Szakdolgozat

Miskolci Egyetem

Sudoku táblákat generáló algoritmusok és webes játék fejlesztése

Készítette:

Nagy Dávid

Programtervező informatikus

Témavezető:

Dr. Agárdi Anita

Miskolc, 2024

Miskolci Egyetem
Gépészmérnöki és Informatikai Kar
Alkalmazott Matematikai Intézeti Tanszék **Szám:**

Szakdolgozat Feladat

Nagy Dávid (BQCMAU) programtervező informatikus jelölt részére.

A szakdolgozat tárgyköre: logikai összefüggések algoritmizálása, játékfejlesztés

A szakdolgozat címe: Sudoku táblákat generáló algoritmusok és webes játék fejlesztése

A feladat részletezése:

Sudoku játékok bemutatása, típusonkénti generáló programok megvalósítása és ezek alkalmazására webes felületen történő játék megalkotása. Az adatok készítésére Python programnyelvet használok, amelyeket weboldalon való megjeleneítéséhez HTML, JavaScript nyelvek alkalmazok.

Témavezető: Dr. Agárdi Anita, egyetemi adjunktus

A feladat kiadásának ideje: 2023.09.27.

.....
szakfelelős

Eredetiségi Nyilatkozat

Alulírott **Nagy Dávid**; Neptun-kód: `BQCMAU` a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős Programtervező informatikus szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy *Sudoku táblákat generáló algoritmusok és webes játék fejlesztése* című szakdolgozatom saját, önálló munkám; az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

- szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy plágium esetén szakdolgozatom visszautasításra kerül.

Miskolc, ... év ..... hó ..... nap

.....
Hallgató

1. A szakdolgozat feladat módosítása
   - szükséges (módosítás külön lapon)
   - nem szükséges

.....	.....
dátum	témavezető(k)

2. A feladat kidolgozását ellenőriztem:

témavezető (dátum, aláírás):	konzulens (dátum, aláírás):
.....	.....
.....	.....
.....	.....

3. A szakdolgozat beadható:

.....	.....
dátum	témavezető(k)

4. A szakdolgozat ..... szövegoldalt
   ..... program protokollt (listát, felhasználói leírást)
   ..... elektronikus adathordozót (részletezve)
   .....
   ..... egyéb mellékletet (részletezve)
   .....
tartalmaz.

.....	.....
dátum	témavezető(k)

5. A szakdolgozat bírálatra
   - bocsátható
   - nem bocsátható

A bíráló neve: .....

.....	.....
dátum	szakfelelős

6. A szakdolgozat osztályzata

   a témavezető javaslata: .....
   a bíráló javaslata: .....
   a szakdolgozat végleges eredménye: .....

Miskolc, ...
a Záróvizsga Bizottság Elnöke

Tartalomjegyzék

1. fejezet

Bevezetés

A játékok manapság mindenhol megjelennek a hétköznapokban valamilyen formában, legyen szó szórakozásról, tanulásról, vagy olykor munkáról. A pozitív élményeket idővel szeretnék hasznos időtöltéssel is összekapcsolni, ha már játszunk, vagy dolgozunk, akkor legalább tanuljunk is akár újat, akár meglévő tudást gyakorolva. Az oktató jellegű és a logikai gondolkodást ötvöző szórakozási lehetőségekre számos formában találhatunk példát. A Sudoku játék világszerte ismert. Eredeti nyomtatott változata újságokban, heti lapokban találhatjuk meg, többnyire keresztrényrejtvényekkel párban. A játék sokszínűsége mellett a nyelvfüggetlensége és egyszerű szabályai előnyben részesítik, hogy élvezhető és a legegyszerűbb formában is játszható legyen. Manapság már a megszokott újság, füzet forma mellett létezik fizikai táblaként is, hasonlóan a társasjátékokhoz, illetve, mondhatni természetesen, digitális megjelenése is hétköznapi. A különböző típusú táblák generálása, mind megoldása, tulajdonságok feltárása általánosan digitálisan történik, algoritmusok segítségével. Több eljárás is van, mind a táblák megfejtése, azok létrehozására, vagy az adat be- vagy kivitelének részére. Utóbbiak kivitelezésére az elmúlt években nagyobb ismeretségnek örvendő mesterséges intelligencia és gép tanulási algoritmusok, modellek is segítségre lehetnek, például képfeldolgozásban.

A Sudoku-ban megfogtak engem az említett gondolkodtató és egyben minimalista jegyek. Alkalmas a játék mind a logika gyakorlására, vagy a komplexebb rendszerek, vagy több lépésben megoldható problémák megfejtésére. Számos érv mellett teszem hozzá a megjelenésének sokszínű formáját.

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

1.1. ábra: 9x9-es sudoku tábla

Számomra a játékok játszása mellett mindig nagyobb hangsúlyt kapott az érdekeltség afelé, milyen működést és logikát is rejt a háttérben annak motorja. Könnyen esett a választásom arra, hogy ezeket megvalósítsam, vagyis a Sudoku játék hagyományos fajtájának generálása mellett egyéb típusokat kreáló kódokat, ezek egy részének vizuális formában megjelenítését és játszását tűzzem ki célnak.

2. fejezet

Ismertetés

2.1. Technológiák az adatgenerálás és webes játékokra

A kettő fő feladat teljesítéséhez ismertetem ezek funkcióját, és hogy milyen lehetőségeket tártam fel a megvalósítás érdekében.

A játék megvalósítása az alap verzió generálásától a weboldalon keresztül játszható formáig több lépésen megy keresztül.

