Tartalomjegyzék

Bevezetés

Választásom a Sudoku, mert már régóta ismerem és mindig érdekelt milyen módok léteznek, mind megoldási, mind generálási oldalon. Először szeretném bemutatni magát a játékot és a játék történetét.

A Sudoku kiváló agytorna és egyben jó kikapcsolódás bárki számára. Csak türelemre és logikus gondolkodásra van szükség. A siker egyik kulcsa, hogy a játékszabályok nagyon egyszerűek. Mégsem lehet megunni, mert számtalan variációs lehetőség létezik és a különböző nehézségi fokozatú táblák kellemes időtöltést biztosítanak kezdőknek és profiknak egyaránt. Néhány szám előre be van írva, de a rejtvény nehézsége elsősorban nem a megadott számok mennyiségétől függ, hanem attól, hogy logikailag mennyire nehéz meghatározni a további számokat. Vannak rejtvények, amelyekben csak kevés számjegy adott előre, ezek lehetnek akár könnyen fejthetők is, ugyanakkor lehet, hogy azokat a rejtvényeket, amelyekben sok szám adott, rendkívül nehéz megfejteni.

Minden jó feladvány megfejthető következtetéssel, találgatás nélkül. Azt még nem tudjuk, hogy minimum hány négyzetnek kell kitöltve lenni induláskor ahhoz, hogy a rejtvény korrekt módon megfejthető legyen, de olyan rejtvényt még senki nem talált, amiben 17-nél kevesebb induló mezővel ez megoldható lett volna.

A sheffieldi egyetem két matematikusa, *Frazer Jarvis* és *Bertram Felgenhauer* szoftverekkel kiszámolta, hogy a kilencszer kilences (későbbiekben 9x9) rácsban 6670903752021072936960 különböző helyes változat létezik. Ez egy olyan hatalmas mennyiség, hogy még egy számítógépnek is évezredekig tartana lejátszani az összes lehetőséget. Tehát attól sem kell félni, hogy elfogynak a feladványok.

A Sudoku története

A Sudoku japán szó, de a név nem azt jelenti, hogy a játék az ázsiai országból származik.

1783-ban *Leonhard Euleregy* svájci matematikus fektette le a játék alapjául szolgáló elméletet, amely arról szólt, hogy egy tetszőleges számsorban és számoszlopban úgy kellett elhelyezni a számokat, hogy adott oszlopban és sorban azok ne ismétlődjenek. Görög számokkal alkotta meg a „bűvös négyzetet", ezért sokáig a „Görög-latin kockák"-nak nevezték.

A kiegészítő szabályt egy nyugdíjas amerikai építész, *Howard Garns* találta ki, 1979-ben. Az akkor már 50 éve keresztrejtvény újságokat kiadó Dell Magazin publikálta először az általa készített fejtörőt.

Garns egy 9x9 négyzethálós modellel lepte meg az olvasókat, amelyeket 3x3 négyzetes hálós egységekre, tömbökre bontott. A játékot ekkor „*Number Place*"-nek nevezték el. A siker mérsékelt volt.

A Sudoku mai népszerűsége egy japán kiadónak, a *Nikolinak* köszönhető. A japán „*Füles magazin*" 1984 áprilisában közölt egy új játékot, a következő néven: „*Suji wa dokusin ni kagiru*" (Magyarul: a számok egyedi helyűek). A játékot az előzőekben említett Dell Magazin hasábjain fedezte fel *Kaji Maki*, a Nikoli elnöke, aki azonnal úgy döntött, hogy népszerűsíti a fejtörőt Japánban. Az eredeti nevet túlságosan bonyolultnak találta, és kissé leegyszerűsítette „Sudoku"-ra. A „Su" a számot jelenti, a „doku" pedig az egyedi helynek felel meg. A játék fantasztikus karriert futott be a japán piacon és azóta is töretlen a népszerűsége. Több folyóirat csak ezzel foglalkozik, és állítják, hogy ők kézzel csinálják a rejtvényeket.

1989-ben Commodore 64-re is megjelent egy Sudoku program DigitHunt címmel.

Az európai karriert Hong Kong-ból indította el *Wayne Gould*. 13 évig ügyvédként dolgozott Új-Zélandon, majd 1982-től tizenöt éven keresztül Hongkongban épített jogi karriert, egészen a városi főbíró címig jutott. Hobbiból előszeretettel írogatott számítógépes játékprogramokat.

1997 tavaszán Tokióban egy könyvesboltban látott meg egy halom Sudoku könyvet. Felvillanyozta ez a találkozás a játékkal. Felesége - aki nyelvész professzor volt az Egyesült

Államokban - javaslatára, próbaként megjelentették az első Sudoku rejtvényt New Hampshire helyi lapjában, ahol az amerikai házuk volt. A Conway Daily Sun szerkesztői nagyon meglepődtek azon, hogy mekkora érdeklődés mutatkozott az új játék iránt.

2004 októberében, az akkor 59 éves *Gould* Londonba utazott. Tudta, hogy a britek szenvedélyes rejtvényfejtők, és angol napilap nem létezik rejtvényoldal nélkül. Előzetes egyeztetés nélkül ment be a Times kiadójához, ahol sikerült egyességet kötnie, mely úgy szólt, hogy a megjelenés viszonzásaként ő ingyen készíti a feladványokat. 2004. november 12. történelmi nap volt a Sudoku történelmében, a Times újságban megjelent az első, *Gould* által készített fejtörő. Azóta is az ő rejtvényei jelennek meg a lapban.

A Sudoku-mánia nem állt le a briteknél. Pillanatok alatt terjedt el az európai országokban is.

A média a XXI. század Rubik kockájának nevezte.

A játék óriási sikerrel tért vissza az Egyesült Államokba is. 2004 közepétől a New York Post állandó Sudoku rovatot indított. Július 11-én a USA Today és a Daily News elindították a Sudoku rovatukat, kiszorítva az évtizedek óta hagyományosan azon a helyen szereplő keresztrejtvényeket. Nem meglepő, hogy a könyvesboltok és újságos standok is tele vannak a Sudoku játékokat tartalmazó magazinokkal.

Ausztráliába 2005 májusában érkezett a játék, s a legnagyobb napilapok csináltak neki hírverést. David Burkett, a Time Inc. kiadó vezérigazgatója maga is meglepődött, hogy milyen gyorsan terjedt el a Sudoku az ötödik kontinensen.

A Sudoku őrület megérkezett Magyarországra is. 2005 őszén jelent meg az első Sudoku magazin magyar nyelvű leírásokkal, majd a könyvesboltokba is eljutottak az első Sudokut tartalmazó kiadványok.

Fülesben rendszeresen jelentek meg 1990-ben Sudoku rejtvények „Bűvös négyzet” néven. Sőt! Többféle változatban is!

De, ahogy az már nálunk lenni szokott: kis újság... kis ország... kis siker...

Diplomamunka célja

- Egy Sudoku program írása
- Sudoku tábla generálása választható méretben és nehézségű szinten.
- Feltérképezni és megismerni az eddig mások által használt technikákat mind a generálásban, mind a megfejtésben.
- Egy játékfelület létrehozása, ami élvezetesebbé teszi a játékot és segít a kitöltés közben. Illetve lehetővé teszi a játék közbeni lementést a későbbi folytatás érdekében.
- Az eddig ismert algoritmusokat úgy átírni, hogy azok különböző nagyságú tábláknál is jól működjenek. Vizsgálni Milyen időbeli változás lép fel az algoritmus futási idejében.
- Megoldani, hogy mi magunk is csinálhassunk táblát és azt folyamatosan ellenőrizhessük, hogy tudjuk jó nyomon járunk-e vele.
- Továbbá az olvasót megismertetni a Sudoku világával és néhány megoldási módjával, mind papíron való megoldást, mind számítógépes megoldást tekintve.

Példa egy Sudoku játékra

Mielőtt tovább mennénk nézzünk egy példát, hogy hogy is játszható egy ilyen játék.

A következőkben a tábla celláit a következőképpen számozom:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

A példát egy **4x4**-es játékon mutatom be, aminek a kitöltése igen egyszerű.

	2	3	
1			4

Első lépésként kezdjünk el végigmenni a cellákon és vizsgáljuk meg, hogy találunk-e olyan cellát ahova csak egy érték írható be.

Az első cellába jöhet a 3-as és 4-es, tehát ez most nem jó. A másodikba írhatunk 1-est, 3-ast és 4-est is. Most megnézhetnénk a következő két cellát is, de mivel látjuk, hogy szimmetrikus a tábla szerkezete, ezért nem lenne a következő két cellában sem biztosra beírható érték (legalábbis nem ezzel a módszerrel).

Az 5. elemet vizsgálva megtaláljuk az első olyan üres cellát, ahova csak egy számot írhatunk, ami a négyes. Ezzel szimmetrikus a 8. cella, melybe ugyanúgy csak egy elem írható be, ami az 1-es.

