuttatjuk az egeszet az eredeti ertekekkel.
Megnezzuk, hogy vegeztunk-e.
Ha igen, akkor kiirjuk es visszaterunk es atalakitjuk a tombot.
Ha az adott elem nem nulla.
Megvizsgaljuk, hogy van-e meg hely a tombben, ha igen akkor meghivjuk a kovetkezo elemre, mert nincs dolgunk az aktualis elemmel, de mehetunk meg tovabb a tablaban.

Ha a vegere ertunk, akkor meghivjuk elolrol a megoldo fuggvenyt.

Behelyettesitjuk az aktualis helyre az egyik lehetseges elemet.

Megvizsgaljuk, hogy van-e meg hely a tombben, ha igen akkor meghivjuk a kovetkezo elemre, mert nincs dolgunk az aktualis elemmel, de mehetunk meg tovabb a tablaban.

Ha a vegere ertunk, akkor meghivjuk elolrol a megoldo fuggvenyt.

void sudoku.Tabla.ToltPuzzle (int[] *puzzle*)

A **makePuzzle()** szamara veletlenszeruen kitolti egy int tomb elso 9 elemet 1..9 egyedi elemekkel

String sudoku.Tabla.toString ()

A 81 elemu tomboket emesztheto formaban adja vissza (formazott String)

Member Data Documentation

char [] sudoku.Tabla.chpuzzle

char [] sudoku.Tabla.karakterkeszlet`[static]`

int [] sudoku.Tabla.kitakart

int sudoku.Tabla.kitakartNum =0

char [] sudoku.Tabla.konvert

int [] sudoku.Tabla.nyolcvanegy

int [] sudoku.Tabla.puzzle

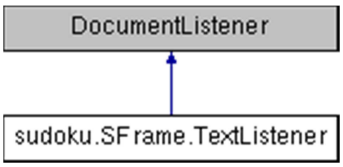
int sudoku.Tabla.szam = 0

int sudoku.Tabla.szam_2 = 0

int sudoku.Tabla.szamlalo = 0

SFrame.TextListener osztály:

Inheritance diagram for sudoku.SFrame.TextListener:



Public Member Functions

- void **changedUpdate** (DocumentEvent arg0)
- void **insertUpdate** (DocumentEvent arg0)
- void **removeUpdate** (DocumentEvent arg0)

Detailed Description

A mezok esemenykezeloje.

Member Function Documentation

void sudoku.SFrame.TextListener.changedUpdate (DocumentEvent *arg0*)

void sudoku.SFrame.TextListener.insertUpdate (DocumentEvent *arg0*)

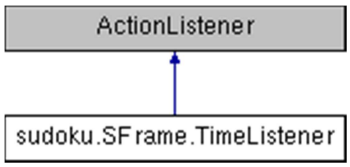
Amikor uj ertek kerul beirasra valamelyik mezobe, akkor megkeresi, hogy melyik mezorol van szo, amennyiben ez a mezo nem szerkesztheto, ugy nem foglalkozik vele. A szerkesztheto mezok eseten megnezi, hogy a beirt karakter megfelelo-e. Ha megfelelo, akkor a tabla kitakartNum szamlalojat csokkenti es segito modban a hatteret zoldre cserele. Ha nem megfelelo a karakter, akkor piros lesz a mezo es nem valtozik a kitakart mezok szama. Amennyiben a kitart mezok szam eleri a 0-t, ugy meghivodi a **solved()** fuggveny.

void sudoku.SFrame.TextListener.removeUpdate (DocumentEvent *arg0*)

Karakter kitorlese eseten, hasonloan az elozo fuggvenyhez megkeresi a mezot. Ha az ertek elozoleg jo volt, akkor a kitart mezok szamat noveli, ha nem volt jo, akkor valtozatlan marad. A mezo erteket kinullazza. Segito modban visszaallitja a hatterszint.

SFrame.TimeListener osztály:

Inheritance diagram for sudoku.SFrame.TimeListener:



Public Member Functions

- void **actionPerformed** (ActionEvent e)

Detailed Description

Az ido esemenykezeloje Masodpercenkent eggyel noveli a time valtozo erteket (segito modban kettovel) es kiirja a kepernyore a megfelelo cimkere

---

Member Function Documentation

void sudoku.SFrame.TimeListener.actionPerformed (ActionEvent *e*)