A megválasztással szeretném elérni, hogy a használt kódok legyenek későbbi használatra is:

- könnyen átláthatóak, kezelhetőek
- hordozhatóak
- dokumentáltak
- egyszerűen tanulhatóak

Adatok megvalósítása

A mai használatos matematikai programozási nyelvek, szoftverek és csomagok tárháza igen széles körű. Ezek közül néhány, általam is körüljárt:

- MatLab (C, Java)
- Maple (C, Java)
- Google Colab (Jupyter Notebook service - Python, R)
- PyCharm (Python)

Ezen technológiákat sorra véve, kezdetben a **MatLab**[1] és a **Maple**[2] szoftverek az igen ismertek közé tartoznak. Használt programozási nyelveik a **C**[3] és a **Java**[4]. Mindezen érvek ellenére szól az is, hogy fizetős licenszhez kötöttek. Igaz, a 15 vagy 30 napos próba verzió elérhető hozzájuk, de ezen hátráltató ok miatt ezeket a lehetőségeket elvetettem.

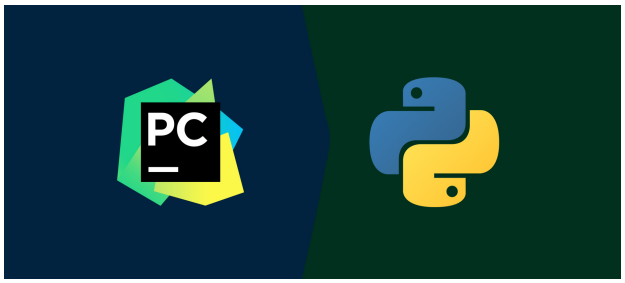
2.1. ábra: MatLab és Maple szoftverek

Google Colab

A Google Colab[5] számos előnnyel rendelkezik, amelyek miatt érdemes alkalmazni. Előnyei közé tartozik az hozzáférhetőség és komfortossága. Ezt azzal biztosítja a felhasználók számára, hogy lehetővé teszi a funkcióját, hogy közvetlenül a böngészőben dolgozzanak projektekkel bármiféle helyi telepítés nélkül. Emellett a Google Colab együttműködési platformot kínál, ahol több felhasználó egyszerre dolgozhat ugyanazon a projekten, javítva a csapatmunkát és a produktivitást. Számos lehetőséget kínál az adatelemzésre és adattisztításra. A **Python**[6] programozási nyelv egyszerű, könnyen olvasható kódok készíthetőek vele, rövidebb fejlesztési időt biztosít és a platformfüggetlen. A Colab segítségével az említett módon, szerveren keresztül való futtatási hátránya a lassúság. A virtuális eszköz hardver kapacitása véges, és korlátozott.

2.2. ábra: Google Colab + Python

A **Python** nyelvre számos mellette szóló érv miatt esett a választásom. A fejlesztői környezetre a **PyCharm IDE**-t[7] alkalmazom. Sok szerkesztési és nyomkövetési segítséget nyújt, lehetővé téve a forráskódok projektekbe szervezését, külön-külön futtatását a projekten belül, valamint a felhasznált Python csomagok feltérképezését és kódkiegészítő funkciók biztosítását. Elérhető Professional (fizetős) és Community (ingyenes) változatban is. Utóbbinak verziónak nincsen időkorlátja. A mellette szóló érvek miatt, a felsorolt és ismertetett szoftverek közül ezt az opciót választottam.



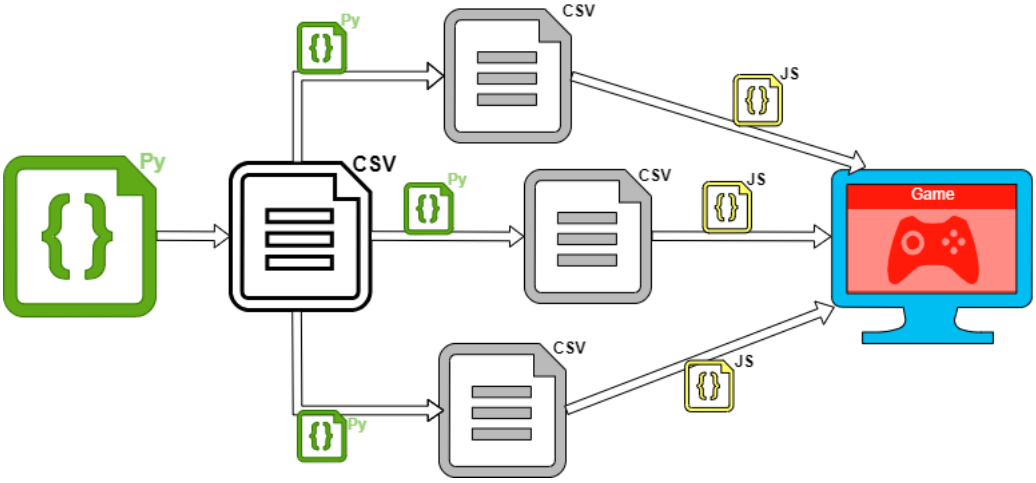
2.3. ábra: PyCharm + Python

Webes megjelenítés

Figyelembe véve, hogy a játék webes megjelenítése nem elsődleges szempontom, ezért a technológiák közül egy kezelhető, egyszerű megoldásra esett a választásom. A tartalomért **HTML**[8], a stílusért **CSS**[9] és az oldal működéséért **JavaScript**[10] nyelveket használok.

2.4. ábra: HTML, CSS és JavaScript logó

A végleges terv megvalósítási folyamatát az alábbi ábra ismerteti. Melyet részletesen a 3. fejezetben fejtek ki.