4	2	3	1
1			4

Tovább haladva a következő üres elem a 10. cellában van, ahova nem kerülhet a sora miatt 1-es és 4-es, illetve az oszlopa miatt 2-es, így az elem ami a helyére kerül a 3-as lesz és hasonló képen a rá következő elem is kitölthető.

4	2	3	1
1	3	2	4

Tovább haladva a következő nem üres cella a 13. ahova nem kerülhet 1-es, 3-as és 4-es. Tehát a 2-es szám maradt ki és ez kerül a cellába. A tömbben lévő még ki nem töltött elem pedig a 4-es lesz.

4	2	3	1
1	3	2	4
2	4		

A következő, azaz 15. elem kitöltése is kézenfekvő, hiszen csupán az 1-es az ami még beleírható és miután beírtuk a tömbben a még be nem írt egyetlen érték a 3-as lesz.

4	2	3	1
1	3	2	4
2	4	1	3

A maradék néhány elemet ezek után könnyű lesz kitölteni, hiszen mindenhol az oszlopot kell megfigyelni, hogy melyik az az egy elem, ami még nincs beírva és megszületik a feladat megoldása:

3	1	4	2
4	2	3	1
1	3	2	4
2	4	1	3

Példa generálásra

Először nézzük egy 4x4-es táblának a generálását.

A tábla celláinak számozása ugyanúgy történik most is, mint a „Példa egy Sudoku játékra” fejezetben.

Mielőtt nekilátunk el kell dönteni, hogy üres táblából kívánjuk generálni véletlen számokkal, vagy egy kirakott táblából kívánjuk generálni a tábla redukálásával és a táblán végzett transzponálásokkal.

1	2	3	4
3	4	1	2
4	1	2	3
2	3	4	1

Az üres táblából való kiindulásnál szükséges, hogy feltöltsük elegendő adattal a táblát, hogy érdemes legyen vizsgálni a kitölthetőségét és azt, hogy van-e több lehetséges megoldása a feladatnak. Ebben az esetben 4 db mező kitöltésére van szükség, hogy megoldható lehessen a feladat.

		3	1
	2	-	4
		1	

		3	2
	2	-	4
		1	

Miután a szabályoknak megfelelően beírtunk négy számot a táblázatba megvizsgálhatjuk, hogy megoldható-e és hogy hány megoldása van.

Jelen esetben minden szám szabályosan van feltéve, de a tábla nem kitölthető. Ha jobban megnézzük, akkor a jobb felső tömbben hiányzik mind az 1-es mind a 2-es számszámjegy, ám csak az egyiket tudjuk beírni, méghozzá a tömb második helyére. Mivel a tábla nem kitölthető, ezért tudjuk, hogy van olyan elem a táblában ami rossz helyen áll vagy rossz értéket tartalmaz. Jó megoldás lehetne, ha kitudnánk szűrni melyik a hibás (jelen esetben bármely szám törlésével a táblából jó megoldást kapunk) és azt eltávolítani. Viszont egy általános generáló nem tudja megvizsgálni, hogy minek kéne máshol lennie, így hát csak próbálkozik és elvesz véletlenszerűen egyet. Most nekünk ez egy speciális eset, miszerint

mindegy melyiket távolítjuk el.

Mivel tudjuk, hogy 3 elem nem adhat megoldást egy 4x4-es sudoku táblára, így mikor eltávolítjuk az egyik elemet érdemes helyette írnunk egy másikat.

Távolítsuk el a 15. helyen lévő 1-est és írjunk be helyette a 10. helyre egy 1-est, ekkor:

1	4	3	2
3	2	1	4
4	1	2	3
2	3	4	1

1	4	3	2
3	2	1	4
2	1	4	3
4	3	2	1

Újabb tábla, újabb vizsgálat, ebben az esetben több jó megoldásunk is van.

Mivel több jó megoldása is létezik a feladatnak, ezért kell még hozzátenni plusz elemet a táblához. Ezt megtehetjük véletlenszerűen, ahogy eddig is, vagy megtehetjük, hogy megoldjuk (a vizsgálathoz amúgy is meg kell) és ahol több lehetőség is van, ott választunk következő lépésben egy elemet a táblához. Az első esetnek nagy hibája, hogy választhatunk akár oda is elemet, ahol most a kitöltés folyamán egyértelmű volt az érték minden megoldás esetén, így hiába választottunk új elemet a táblánkhoz, az nem visz előrébb, sőt még hibás megoldáshoz is vezethet, hiszen ha pl. az első elemnek választjuk a 4-est akkor egy hibás táblához jutunk, pedig a szabályoknak megfelelően jó lenne oda a 4-es:

4	-	3	
1	2		4
	1		

4	-	3	
3	2		4
	1		

Ha a másik irányba indulunk, miszerint a kitöltés alapján választunk egy értéket, mely alapján már egyértelmű lehet a tábla, akkor kapunk egy generált játékot, mert annak már egyértelmű lesz a megoldása:

1	4	3	2
3	2	1	4
4	1	2	3
2	3	4	1

Így kaptunk egy generált játékot az üres táblából, a következő lépésekben:

1. Felírtunk 4 véletlen számot:

		3	
	2		4
		1	

2. Mivel nem volt megoldható, töröltünk belőle egy elemet és mivel 4 elem minimum kell, ezért egy lépésen belül tettünk be a törölt helyett még egyet.

		3	
	2		4
	1		

3. A táblának több jó megoldása volt, ezért tettünk hozzá egy plusz elemet a kitöltés segítségével.

		3	
	2		4
4	1		

A fentiekben előforduló esetek léphetnek fel nagyobb tábláknál is és hasonlóan kell dolgozni velük, hogy egy üres táblából Sudoku táblát kapjunk.

Kitöltött táblából való generálás:

Már a fentiekben is bemutatott táblából érdemes kiindulni, hiszen egy olyan táblát könnyű előállítani, szinte magától adódik.

Három fő részből tevődik össze a kitöltött táblából való generálás:

1. Megkeverjük a táblát megengedett módon, hogy az továbbra is sudoku tábla maradjon

2. Elveszünk belőle véletlen vagy előre eltervezett helyekről elemeket úgy, hogy közben vizsgáljuk, hogy az adott szám visszaírható-e a többi még fent lévőből.

3. További elemeket veszünk el belőle, de ekkor már az egész tábla kitöltését vizsgáljuk, hogy nem-e kapunk több jó megoldást.

Megengedett keverési szabályok:

- A számjegyek permutációja
- A mátrix transzponálása (sor-oszlop csere)
- A sorok permutálása egy blokkon belül
- Az oszlopok permutálása egy blokkon belül
- A sor-blokkok permutálása
- A oszlop-blokkok permutálása

Bármely lépés kihagyható belőle, de vagy a 2.-nak vagy a 3.-nak szerepelnie kell a listán. Az elsőt bármikor végrehajthatjuk, nem fogja elrontani a sudoku táblánkat.

Továbbra is nézzük a példánkat 4x4-es sudoku táblán.

1	2	3	4
3	4	1	2
4	1	2	3
2	3	4	1

Első lépésként végezzünk el néhány megengedett keverést:

Permutáljuk az 1-es és 2-es számjegyeket:

2	1	3	4
3	4	2	1
4	2	1	3
1	3	4	2

Transzponáljuk a táblánkat:

2	3	4	1
1	4	2	3
3	2	1	4
4	1	3	2

Permutáljuk az első és második sort:

1	4	2	3
2	3	4	1
3	2	1	4
4	1	3	2

Permutáljuk a harmadik és negyedik oszlopot:

1	4	3	2
2	3	1	4
3	2	4	1
4	1	2	3

Permutáljuk az első és második sorblokkot:

3	2	4	1
4	1	2	3
1	4	3	2
2	3	1	4

Permutáljuk az első és második oszlopblokkot:

4	1	3	2
2	3	4	1
3	2	1	4
1	4	2	3

Minden keverésre felírtam egy példát, hogy láthassuk hogy is működik. Attól függetlenül, hogy én most minden keverést csak egyszer használtam és szépen egymás után, még lehet és érdemes is őket keverni, hiszen ha pl. két ugyanolyan keverést csinálnánk ugyanazon paraméterekkel akkor visszakapnánk a két keverés előtti táblát és olyan lenne mintha nem is csináltunk volna semmit.

Kész vagyunk a keveréssel, most jöhet a redukálás. Először kezdjük el véletlenszerűen és minden egyes kivett elemet nézzünk meg, hogy visszatudunk-e írni a helyére a többi ismeretében.

4	1	3	2
2	3		1
3	2	1	4
1	4	2	3

Elvettük a 7. helyről a 4-est, ezt az elemet könnyen visszatudnánk írni, hiszen mindegy, hogy sorát, oszlopát vagy tömbjét nézzük, csakis a 4-es írható be.