2.5. ábra: Terv a fájlok folyamatára

3. fejezet

Tervezés

Mindenekelőtt bemutatnám a játék alapszabályait. A sudoku egy logikai játék, melyben megadott szabályok szerint számjegyeket kell elhelyezni egy táblázatban. Legközönségesebb változatában egy 9x9-es táblázat van 3x3, azaz 9 darab 3 sorból és 3 oszlopból álló résztáblákra felosztva, és minden ilyen területet az 1, 2, 3, 4, 5, 6, 7, 8, 9 számcsoporttal kell kitölteni úgy, hogy az egész 9x9-es táblázat minden sorában és minden oszlopában az 1...9 számok mindegyike egyszer forduljon elő, más szóval egyediséget ad. A rejtvény készítője a megoldhatóság érdekében előre kitölti néhány szükséges számmal a 9x9-es táblázat bizonyos celláit – általában úgy, hogy csak egy megoldása létezzen a rejtvénynek. A rejtvényfejtő feladata kitölteni a maradék cellákat a fentebb írt feltételeknek megfelelően. A hagyományos sudoku esetében a különböző táblák száma $6,670,903,752,021,072,936,960$, ami $6.671 * 10^{21}$. [**11**]

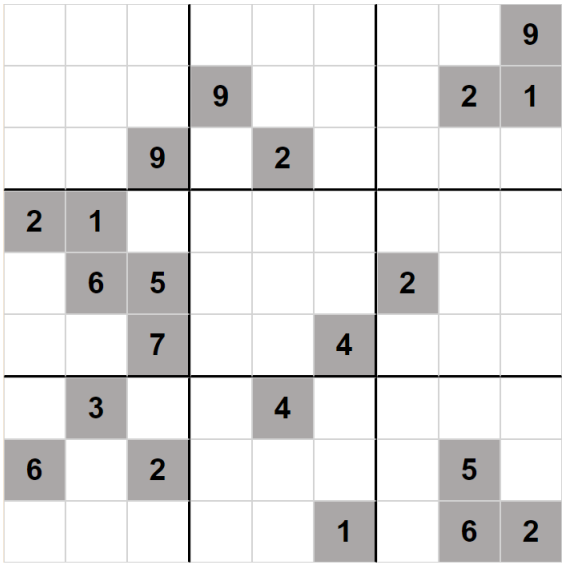
Nehézségi szempontból nézve, a tábla készítője a megfejtéshez előre megad bizonyos cellákat, amelyek a megoldást rejtik. A komplexitást növeli az, minél kevesebb ilyen szám van biztosítva a játékos számára. Ellenben ennek a mértéke nem lehet kisebb, mint 17, azaz ennyi minimális számú cella lehet előre megfejtve.[**12**] 16 vagy ettől alacsonyabb szám megadásnál a tábla megoldása nem garantált, hogy egyedi marad. A megoldást generáló eljárásomban alapvetően az üres táblából dolgozva, 1 sort, azaz 9 egymást követő elemet választok ki. Ez a szám bőven kisebb, mint az egyediséget biztosító mérték, ugyanakkor kínálja a kimenetelek több kombinációját. Például 1-9-ig feltöltve az 1. sort, több megoldást is eredményez megfejtés során.

A játék hasonlóan ismert, 4x4-es, 6x6-os és 16x16-os (mega) verziójában hasonlóak a szabályok az előzőhez, a használható számjegyeket a tábla neve is mutatja, amely szintén utal a tábla méretére is. A 6x6-os fajtában az részterületek mérete 2x3-as, vagy 3x2-es méretű lehet. A mega változatban, készítőtől függően, a 10-estől magasabb szintű számrendszerek számjegyeihez, itt is szokás alkalmazni a betűk bevezetését, ahogyan a 16-os számrendszerben történik.

A játék típusok között különbséget teszek azok között, amelyek egy alaptáblából is kivitelezhetőek, valamint azok között, amelyek csak megadott kritériumok szerint felelnek meg az adott változatnak. Az értékek közötti kapcsolatokat *tulajdonságok*nak nevezem a dolgozatban, amelyek különböző módon, minden esetben segítik a tábla megfejtését. A típusokat olyan sorrendben ismertetem, ahogy azok tulajdonságaik kategorikusan megállapíthatók. Elsőnek foglalkozom azokkal, amelyek cellánként függetlenek egymástól, majd, amelyek a szomszédos értékekkel kapcsolatosan jellemzőt adnak és végül azokkal a tábla verziókkal, amelyek formálisan is különböznek a megszokottaktól.

3.1. Hagyományos Sudoku

A legáltalánosabb típus, 9 sorból és ugyanennyi oszlopból áll. Táblának vagy mátrixnak is mondhatjuk. Méretét tekintve tartalmaz 9 darab 3x3-as részcsoportot, más szóval blokkot. A táblát 1-9-ig töltjük fel elemekkel olyan módon, hogy ezen értékek csakis egyszer forduljanak elő az adott sor-, oszlop- és blokkban.



3.1. ábra: 9x9-es Sudoku terv

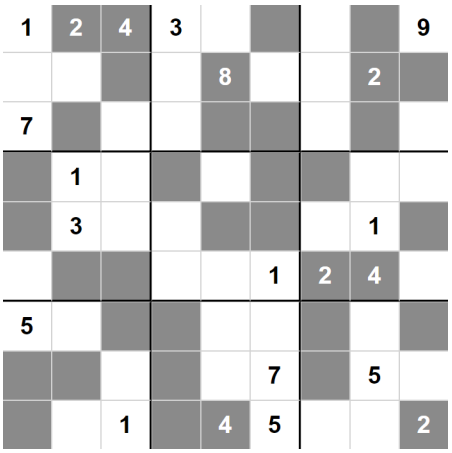
Nagyságától függően ezt a típusú táblát használja alapul a többi verzió is kiindulásként, így kivétel nélkül mindegyikre igazak a sudoku alapszabályai.

3.2. Egyéb Sudoku típusok

3.2.1. Önálló

Páros-Páratlan

A Páros-Páratlan nevű változatban az értékek valamilyen megkülönböztető jelölést kapnak, attól függően, hogy a megoldásban a cella páros vagy páratlan értéket tartalmaz. A jelölés lehet háttér vagy jelölőkeret színkülönbség is.



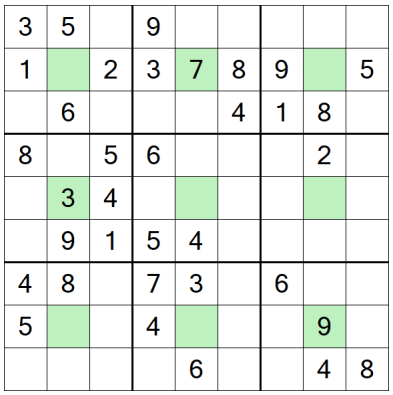
3.2. ábra: Páros-Páratlan Sudoku terv

3.2.2. Struktúrális

Az itt említésre kerülő Sudoku verziókat nem lehetséges bármilyen alaptáblára, mint tulajdonságot megállapítani, ez annak struktúrájától függ.

Közép-Pont-Sudoku

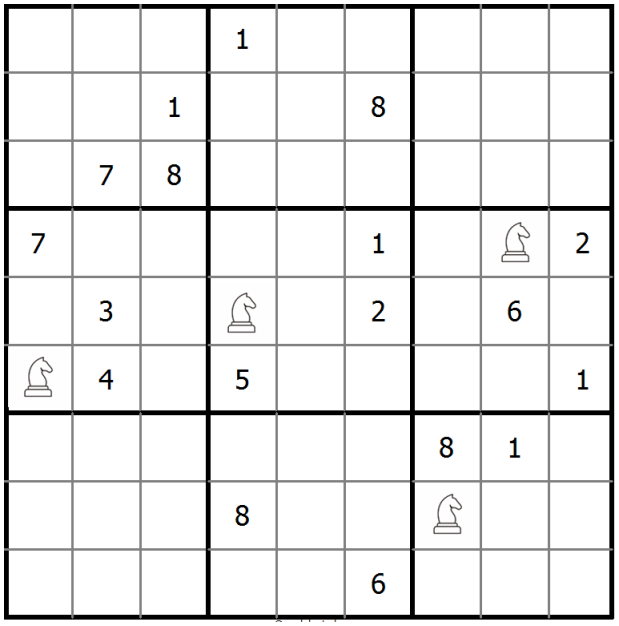
A neve a tábla 3x3-as blokkjaira utal. Az összesen 9 darab részterületeknek a középső celláiban tartalmazott értékeinek is egyedisége van a teljes tábla szintjén. Ebből adódóan a játék ilyen celláinak megkülönböztető háttérszínük van.



3.3. ábra: Közép-Pont-Sudoku terv

Huszár

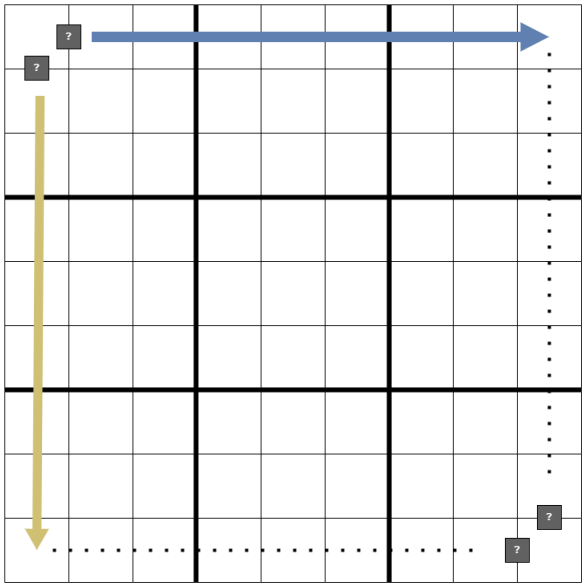
A Huszár verziójában a számcsoport egy kijelölt számjegye lesz az úgynevezett Huszár, amelyre a sakk ugyanilyen nevű bábúja lépésszabálya érvényes. A kijelölt számjegy a számcsoport legnagyobb, alap esetben 9-ese az. A ráadás szempont az, hogy nem állhat ugyanilyen számjegy ennek, a sakkjátékból ismert, L alakú lépéshatárába. Röviden leírva, 2 egység lépés valamely irányba, vízszintesen vagy függőlegesen, majd 1 egység lépés az ellenkező tengelyen.



3.4. ábra: Huszár Sudoku terv

3.2.3. Tulajdonságok

Általánosan a cellák, pontosabban a bennük tárolt értékek, közötti jellemzőkről és darabszámaikról azt érdemes tudni, hogy adott N méretű sor vagy oszlop esetén N-1 számú lesz. Észrevehető, hogy egy 9 elemű sor esetén 8 tulajdonság rejlik benne, de mind a 9 sor esetén, így a kapott mátrix 9*8-as dimenzióval rendelkezik. Függőleges vizsgálatnál a sor 9, az oszlopok száma 8, a mátrix következetességképp 8*9-es. A tulajdonságok generálásához általános iteratív módon szükséges megírni az algoritmust, hogy minden érték párra megtudja ezt állapítani az összes bemenet esetén. Az érték párok száma a 9x9-es táblában egy részterületre 12 darabot jelent, összesen így pedig 108 az egész táblára. A jellemzők megtalálásának szempontja, hogy 2 érték között történjen ez. A vizsgálat során a kiválasztott érték, és a rákövetkezőek vannak párba állítva a feltétel ellenőrzéséhez, amely sor, vagy oszlop szintjén történik. Egy szempontú tulajdonságot használó táblák esetén egy vízszintes és egy függőleges halmaz kerül ki külön.