4	1	3	2
2	3		1
3		1	4
1	4	2	3

Szintén könnyen meghatározható a most elvett 2-es a 10. helyről.

4	1	3	2
2	3		1
3			4
1	4	2	3

Elvettük a 11. helyről az 1-est. Ha megnézzük az egyesen kívül az adott helyre semmilyen más szám nem írható be, hiszen a sorában, oszlopában és tömbjében megvan minden más szóba jöhető szám az 1-esen kívül.

4	1	3	2
2	3		1
3			4
	4	2	3

A 13. helyről elvett 1-es is könnyen látható hogy visszaírható a többi fent lévő elem ismeretében.

4		3	2
2	3		1
3			4
	4	2	3

A 2. helyről elvett 1-es is könnyedén belátható hogy kitölthető a még fent lévő számok ismeretében.

4		3	2
2	3		1
3			4
	4	2	

A 16. elem értéke az oszlopa segítségével könnyen meghatározható, hiszen csak a 3-as hiányzik belőle.

4		3	2
2	3		
3			4
	4	2	

A 8. helyről leszedett 1-es is meghatározható, mivel a sorában ott van a 2-es és hármas és az oszlopában a 4-es.

4		3	2
	3		
3			4
	4	2	

Az 5. helyről elvett 2-es is visszaírható de itt már nem elég a hozzá tartozó sort, oszlopot és tömböt nézni, mivel már az 1-es lekerült a színről. Ha megnézzük a sorában, oszlopában vagy tömbjében lévő még ki nem töltött elemeket, akkor láthatjuk hogy másik vele egy sorban, oszlopban vagy tömbben lévő üres elem helyére sem kerülhet a 2-es szám, tehát egyértelműen az 5. helyre kerül a 2-es.

4		3	2
3			4
	4	2	

A 6. helyről elvett 3-as is visszaírható, mivel az adott tömbbe egyedül erre a helyre lehet 3-ast tenni.

		3	2
3			4
	4	2	

Az 1. helyről elvett 4-es visszaírható, mert a sorában csak is egy helyre lehet írni a 4-est és ez az 1. hely, ahonnan leszedtük.

Több értéket már nem tudunk úgy leszedni, hogy az egy lépésben visszatehető legyen, így hát jöhet a 3. lépés, mikor megvizsgáljuk a táblát, hogy ha elveszünk belőlük, akkor a még fent lévők egy vagy több jó megoldást adnak, ha egyet, akkor leszedjük a kiválasztott elemet, ha többet akkor visszaírjuk.

		3	2
3			4
	4		

Kiválasztottuk a 11. elemet, aminek a hiányában még egyértelmű megoldást kapunk, ezt eltávolítjuk a táblából és utána több elemet nem tudunk eltávolítani és kész a játékunk.

Megoldási módok és jellemzőik

Az interneten és a megoldásokkal foglalkozó könyvekben, folyóiratokban számos olyan módszert találunk, mely alkalmas már önmagában is egy tábla helyes kitöltésére.

- http://www.ocf.berkeley.edu/~jchu/publicportal/sudoku/sudoku.paper.html, ahol a Knuth's Dancing Links algoritmusát írják le.
- http://en.wikipedia.org/wiki/Algorithmics_of_sudoku, ezen a honlapon több algoritmus is szerepel a megoldási témában.

A hangsúlyt azokra a módszerekre kell fektetni, ami alapján a játékkal foglalkozó felhasználó is dolgozik, azaz nem csupán az a lényeges, hogy kitölthető-e egy adott tábla egyértelműen, de az is hogy az adott tábla kitöltése mennyire igényel a játékostól radírozást, vagyis előre gondolkozást a kitöltés egyes fázisainál.

A legtöbb úgynevezett backtrack nélküli algoritmus arra épít, hogy megjelöli minden még nem kitöltött cellánál, hogy mely értékek írhatóak be. Ezzel csupán az a gond hogy ezekhez az algoritmusokhoz legtöbbször adott soron, oszlopon vagy tömbön belül minden nem kitöltött értéknél fel kell hozzá írni ezeket a lehetséges adatokat, hogy valamire rájöjjünk és ezt egy kitöltő csak végső esetben teszi meg, amikor már nem tud más módszerhez nyúlni. Eme észrevétel alapján elsősorban olyan algoritmussal kell/kéne nekünk is dolgozni, ami nem igényli tőlünk eme plusz adatok felírását.

Persze a kevesebb adattal való dolgozás több összehasonlítást és ezáltal hosszabb algoritmusokat fognak eredményezni. Ez a későbbiekben bemutatásra kerülő programomnak az egyik komoly hibája/hiányossága.

A legtöbb algoritmus nem összetett, így önmagában nem alkalmas, vagy csak igen lassan a feladat megoldására. Ennek megfelelően a programom sem egyetlen algoritmussal dolgozik, mikor megold, vagy generál egy táblát. A megoldásnál még mielőtt használnám a backtrack módszert (hogy kipróbálom a lehetséges kitöltéseket és ha nem jó akkor mást írok a helyére) megpróbálom megoldani a táblát egészében, vagy csak részben olyan módszerekkel/ algoritmusokkal, melyeket alkalmazva a beírt számjegyek biztosan helyesek és a további algoritmusok számára lecsökkentik a próbálkozások számát.

Egy megfelelő algoritmus lehetne, hogy a kettőt összevonjuk és mikor olyanhoz érünk,

ahol nem vagyunk biztosak a kitöltésében, akkor a cella lehetséges értékei közül választunk egyet és úgy tartjuk számon ezt a cellát és az ezután kitöltött cellákat, hogy ami érték benne és az általa feltételezett későbbiekben kitöltött cellákban van még nem biztos hogy helyes a tábla egészére nézve, így ha a továbbiakban olyan cellára találunk, amit már nem tudunk kitölteni, akkor visszatudjuk bontani és más értékkel kell hogy próbálkozzunk, hiszen az adott cella adott értékére a táblának nem lesz megoldása. Ezáltal jócskán csökkenhet a véletlenszerűen kitöltendő cellák száma és így nem kéne az összes cella összes lehetséges értékére végignézni hogy az jó lesz-e.

Annak az eldöntésére, hogy a játéknak pontosan egy megoldása van-e a backtrack módszer egy jól és egyszerűen használható algoritmust ad, miszerint töltsük ki egyféleképpen és vegyük úgy hogy az utolsó kitöltött cella értékére sem volt jó a tábla kitöltése. Ekkor ha az algoritmus szépen visszamegy az első elemig, amit kitöltöttünk és aszt mondja hogy nincs olyan amire a tábla kitölthető, akkor biztosak lehetünk hogy egy megoldásunk van. Persze ez azt jelenti hogy ahhoz hogy eldönthessük egyértelmű megoldása van-e a táblának, végig kell mennünk minden egyes kitöltetlen cellán és végigpróbálni az összes lehetséges értéket.

Nézzünk néhány már ismert algoritmust:

Naked single:

Azokat a cellákat tölti ki ez az algoritmus, melyeknél csupán egy értéket vehet fel a cella, mivel a többi érték megtalálható a cella sorában, oszlopában illetve tömbjében.

Kétféleképpen lehet megvalósítani:

Az egyik, hogy a még nem kitöltött cellák lehetséges értékeit felírjuk és megvizsgáljuk, hogy melyik cellában maradt egy lehetséges érték. Mikor találunk egyet, akkor azt beírjuk és a vele szabály szerint kapcsolatban álló még nem kitöltött cellák lehetséges értékeiből elvesszük ezt az értéket. Amennyiben ekkor keletkezne olyan cella, ami még nincs kitöltve de lehetséges értéke sincs, akkor tudjuk, hogy a tábla nem megoldható.

A másik módszer, hogy sorba megyünk a még nem kitöltött cellákon és hasonlóképpen megvizsgáljuk, hogy milyen lehetséges értékeket vehet még fel, amennyiben csupán egy értéket vehet fel, akkor azt beírjuk és haladunk tovább. Ez a fajta megoldása sokkal tovább tart, hiszen minden egyes alkalommal mikor az adott cellára érünk végig kell néznünk 3*N-3 db elemet ahhoz, hogy megtudjuk milyen lehetséges értékeket vehet fel, de viszont nem kell plusz adatot tárolni. Megjegyzés: Általában a játékosok ezt a megoldást választják, hiszen

ekkor nem írják teli a táblát.

Nézzünk rá egy kis példát a **Naked single-re**:

2 3 4	3 4	1 2	1 2 3
1	3	2	**4**
4	**2**	**3**	1
3 4	1 3 4	1 2 4	1 2

A táblának az 5. 8. 10. és 11. cellája van kitöltve, és ezek alapján a többi cellának 4 lehetséges értéke közül leszedtük azokat, amikkel kapcsolatban vannak.