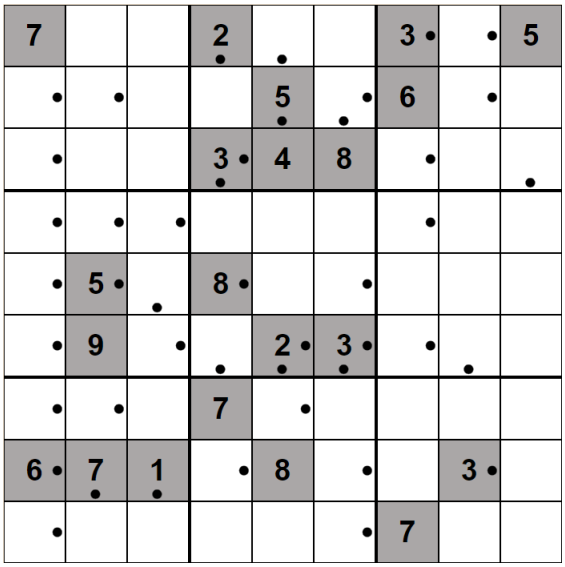


3.5. ábra: Egy szempontú tulajdonság szekvenciálisan

Szomszédok

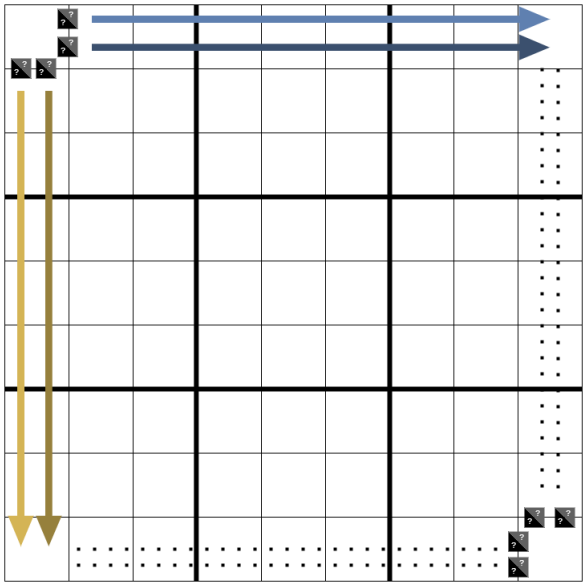
A szomszédok nevű verzió bemutatja az előbb megismert tulajdonságú táblákat.

Azon szomszédos értékek között van fekete kör jelölés, amely mutatja, hogy a két érték különbsége 1-et ad. Itt a részcsoportok között is találhatunk a tulajdonságra utaló jelölést.



3.6. ábra: Szomszédok Sudoku terv

Több jellemzővel rendelkezőek helyzetében szétválasztva a szempontok szerint kerülnek ki tengelyek alapján párban.

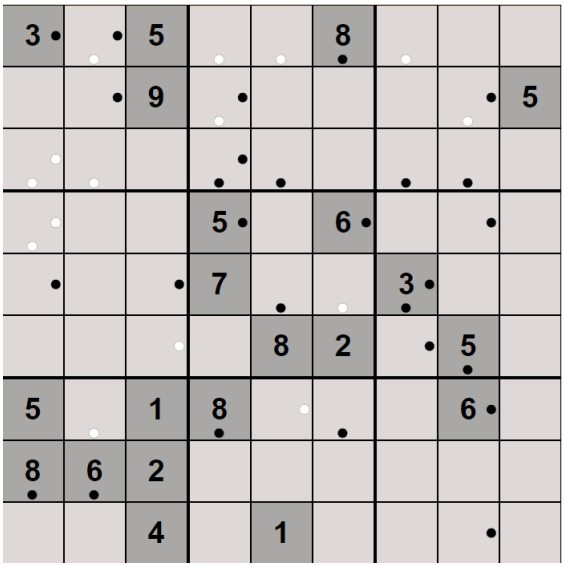


3.7. ábra: Kettő szempontú tulajdonság szekvenciálisan

Kropki

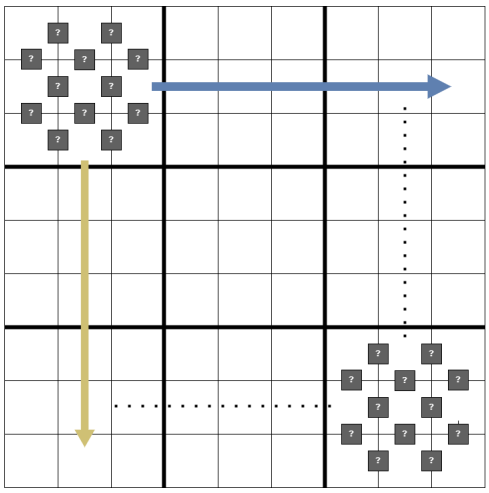
Kombinált típus, egyszerre kettő jellemzőt tartalmaz a tábla. Ezért is adja vissza könnyedén a kettős szempontú szekvenciális keresést.

Az első, a Szomszédos típusban megismertet jelöli. A másik tulajdonság alapján a szomszédos cellák egymás kétszeresét vagy, fordított szempontból, felét jellemzi. A jelölést érdemes megkülönböztető módon biztosítani, hogy a játékos kevés ismerettel is megfelelően tudja alkalmazni itt. A bemutatásom sorrendje miatt a Szomszédos szimbóluma fekete kör, és az újat biztosítja a fehér kör.



3.8. ábra: Kropki Sudoku terv

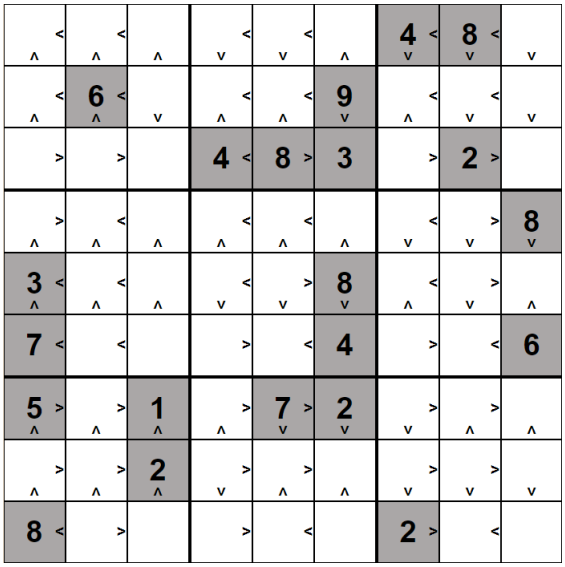
Kivételes esetben, a keresést és később a generálást végezhetjük az említett sorrendben, kihagyva a blokkok közötti vizsgálatot, vagy másképp mondva, a blokkok határain fekvő, más blokkal szomszédos cellák közötti kapcsolatot. Az utóbbi eljárást alkalmazva egy 3x3-as csoport esetén 6 függőleges és 6 darab vízszintes tulajdonságot kapunk, a teljes táblánál ez összesen 9*12.



3.9. ábra: Egy szempontú tulajdonság blokk szinten

Relációs

A Relációs sudoku játékban az értékek közötti tulajdonság megmutatja melyik szám a nagyobb a részerületben a megoldáshoz. Alkalmazza a részeterületekben történő keresést.



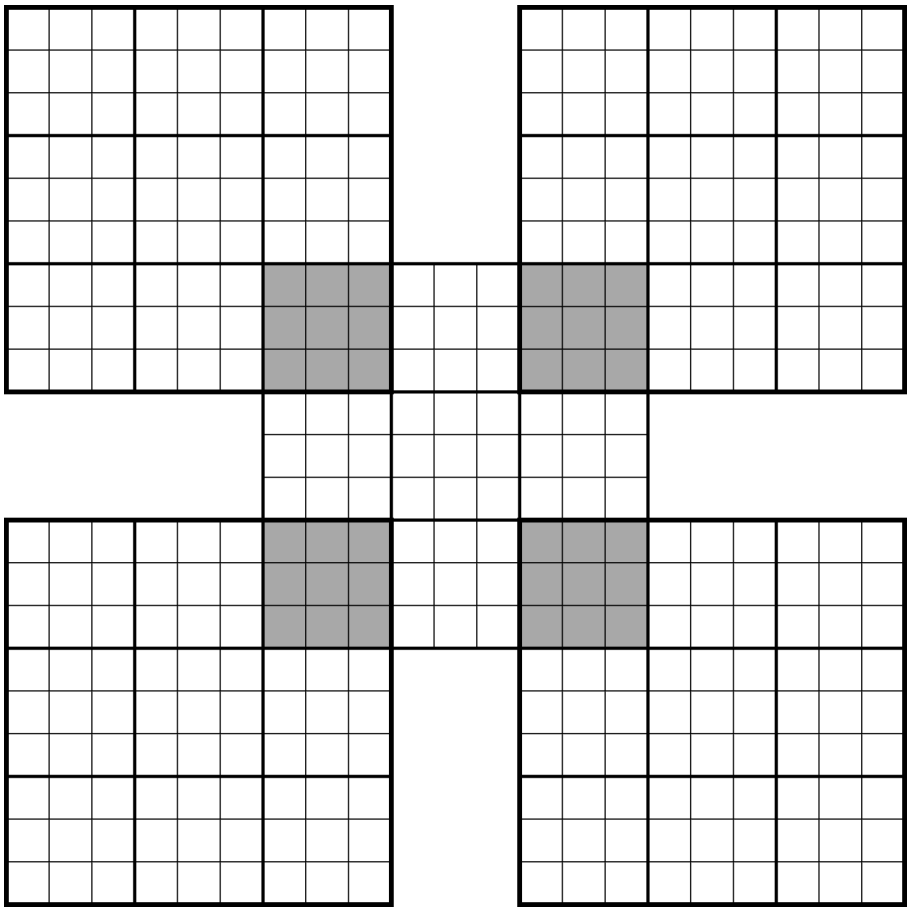
3.10. ábra: Relációs Sudoku terv

Fontos, hogy ez a típus a részterületek közötti tulajdonságra nem tér ki. A JavaScript kódban tekintettel kell erre lenni a megvalósítás során. A tárolt adatok hossza kettő jellemzőnél 288, egy jellemzőnél 144, ennek redukált változatánál 108.