Ha megnézzük első lépésben 4 db olyan cellát is találunk, ami még nincs kitöltve és egyetlen lehetséges értéke van. Ezek sorra a 6. 7. 9. és 13. cellák. Töltsük fel ezeket:

2	3	1	3
1	**3**	**2**	**4**
4	**2**	**3**	**1**
3	1	4	2

A következő lépésben már az összes cellának az értéket kitudjuk tölteni, hiszen mindenhol csak egy lehetséges érték maradt.

Hidden single:

Végigmegy egy soron, oszlopon vagy tömbön és megnézi, hogy van-e olyan érték, ami még nincs az adott soron, oszlopon vagy tömbön és csupán egy helyre lehet beírni, amennyiben talál ilyet azt beírja és vizsgálja tovább a táblát. Miután beírt egy értéket, utána kihúzza azt az értéket a hozzá tartozó szomszédai (egy elem szomszédjának tekintek minden olyan elemet, ami vele egy sorban, oszlopban illetve tömbben van) lehetséges értékei közül. Ha ekkor fellép olyan eset, mikor létezik elem ami nincs kitöltve és lehetséges értékei is

elfogytak, akkor a tábla nem kirakható.

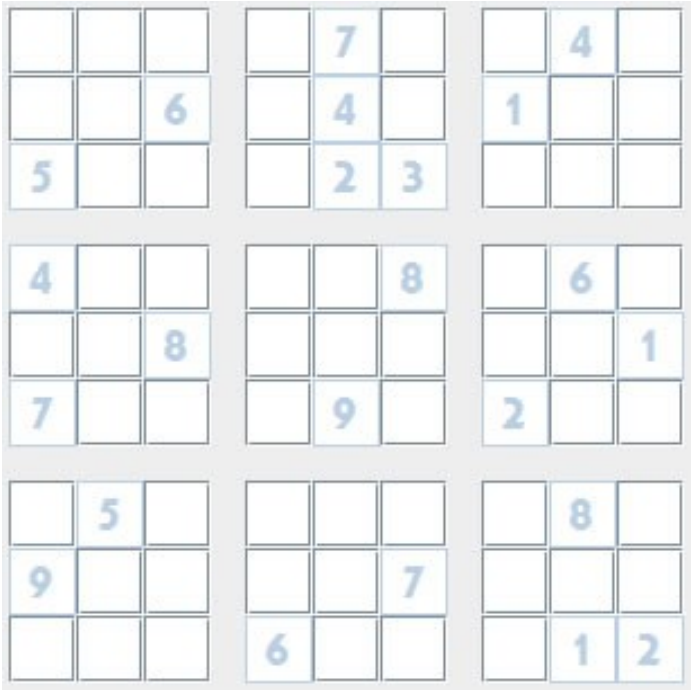
BackTrack:

Mikor már a többi algoritmussal nem tudunk tovább haladni az adott táblán, akkor jöhet a backtrack.

Elkezdjük a tábla elejénél és folyamatosan megyünk végig balról jobbra fentről le irányban a tábla elemein, ha olyanhoz értünk, ami még nincs kitöltve, akkor megnézzük melyik az a legkisebb szám ami beleírható és megyünk tovább. Amikor olyanhoz érünk, aminek már nincs lehetséges értéke, akkor visszamegyünk a legutóbb kitöltött elemig és egy a beírt számnál nagyobbat írunk be a helyére, majd megyünk tovább a kitöltésben. Amennyiben végigmegyünk a táblán és már az NxN+1. Elemet kéne feltölteni (ami nem létezik), akkor megállunk és tudjuk, hogy a megoldás amit kaptunk helyes, hiszen minden elemhez írtunk értéket és azokat helyesen írtuk be. Amennyiben az algoritmus a 0. elemet szeretné megváltoztatni (ez sem létezik), akkor viszont biztosra tudjuk, hogy a táblának nincs megoldása.

Ahhoz hogy megtudjuk hány jó megoldása van, először végig kell rakni a táblát az előbb leírtak szerint és az utolsó kitöltött elemet növelni, ezáltal visszabontjuk a kirakott táblánkat. Amennyiben a tábla visszamegy az elejére, tudjuk hogy a táblának csak egy megoldása volt, méghozzá az amit az előbb találtunk meg. Ha viszont újra elérünk a tábla végéig, akkor tudhatjuk, hogy a táblánknak több mint egy jó megoldása van. Számunkra az hogy mennyi és ezek melyek nem érdekes a generálás szempontjából, hiszen a játékosnak csupán egyértelműen kitölthető táblát kell adni.

Generálási módok és jellemzőik



A fentiekben egy általános Sudoku táblát látunk, mely 9x9-es méretű. Ezt a szakdolgozatomhoz használt programmal generáltattam. Ha valaki jobban megvizsgálja láthatja, hogy 23 elemből áll, de már tovább nem redukálható, azaz nem lehet egy cellának az értékét sem törölni úgy, hogy utána is egyetlen megoldást kapjunk, ez alapján látható, hogy amennyiben a generálás véletlenszerűen történik és nem teszünk bele visszalépéses lehetőséget (miszerint amennyiben számunkra nem megfelelő a kitöltött mezők száma, akkor visszategyen törölt elemeket és más elemek törlésével foglalkozzon), akkor átlagosan csak hasonló méretű táblát fogunk kapni és ritkán érhetjük el, hogy csupán 17 szám kitöltésével kapjunk egyértelműen kitölthető táblát.

A generálás sebessége nem feltétlenül a generálási algoritmus miatt van, sokkal inkább a generálás által létrehozott tábláknak a tesztelési algoritmusán, amit számtalanszor le kell futtatni míg megkapjuk az eljárás által jónak gondolt végső generált táblát. Szóval egy rossz tesztelési algoritmussal rengeteg időt veszthetünk el.

Két fő részre lehet osztani a generálási módokat.

Már kitöltött táblából való generálás.

Üres táblából történő generálás.

Kitöltött táblából történő generálás

A már kitöltött táblából történő generálás lényegében a tábla redukálását jelenti, azaz közben nem változik meg az eredmény tábla, amit megkapunk, ha meghatározzuk a megoldását.

Első lépésben szükségünk lesz egy táblára,

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

Ennek a táblának az előállítása igen egyszerű és bármely Sudoku táblánál használható.

A lényege, hogy a bal felső sarokból kiindulva elkezdjük sorba írni az adott Sudokuba írható számjegyeket, ami 9x9-es esetben 1 2 3 4 5 6 7 8 9. Mivel annyi számjegyünk van ahány cellából áll az első sor, ezért mire végigérünk rajta felírtunk minden számot és így az első sorban pontosan egyszer szerepel minden érték. A második sornál is hasonló a helyzet, csak egy másik számmal kell kezdenünk, mivel hogy 1-es már van a tömbben és az oszlopban. Az általam használt algoritmus úgy választ ki az adott sornak kezdő számot, hogy addig nézi a rá következő elemet a sorozatban, míg egy megengedettet nem kap. Így első esetben a következő szám az 1 lenne, de mivel az nem lehet a Sudoku szabályai szerint, ezért veszi a 2-őt, ez továbbra sem jó szám hogy beírhassuk, és így eljutunk a 3-as után a 4-eshez, ami már megengedett, ekkor ezt beírjuk és újra végigírjuk a számokkal az adott sort, itt

lényeges hogy sorrendben írjuk őket. Amikor a Sudoku legmagasabb számjegyéhez érünk, ami jelen esetben a 9, akkor a következő számjegy a legkisebb számjegye lesz, ami az 1.

Ebben a módszerben az a jó, hogy az NxN-es táblában csupán N-1 cellára kell ellenőrzést végeznünk illetve monotonon irkálni a számokat egymás után és mégis biztos hogy helyes táblát fogunk kapni.

Miután megcsináltuk a táblát a fentiek alapján, szükségünk lesz a tábla átalakítására ahhoz, hogy ne mindig ugyan az legyen a táblának a megoldása, ezt az átalakítást a következő lépésekkel tehetjük meg:

- A számjegyek permutációja
- A mátrix transzponálása (sor-oszlop csere)
- A sorok permutálása egy blokkon belül
- Az oszlopok permutálása egy blokkon belül
- A sor-blokkok permutálása
- A oszlop-blokkok permutálása

Az első pár lépésnél tudunk anélkül dolgozni, hogy meg kéne határoznunk a tábla teljes megoldását és azt hogy egyetlen megoldása van-e. Egyszerűen elég azt vizsgálnunk, hogy a redukálandó elem, amit aktuálisan eltávolítani akarunk a táblából visszaírható-e a tábla többi még kitöltött elemeinek ismeretében. Amennyiben kitölthető, úgy egyértelműen tudjuk, hogy ha eddig jó volt a tábla, tehát csak egyetlen megoldása volt és minden elem egyértelműen meghatározható volt, akkor egy elem elhagyása, melynek kiderítése is egyértelmű, továbbra is jó táblát fog nekünk adni. Amikor már nem találunk olyan értéket, ami ilyen módon eltávolítható, akkor egyik lehetőségként befejezzük a tábla redukálását és késznek nyilvánítjuk a táblát (ez akár lehet a gyenge játéknak az előállítási módszere is). Másik lehetőség, hogy a táblát tovább redukáljuk, ekkor már érdemes lépésenként megvizsgálni hogy az adott redukált tábla kitölthető-e, illetve hogy hány megoldása van, így ez sokat ront a futási idején, de viszont ha valahol rossz elemet választottunk ki az eltávolításra, akkor azt visszaírjuk és kiválaszthatunk egy olyan elemet, melyet még nem próbáltunk meg eltávolítani. Egyik lehetséges módszer arra, hogy nehéz feladatot állítsunk elő, ha minden egyes még nem vizsgált kitöltött cellát kipróbálunk, hogy ha eltávolítjuk még egy megoldású lesz-e a feladat.