3.2.4. Formabontás

Szamuráj

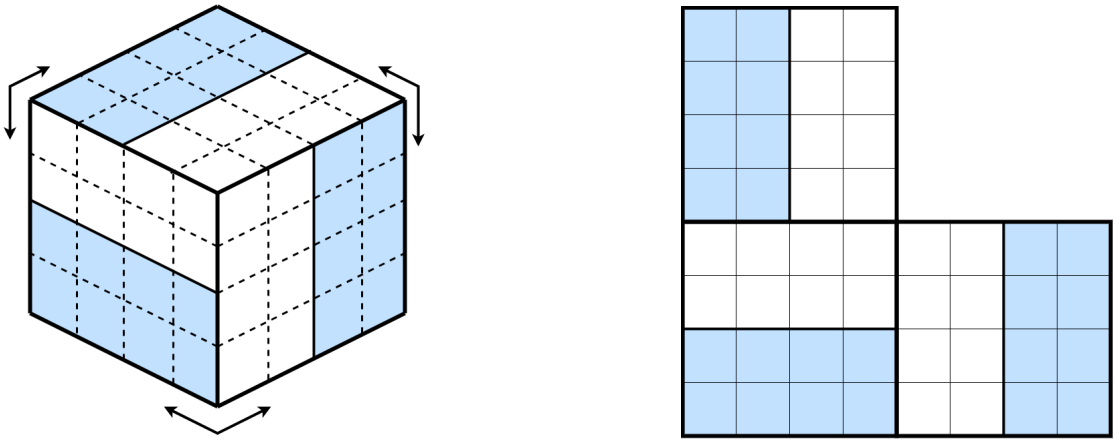
A Szamuráj játékban általában 5 sudoku tábla van összefűzve. Az összekapcsolás módja hasonló a halmazok metszetére, itt a táblák egy részcsoportja illik össze egy másik tábláéval, amellyel annak megoldása teljesen megegyezik. Jellemzően középen található egy tábla, amelynek a 4 sarok blokkja van metszésben egy-egy újabb, azonos méretű táblával. Ezen módon kialakult mátrixok innentől rendelkeznek 21 (9+3+9) cellahosszúságú sorokkal, és/vagy oszlopokkal. Ha ezeket a sorokat és oszlopokat tekintjük, akkor itt nem érvényesek az értékek egyszeri előfordulású szabályai, emiatt ebben az esetben az alap méretű táblát tekintjük ezek környezetének. A Szamuráj játékban a nehézségi szintet fokozottan növeli, hogy egyszerre több táblát is meg kell fejteni a teljes megoldáshoz.



3.11. ábra: Szamuráj Sudoku

Kocka

Több táblát alkalmazó játék. A térbeli kockát a játékban - egyelőre csak - síkban ábrázolva 3 oldalát lehetséges kihasználni, amelyek 1 csúcsban érintkeznek. Az így kialakuló szomszédos élek mentén is érvényesül a táblák között a számsorok egyedisége. A kocka érintett oldalait vizsgálati szempontból kiterítve egy L alakot kaphatunk.



3.12. ábra: Egész és kiterített Kocka Sudoku

Amorf (Jigsaw)

Az Amorf, vagy Jigsaw (puzzle) változatban a korábban megszokott szabályos részterületekkel ellentétben amorf, egybefüggő alakzatok zárják a tábla számcsoportjait. Ezekre bizonyos esetekben igaz, hogy a kapott tábla, részcsoportjainak alakját tekintve középpontos-, vízszintes- vagy függőleges szerinti szimmetriát figyelhetünk meg. Az alakzatok kinézete hasonlíthat a kirakós vagy puzzle játék darabjaira, innen is utal a neve rá.

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

3.13. ábra: Amorf Sudoku

Killer-Sudoku

A Killer-Sudoku részben alkalmazza az Amorf típust, itt a részterületek nem a teljes számcsoportot fedik le, ugyanakkor az azokat tartalmazó számok összege van feltüntetve a terület első bal felső sarkában. Az említett kisebb területekben, vagy más szóval ketrecekben, található minimum 1, de maximum 9 számjegy is. Az így alkotott ketreceket tekintve az alap 1 nagyságú méret mellett a többi 8 méretben található számokból alkothatott összes kombinációk száma a megfejtéshez 120. Az összegek legkisebb mérete 3 (1+2), a legnagyobbé 45 (minden szám a csoportban).[**13**]

16			9	11		20	5	
9	5	15		12			8	
			30	5	8		12	
13						3		13
12	8			8		9		
			9	9	15	20		
13	20					6		19
	12		9	14	8	7		
							13	

3.14. ábra: Killer Sudoku

4. fejezet

Megvalósítás

4.1. Generáló algoritmusok

Kezdetben *Python* nyelven egy 9x9-es sudoku tábla megoldása történik. Az alap ötlet egy sudoku megoldó programkódból indult. [**14**] Az eljárásban egy megoldatlan sudoku tábla értékei vannak eltárolva. Az algoritmus üres mezőket keres, a tábla bal felső sarkát az első elemnek választva halad jobbra, majd azzal végezve, lép lefelé új sorba téve ugyanezt. A talált üres helyre a játékszabályoknak megfelelően sikeresen tesz bele egy számot a legkisebbtől a legnagyobbig. A folyamatot addig végzi, amíg talál üres mezőt a táblában, közben rekurzív módon hívja meg önmagát. Ezen lépés azért előnyös, mert sikertelenség esetén tábla visszafejthető olyan korábbi szintre, ahol még újabb próbálkozásokat tehet a megoldás elérése érdekében.

Programkód 4.1. Alaptábla generálás

```
def solve(board, iterations):
        find = find_empty(board)
        isSolve = False
        if not find:
                return True, iterations
        else:
                row, column = find
        for i in range(1, 10):
                # Increase iterations if a number is tried
                iterations += 1
                if valid(board, i, (row, column)):
                        board[row][column] = i
                        # Recursive call to solve the rest of the board
                        isSolve, iterations = solve(board, iterations)
                        if isSolve:
                                return True, iterations
                        board[row][column] = 0
        return False, iterations
```

A tovább alakított változatban tekintettel voltam arra, hogy ne egy, előre meghatározott, részben megfejtett mátrixszal dolgozzon az eljárás, így ennek saját generálását valósítottam meg. Egy kezdetben üres, vagyis 0 értékekkel feltöltött, meghatározott méretű táblát kapunk. A *Python* mátrix típusa valójában nem egy különálló típus, hanem olyan lista, amelynek az elemei is listák. Ettől fogva könnyebbnek találtam ennek kezelését soronként megtenni. A 9 darab sor közül kiválaszt 1-et véletlenszerűen, ezt a listát felölti 1-9-ig sorban elemekkel, majd összekeveri a *random.shuffle()* függvény segítségével. A korábban felsorolt megoldási lépéseket a tábla maradék 8 listáján végzi el, hogy visszatérhessen annak teljesen megfejtett változatával.