Persze ez borzasztóan időigényes, hiszen annyiszor kell kétszer kitölteni hozzá a táblát, amennyi még meglévő cellaértékünk van. Továbbá az is növeli a kitöltések futás idejét, hogy közben egy már nehezebben kitölthető táblával van dolga a programnak. Ennek a futási időnek a kiküszöbölése érdekében lehetne egy olyan megoldást használni a redukálásnál, hogy miután már eltávolítottuk az összes könnyedén visszaírható értékeket, egyszerre több még nem üres cellának az értékét is eltávolítjuk és csak ezután végzünk el egy ellenőrző kitöltést, hogy megtudjuk a törölt cellaértékek maradhatnak-e törölve, vagy esetleg térjünk vissza oda ahol még nem töröltük őket (illetve írjuk vissza a törölteket).

Amennyiben a kiválasztott cellák törlése után is csupán egy jó megoldása van a feladatnak akkor sikerült a redukálásunk és rengeteg műveletet spóroltunk meg vele. Így csökken a redukálandó táblának a mérete és egy kisebb halmazból tudjuk kiválasztani azokat a cellákat, melyek értékeit töröljük és újra megnézzük, hogy a tábla még egy megoldású-e. Ekkor persze már a tábla méretének csökkenése miatt az egyszerre kitörölt elemek számát is érdemes csökkenteni.

Ha több jó megoldást is kapunk akkor viszont kénytelenek vagyunk visszaírni mindezen elemeket, hiszen nem tudjuk melyik elem vagy elemek miatt lett több jó megoldása is a feladatnak. Ezáltal nem csökken a tábla mérete. Ekkor is érdemes újra próbálkozni és az egyszerre redukált cellák számát is csökkenteni, hiszen lehet az egy lépésben törölt elemek számának nagysága miatt nem sikerült jó megoldást kapnunk.

Üres táblából történő generálás

Ennél a módszernél nem kell redukálnunk a táblánkat, csupán hozzátenni szükséges. Alap esetben egy teljesen üres táblánk van és ezt kell feltölteni úgy hogy a beírt értékek egyetlen megoldást adjanak.

Egyik módszer lehet a véletlen módszere, mikor a tábla feltöltését véletlen helyeken és számokkal végezzük el.

Mielőtt vizsgálnánk a tábla kitölthetőségét szükséges elegendő mennyiségű elemet feltölteni a táblázatba. Mivel tudjuk, hogy a 4x4-es táblában a minimálisan kitöltött elemek száma 4, ami az egész táblának ¼ része, ami 25% és a 9x9-es táblának a minimálisan kitöltött elemek száma 17 (bár még nem bizonyított hogy nincs kevesebből kitöltött tábla) ami 17/81 része, ami körülbelül 21%, ezért egy jó kiinduló pont lehet, ha az elején az első vizsgálat előtt kitöltjük a táblának legalább 20%-át.

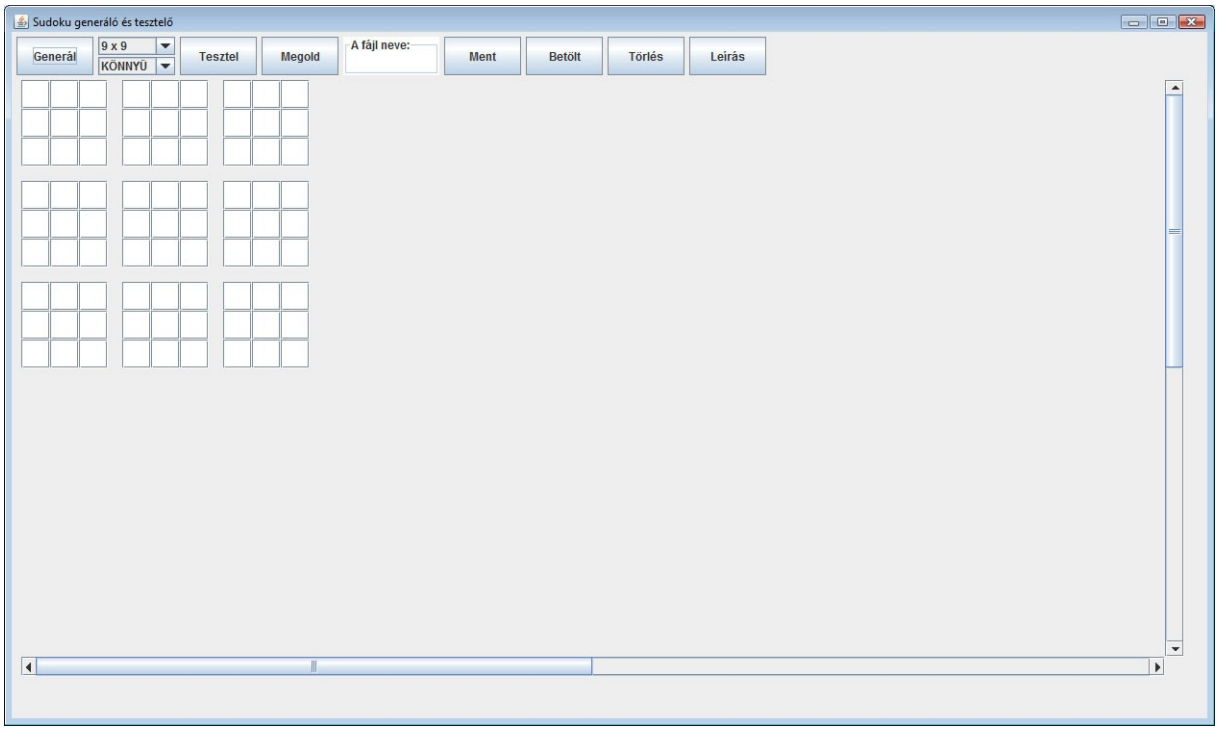
Másik módszer lehet az is, hogy a táblát előre kijelölt helyeken töltjük fel adatokkal ezáltal egy formát kialakítva a játéknak, persze itt is szükséges hogy elegendő kitöltött cella és inkább kicsit több helyen írjunk be számokat a táblába, hogy a játék megoldható lehessen. Ennél a megoldásnál, hogy a formát megtartsuk érdemes a későbbiekben a számokat helyettesíteni másikkal, amennyiben nem kapunk egyértelmű megoldást, vagy ha rossz megoldást kapunk, ahelyett, hogy felvennénk az alakzaton kívül más helyeken is elemeket. Miután kész az adott alakzatra a játék, a nehezítés érdekében érdemes lehet megnézni, hogy a tábla redukálása mellett megoldható marad-e a játék, persze ekkor megbontjuk az általunk kigondolt alakzatot.

Program kezelése

Amennyiben a gépen van Java telepítve akkor a program további telepítést nem igényel.

A program mellé mellékeltem egy jdk programot, aminek telepítése után már használható a program.

Elindítás után a következő ablak jelenik meg:



A játékot a Sudoku szabályainak megfelelően kell játszani, azaz:

Minden sorban, oszlopban és tömbben egy szám csak egyszer szerepelhet.

A játékban a generálás előtt lehetőségünk van beállítani a generált tábla nehézségét könnyű illetve nehéz fokozatra.

A "Generál" gombbal generálhatunk egy játékot a beállított méretű táblára.

A "Tesztel" gomb megmondja, hogy az aktuálisan megjelenő játéknak hány megoldása van, illetve van-e neki.

A "Megold" gomb megoldja a játékot és az általa beírt számokat pirossal írja, ezzel is mutatva hogy miket töltött ki ő. Amennyiben nem megoldható vagy több megoldása van, azt jelzi, ekkor nem tölti ki a táblát.

A "Ment" gomb elmenti "a fájl neve" mezőben szereplő fájlnévre az adott játékot, amennyiben már létezik a fájl, akkor azt felülírja, ha nem, akkor létrehozza.

A "Betölt" gomb betölti a "a fájl neve" mezőben szereplő fájlban lévő játékot a programba, ekkor törlődik az előtte fent lévő játék.

A "Törlés" gomb több lépcsőben töröl:

1. Törli a piros számokat.
2. Törli a kék (általunk beírt) számokat.
3. Felszabadítsa a blokkolt számokat.
4. Törli a fent lévő elemeket.

Sorra halad a lépéseken és ha valamelyik lépésnek volt eredménye, akkor megáll.

Fontos megemlíteni, hogy a játék a mentéseket abba a könyvtárba helyezi, ahol a program is megtalálható, így a mentések működése érdekében célszerű a programot egy a felhasználó számára írható és olvasható könyvtárba helyezni.

A táblát mi magunk is elkezdhetjük feltölteni saját elképzelésünknek megfelelően és közben tudjuk ellenőrizni, hogy jó-e, illetve hogy egyértelmű-e a megoldása. Amennyiben nem kívánunk magunk játékot létrehozni, úgy a generálás gombbal a program létrehoz egy játékot, amit ezután vagy lejátszunk, vagy akár arra is megvan a lehetőségünk, hogy a töröl gombbal leszedjük az írásvédelmet a kitöltött cellákról és átalakítsuk a program által generált játékot.

A program jól használható akár memóriajáték gyanánt is, hiszen ha egy játékot megoldatunk a géppel és utána rövid megfigyelés után töröljük az általa pirossal kitöltött cellák értékeit, akkor emlékezetből visszaírhatjuk azokat és ezt amennyiben nem sikerült megcsinálni, akár újra csinálhatjuk, míg a tábla kitöltése kész nem lesz. Ez pl. jó módszer lehet arra, hogy hogyan töltsünk ki egy nagyobb méretű táblát anélkül hogy órákat ülnénk mellette.

Program felépítése

A program két osztályból épül fel.

Az egyik a **Main** osztály ami a program megjelenéséért és működésének kapcsolataiért felel. A másik a **General** osztály, ami a táblák generálásának és tesztelésének algoritmusaiért felel.

A Main osztály fontosabb elemei

A felület felépítésénél a következő egységek vesznek részt:

- JButton osztályból származtatott objektumok: mentes, betoltes, general, tesztel, megold, torles, leiras
- JTextField[] tabla
- JTextField filenev
- JComboBox N, nehezseg
- JScrollBar horizont, vertikal
- Container ablak
- JPanel egesz

Minden gombhoz van egy akció rendelve, ami akkor fut le, ha az adott gombot használjuk.

Mentes: Csupán akkor fut le, ha a filenev(JTextField) -ben van valamilyen adat.

Az éppen fent lévő tábla alapján generál egy General objektumot, amit elment a filenev mezőbe írj fájlnév.mur kiterjesztéssel (a .mur egy fantázia kiterjesztés), amennyiben ez a fájl már létezik, úgy az felülíródik.

Azzal, hogy egy objektumot mentek el ahelyett, hogy a puszta adatok helyett az átláthatóságot, illetve azt teszi lehetővé, hogy a mentett fájlt ne lehessen könnyen változtatni, csupán, ha tudjuk milyen osztály alapján kell azt megnyitni.

Betoltes: A filenev-ben adott fájl-t beolvassa és létrehoz egy General objektumot a fájl tartalma alapján. Ezt az objektumot felhasználva állítja be a megjelenő objektumokat.

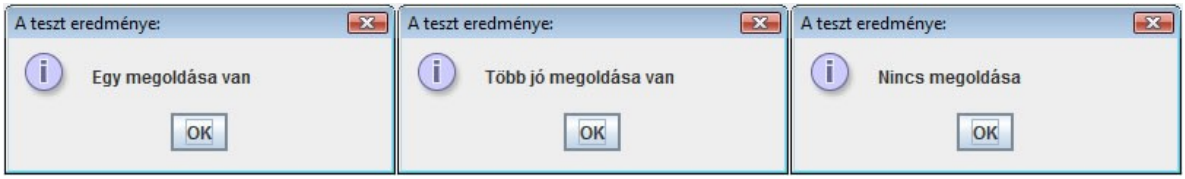
General: Létrehoz egy General objektumot az adott tábla mérete alapján, majd végrehajtatja a generált objektummal a generálási eljárást és lekérdezi a végeredményét, ami

szerint feltölti a tabla[] objektumokat adatokkal, illetve beállítja az írhatósági tulajdonságukat, hogy megkülönböztethessük a generálás által létrehozott elemeket azoktól, amiket mi viszünk fel később.

Tesztel: Feltölti a Generál osztályhoz tartozó objektumot adatokkal és lekérdezi annak megoldhatóságát. A következő három lehetőség léphet fel a lekérdezés közben:

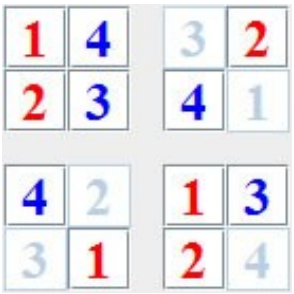
- Egy jó megoldása van
- Több jó megoldása van
- Nincs megoldása.

Megvizsgálja a lekérdezés eredményét és egy felugró ablakban jelzi az eredményt számunkra.

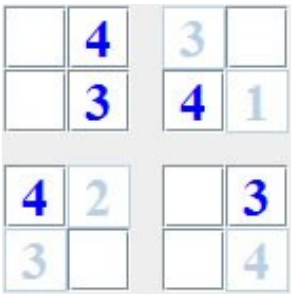


Megold: A tesztel-hez hasonlóan működik, ám ha csupán egy jó megoldást talál, úgy a még nem kitöltött mezőket feltölti a megoldás szerinti értékekkel pirossal jelölve, hogy lássuk mi az amit ő töltött ki, illetve majd a törlésnél lesz szerepe, hogy külön van jelölve. Természetesen, ha több jó megoldása létezik vagy nincs megoldása, akkor azt egy felugró ablakban jelzi számunkra.

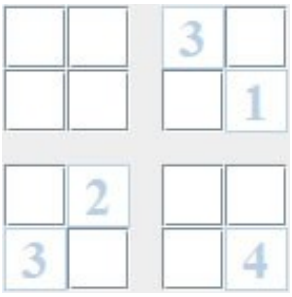
Torles: Sorra megy a tabla[] elemein és vizsgálja milyen típusú mezőket talál kitöltés értelmében, megkülönbözteti a pirossal írt mezőket, a kék írható mezőket és a kék írásvédett mezőkket. Ebben a sorrendben törli az első fajta talált elemeknek az értékeit, így ha van fent minden típusból, akkor 4x-i aktiválásával (lenyomásával) érjük el az üres táblát a következők alapján:



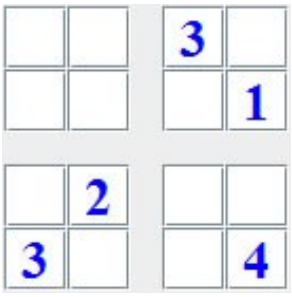
- Pirosak értéke törlődik



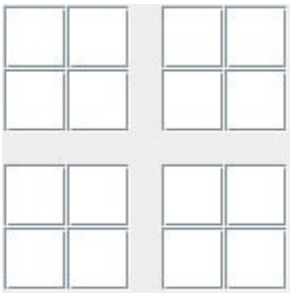
- Kék írhatók értéke törlődik



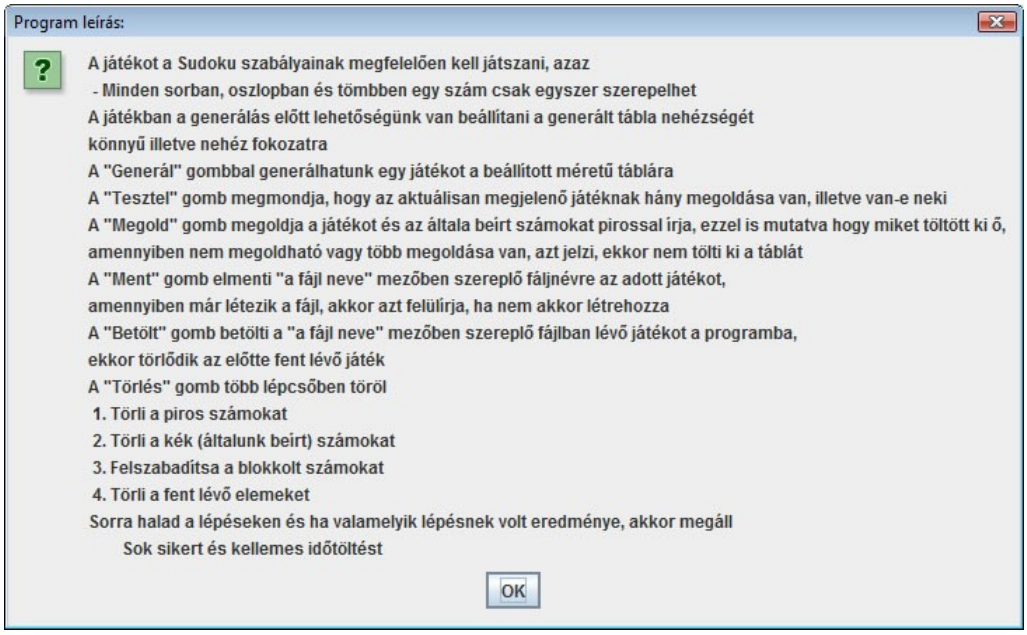
- Kék írásvédett elemek írásvédettsége változik



- Kék írhatók értéke törlődik



Leiras: Egy felugró ablakot jelenít meg, amiben felsorolja a program használatához szükséges ismereteket.



tabla[] (JTextField): Alapból nem tartalmaz értékeket és a program indulásánál jön létre mind a lehetséges 625 eleme, ez felel a program kicsit lassú betöltéséért. Tábla méretének megfelelően vannak jelen számukat tekintve és elhelyezve az ablak (Container) objektumban.

N (JComboBox): Mikor megváltoztatjuk értékét, eltűnik a tábla amin eddig dolgoztunk és megjelenik egy új üres ablak, aminek minden eleme üres.

nehezseg (JComboBox): Ezzel mondhatjuk meg a programnak, hogy a következő generált tábla milyen nehézségű legyen. Egyenlőre két nehézségi szint létezik, a könnyű és a nehéz.

filenev (JTextField): Tartalmát szabadon változtathatjuk. A programban csak mentésnél és betöltésnél van szerepe.

ablak (Container): Tartalmazza a tabla[] elemeit.

Ez választja szét a játékfelületet a gomboktól, hogy ha szükséges, akkor tudjuk a benne lévő objektumokat mozgatni anélkül, hogy az egész ablakot mozgatnánk.

egesz (JPanel): Tartalmaz minden grafikai objektumot. Ez adja a program ablakának grafikai megjelenését és annak beállításait.

horizont, vertikal (JScrollBar):Vízszintes, illetve függőleges mozgatását végzi a tabla[] elemeinek, hogy az egész táblát láthassuk bármilyen szabvány képernyőfelbontás mellett.

A Generál osztály néhány lényeges része

Fontosnak tartom megemlíteni a tábláknak a tárolási módját és azt hogy ilyen tárolás mellett, hogyan lehet feldolgozni azt az algoritmusok számára.

A táblát tömbben ábrázolom. A tábla számozása egy NxN-es tábla esetén, 0-tól n^4-1, ahol n^2=N. A számozás egy tábla esetén a bal felső sarokból indul és balról jobbra halad, ha a sor végére értünk, akkor megyünk a következő sor bal szélső elemére és így tovább, míg elérünk a tábla végére.

Az n számok szükségesek ahhoz, hogy egy kiválasztott cellához megadható legyen a program bizonyos részeinek, hogy melyik oszlopban, sorban, illetve tömbben van. Itt a sor, oszlop tömb megadásánál nem úgy kell érteni, hogy megadjuk 1-N-ig, hogy hányadikban van (balról jobbra fentről le), hanem megadjuk, hogy az a sor, oszlop vagy tömb a tábla mely eleménél kezdődik, ugyanis az algoritmusoknak erre van szüksége ahhoz hogy tudják honnan kell kezdeniük számolni a táblával adott esetben. Nem csupán a ciklusok kezdeti értékeinek beállítása miatt szükséges ezen értékek kiszámítása, hanem a tábla transzponálásainál is fontos szerepet játszik, hogy eldönthessük pl. azt, hogy a kiválasztott két oszlop egy tömbhöz tartozik-e, ami feltétele a két oszlop felcserélésének.

Nézzük meg milyen képlettel lehet meghatározni ezeket az értékeket:

A következő képletekben használt jelek és magyarázataik:

hol-az a cella, amihez tartozó sort, oszlopot vagy tömböt keressük.

%-(mod)maradékos osztás.

/-(div)egész részes osztás.

Sor első elemének kiszámítása:

=(hol / n^2) * n^2

Oszlop első elemének kiszámítása:

=hol % n^2

Tömb első elemének kiszámítása:

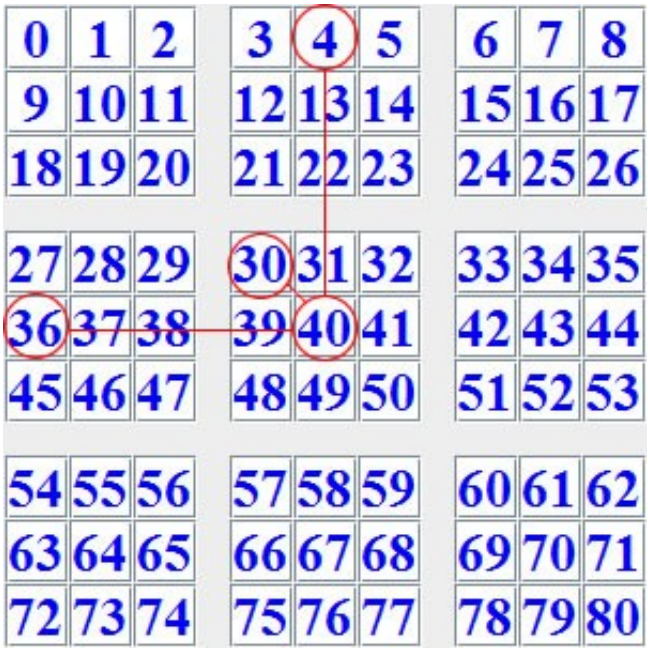
sor=(hol / n^2) * n^2 - ((hol / n^2) * n^2) % n^3

oszlop=(hol % n^2) – (hol % n^2) % n

=sor+oszlop

A számolgatás azért szükséges, hogy több -féle méretű táblához ne kelljen külön algoritmusokat írnunk. Fejlesztési lehetőségként felemlítettem, hogy ne csak NxN-es táblákra legyen megírva a program. MxN-es esetben kissé meg kell változtatni a számításokat és ciklusok paramétereit, ám utána működni fog mind a generálás, mind a megoldás.

Nézzünk példát a számításokra:



Itt láthatjuk az ábrát, ami segít a számolás végeredményét megmutatni.

Legyen a kiválasztott elemünk, azaz a hol=40 és a táblánk mérete 9x9, azaz az n=3.

Első lépésben nézzük meg, hogy melyik a sorának a legelső eleme:

=(hol / n^2) * n^2 azaz (40 / 3^2) * 3^2 = 4 * 9 = 36, ahogy az ábrán is láthatjuk.

Oszlopának első eleme:

=hol % n^2 azaz 40 % 3^2 = 4, az ábra alapján is ezt láthatjuk.

Tömbjének első eleme:

sor=(hol / n^2) * n^2 - ((hol / n^2) * n^2) % n^3 azaz

(40 / 3^2) * 3^2 - ((40 / 3^2) * 3^2) % 3^3 = 4 * 9 - (4 * 9) % 27 = 27

oszlop=(hol % n^2) - (hol % n^2) % n azaz

(40 % 3^2) - (40 % 3^2) % n = 4 - 4 % 3 = 3

=sor + oszlop azaz 27 + 3 = 30

Generálás tesztelése

A generálás ideje a nehézségi beállítástól függ. A mostani beállítás szerint a nehéz fokozat a legalaposabban redukálja a generált táblát, azaz ami elem fent van, azt már csak azért nem szedi le, mert vagy túllépett egy időkereten vagy mert ha leszedné, akkor már nem lehetne a táblát egyértelmű megoldással kitölteni. Ennek megfelelően egy táblának a kitöltése inkább nehéz lesz, mintsem könnyű (kivéve a 4x4-es esetet, amikor a kitöltés mindenféleképpen könnyű).

1000 db tábla generálása könnyű fokozaton

(Ebben az esetben csak a Naked Single és Hidden Single szabályokat használhatjuk)

	4x4	9x9	16x16	25x25
Megmaradt elemek számának átlaga	5,181 (32,3%)	35,506 (43,8%)	130,139 (50,8%)	348,991 (55,8%)
Tábla generálás átlagos ideje (ms)	0,116	0,289	1,178	4,436
Legkevesebb fennmaradt elemek száma	4 (25%)	32 (39,5%)	122 (47,7%)	339 (54,2%)
Legtöbb fennmaradt elemek száma	8 (50%)	41 (50,6%)	138 (53,9%)	360 (57,6%)
A legrövidebb idő alatt generált tábla (ms)	0	0	1	3
A legtöbb idő alatt generált tábla (ms)	3	4	4	11

25 db tábla generálása nehéz fokozaton

(Ekkor minden szabály használata megengedett, még a visszalépés is)

A következőkben csökkentettem a generált táblák számát, ám ezekből is jól látható a különböző nagyságú táblák közötti generálási különbség.

	4x4	9x9	16x16
Megmaradt elemek számának átlaga	4,04 (25,25%)	23,52 (29%)	105,076 (41%)
Tábla generálás átlagos ideje (ms)	1,760	291,720	47749,730
Legkevesebb fennmaradt elemek száma	4 (25%)	21 (25,9%)	98 (38,3%)
Legtöbb fennmaradt elemek száma	5 (31,3%)	25 (30,9%)	114 (44,5%)
A legrövidebb idő alatt generált tábla (ms)	1	41	786
A legtöbb idő alatt generált tábla (ms)	6	1797	610256

25x25-ös tábla 3 db generált táblánál:

a fennmaradó elemek száma átlagosan 625 db-ból 305 db (48,8%)

a generálás ideje átlagosan: 573764.66ms (9min 33s 764.66ms)

a legritkábban kitöltött tábla 625 db-ból 299 db (47,84%)

A legtöbb fennmaradt elem 625 db-ból 314 db (50,24%)

a leggyorsabban generált tábla 4557ms (4s 557ms)

a leglassabban generált tábla 1627205ms (27 min 7s 205ms)

Illetve a 25x25-ös táblánál nehéz esetben gyakran előfordul, hogy 120 perc után sincs kész a tábla generálása.

A tesztelésnél lehet látni, hogy mekkora különbség van a nehéz típusú generálások között időben, míg a könnyűnél a generálás ideje nem sokban tér el, de már ott is tapasztalható hogy nem egyszerűen lineáris változásról van szó a tábla generálásához szükséges időt tekintve, hanem sokkal inkább egy exponenciális változásra hasonlít.

Program bővítésének lehetőségei

A programot rengeteg helyen lehet bővíteni, kezdve a grafikai megjelenéstől egészen az algoritmusok átírásáig/javításáig.

A program csak NxN es Sudoku táblákra van megírva, de kis átírással meglehetne oldani akár NxM-es re is, úgy mint pl 2x3-asra, ahol a 2x3 a tömb méretére utal és 2*3 azaz 6 számjegyből áll. Itt az elnevezés kissé megtévesztő lehet, hiszen pl. a 9x9-est sem a tömbök mérete alapján, hanem az egész tábla mérete alapján nevezték el.

Példa 2x3-as -ra:

1	2	3	4	5	6
4	5	6	1	2	3
5	6	1	2	3	4
2	3	4	5	6	1
3	4	5	6	1	2
6	1	2	3	4	5

Nem csupán a méreteket lehetne tovább bővíteni, de akár egy-egy tábla méreten belül is lehet változtatni a tömbök beosztását, hogy ne négyzethálós, hanem egy általunk kiválasztott alakzatot írjanak le a táblán belül.

Továbbá lehet bővíteni több funkciógombbal és az elhelyezés jobb kihasználása érdekében menürendszert írni rá, hogy több minden elférjen és ne legyen zavaró, ha az ember kis felbontású ablakot használ, hogy nem lát minden funkcióbillentyűt.

Külön nehézséget lehetne még beállítani, ami nem csak a generálásnál adna egy rendszert a játék nehézségét tekintve, hanem akár a játékost is kitöltés közben a nehézségi szintnek megfelelően segíti. Ez a segítség lehetne pl. hogy ha valaki kezdőre állítja, akkor akár még a tippelgetésbe is belemegy a játékos és szól amennyiben az éppen kitöltött mező értéke jó-e oda a feladat megoldását tekintve, tehát nem csak akkor szólna, mikor van az adott sorban, oszlopban vagy tömbben megegyező elem, de akkor is ha a tábla végeredményét tekintve rossz megoldás lesz az adott kitöltéssel az adott elem. Normálnál lehetne, hogy csak akkor szóljon ha a szabálynak ellentmond, illetve nehéz fokozaton egyáltalán nem szólna, hogy az adott elem jó/helyes értékkel lett-e feltöltve vagy sem.

Továbbá sokan szeretik, ha játék közben láthatják a még a cellába írható elemeket, mivel ez sokszor gyorsítja is a játékot nagyobb sikerélményt adva a játékos számára. Ennek a lehetőségét lehetne továbbá bővíteni azáltal, hogy nem csupán billentyűzetről írnánk be a számokat az adott helyre, hanem az adott még betehető elemre kattintva oldhatnánk ezt meg. Ezt a megoldást nehezíti, az hogy a tábla mérete nem csupán 9x9-es mint a legtöbb sudoku programban, hanem rendre változtatható és pl. egy 16x16-osnál már nehezebb mind a 16 számot egy cellába zsúfolni és nem is nézne ki jól. Esetleg cellákra egyesével lehetne megcsinálni, hogy mikor kijelölöd vagy csupán ráviszed a kurzort az adott cellára, akkor egy másik helyen megjelenne a még beleírható számoknak egy halmaza, ahonnan kiválaszthatjuk az általunk jónak vélt elemet.

További bővítési lehetőségként lehet helyettesíteni a számokat betűkkel, alakzatokkal, képekkel vagy akár színekkel is gyerekek számára, akik annyira még nem boldogulnak a számokkal vagy bárki másnak, aki szereti a változatosságot.

Végül szeretném megköszönni tanáromnak Aszalós Lászlónak a tanácsait segítését

Összefoglaló

A program írása és tesztelései során kiderült, hogy az általam használt algoritmusok mellett jobban lehet használni a már kitöltött táblából való generálást, mint az üres táblából valót, ezért ezt a megoldást írtam bele a programomba.

Még sok algoritmus van, amit nem vizsgáltam meg alaposan és ezáltal nem is programoztam le, így a program könnyen gyorsítható lesz a továbbiakban több nem rekurzivitáson alapuló algoritmus használatával.

Sajnos nem sikerült elérnem a 17 elemmel kitöltött Sudoku táblának a generálását. Szóval ez még mindig ösztönöz arra, hogy rájöjjek a generálásának titkára, illetve hogy milyen lehetséges módszerrel fogom tudni megállapítani a 16 elemmel kitöltött helyes Sudoku feladvány létezését. Van aki azzal próbálkozott, hogy *Gordon Royle* első "17-es"-éből levett egy számot, és megkereste a megoldásait. 265244 megoldása lett. Ekkor letett arról a tervéről, hogy redukáltként keresi meg a 17-el kitöltött táblák közül a 16 elemmel kitölthetőt.

A program még koránt sincs befejezve és a későbbiekben tervezem továbbírását, illetve néhány helyen az átírását is. Kíváncsi leszek továbbiakban az NxM-es tábláknak generálására és a hozzájuk szükséges időre, illetve hogy milyen kihívást nyújt a későbbiekben olyan Sudoku variánsok írása a programhoz, amiket újságokban is látunk, úgy mint amikor több Sudoku táblát egy-egy tömbnél fogva összekapcsolunk.

Az én értékelésem alapján most létrejött egy Sudoku program, ami eddig is és a továbbiakban is jó lesz unalmas óráimnak eltöltésében. Bár sok hasonló program létezik, azért ez mégiscsak jobban tetszik, már csak azért is, mert ha valami nem tetszik benne azt könnyedén átírom és már jó lesz.

Irodalomjegyzék

Csizmazia Albert: Sudoku Utolsó látogatás:2009.05.04
http://progkor.inf.elte.hu/200506.2/BEAD/sudoku.htm

Csermi: Sudoku történelem Utolsó látogatás:2009.02.05
http://oldal.eu/sudoku/sustory.html

Forum: Sudoku egyedi feladvány (Fórum) Utolsó látogatás:2009.02.16
http://prog.hu/tudastar/90646-10/Sudoku+egyedi+feladvany.html

Terminology – Sudopedia (Sudokuval kapcsolatos írások gyűjteménye)
Utolsó látogatás:2009.05.11
http://www.sudopedia.org/wiki/Terminology

Posted by David: Sudoku generator Utolsó látogatás:2009.05.14
http://davidbau.com/archives/2006/09/04/sudoku_generator.html

Created by fuji Let's: Make Number Place (példa hozás papíron való generáláshoz)
Utolsó látogatás:2009.03.12
http://puzzle.gr.jp/show/English/LetsMakeNPElem/01

Más Sudoku irodalmak, melyek olyan algoritmusokat írnak le, amit nem használtam fel, de megvizsgáltam:

Document by: Jonathan Chu (Dr. Donald Knuth’s Dancing Links Algorithm)
Utolsó látogatás:2009.05.12
http://www.ocf.berkeley.edu/~jchu/publicportal/sudoku/sudoku.paper.html

Algorithmics of Sudoku
http://en.wikipedia.org/wiki/Algorithmics_of_sudoku.

Köszönetnyilvánítás

Köszönetet mondok Dr. Aszalós Lászlónak, hogy lehetőséget biztosított munkám sikeres elvégzéséhez és a diplomamunkám megírásához. Köszönöm segítőkész támogatását és dolgozatom bírálatát.