Programkód 4.2. Tábla kiürítése és inicializálása

```
def main(iterations = 0):
        # Fill the 9x9 board with zeros with comprehension method
        board = [[0 for i in range(9)] for j in range(9)]
        # choose random row index
        choosen_row_index = random.randrange(9)
        # fill the choosen the row with elements from 1 to 9
        board[choosen_row_index] = [i for i in range(1, 10)]
        # Shuffle the previous row
        random.shuffle(board[choosen_row_index])
        # Start time from this point
        start_time = t.time()
        # Call the solver function
        # with current board and with intialized iterations
        _, iterations = solve(board, iterations)
        ex_time =  t.time() - start_time
        return board, ex_time, iterations
```

Az eljárásban a kiválasztott sor összekeverése elhanyagolható, ezzel viszont teljes futtatás esetén az összes létező 9x9-es sudoku megfejtett táblák értékeinek növekvő sorrendjét kapjuk. Ha ezt a halmazt az elejéről kezdjük vizsgálni, feltűnhet, hogy a legtöbb kapott adatsor az 1-2-3-4-5-6-7-8-9 sorral kezdődik. Ennek oka, hogy a program a megfejtéshez a legkisebb értéktől a legnagyobbig tölti fel, és emellett vizsgálja meg az értékek szabályosságát.

Az alaptáblák általános generálására az említett megoldó algoritmust használtam fel egy külön fájlban. Az itt kapott adatokat egy CSV fájlba mentettem ki. A CSV fájl típus, *Comma-Separated-Values*[**15**] nevéből adódóan, az értékek vesszővel vannak elválasztva. Erre azért esett a választásom, mert a játéktábla számsorai mellett van mód egyszerre más értékeket is tárolni szükség esetén. Tesztelés formájában megtettem ezt az iterációk számával és a generálási idővel is. Minden előkészítéskor egy *dictionary* változót készítek, amelyben kulcs-érték párok formájában tudom a megfelelő adatokat eltárolni. A fájlba mentés előtt a kapott adatok egy konvertáláson esnek át. A mátrix értékeit egyben kezelem, ezért annak elemeit először string-gé füzöm össze, majd egész számtípussá konvertálom. A kimentés előtt mindezt egy kétdimenziós *Dataframe*-mé alakítom át, hogy címkézett adatszerkezetet kapjak, ezzel felgyorsítva a kezelési móddal a futási időt más adatszerkezetekkel szemben. Fontos, hogy a CSV fájlba csak egyedi táblaértékek kerülnek.

Teszt jelleggel készítettem különböző rekordszámú fájlokat, amelyek, 300, 5000, 10000 és 25000-ig terjedően fordultak elő. Ennek célja, hogy a változatos játék típusok algoritmusai nem minden esetben követelnek meg azonos mennyiségű sort, ezzel gyorsítva a futási időt. Például, ha nem igényel egyszerre több táblát, amelyek összevont alakban teljesítik a játék létrehozását, vagyis 3 vagy 5-t egyszerre, mint egy Samurai vagy Kocka játékban.

Programkód 4.3. Táblák tárolása

```
BOARD = pd.read_csv(csv_file_name)
# If the current values are not unique, it breaks
if not BOARD["values"].is_unique:
        return
while True:
        with open(csv_file_name) as f:
                SB = f.read()
        # Dictionary to store datas of a generated board
        data = {
                "values": None
        }
        # Call the board generator function from sudoku_solve.py file
        board, time, iterations = SS.main()
        numbers = ""
        # Store board's elements in a string
        for row in board:
                for item in row:
                        numbers += str(item)
        data["values"] = int(numbers)

        df = pd.DataFrame(data, index=[0])
        if numbers not in SB:
                df.to_csv(csv_file_name, mode="a",
                        index=False, header=False)
        else:
                print(f"This board is already in
                                the CSV file: {numbers}")
```

4.2. Generálások általános lépései

Minden típus esetében az alap sudoku táblákat tároló CSV fájl beolvasásra kerül. Egy mátrixban tárolja az összes sort, majd a vizsgálathoz a kiválasztottat átmenetileg egy önálló mátrix formátummá konvertálja. A tulajdonságok listáját vagy listáit inicializálja. A keresés végén a listákat átadja a *DataFrame* típusú változónak, hogy azon keresztül az új adatokkal biztosított fájl kerüljön mentésre.

A mátrix formává való konvertálás saját függvényben valósul meg egy külön modulban. Ebben string formátumból alakítja ki a listán belüli lista típust, amellyel visszatér. Az ilyen típusban átláthatóbb mind az elemek használata, különösen, ha egymáshoz viszonyítottan tesszük ezt, mind a teljes értékhalmazra vonatkozó hibakeresés.

Programkód 4.4. Generálás incializálása

```
import string_to_int_matrix as converter
import pandas as pd

all_table_values = []
tables_matrix = []

csv_file_name = "sud_boards_SZD.csv"

BOARD = pd.read_csv(csv_file_name)
# Store the values for each row
for value_row in BOARD["values"][:]:
        all_table_values.append(value_row)
. . .
for table_row in range(len(all_table_values)):
        # Convert table value string to matrix
        tables_matrix.append(
                converter.convert_string_to_int_matrix(
                        all_table_values[table_row] ) )
        current_board = tables_matrix[table_row]
        . . .
        # Loop for the matrix's elements
        for row in range(len(current_board)):
                for column in range(len(current_board[row])):
                        . . .
```

A tulajdonságokat mutató értékek a mátrixban *numerikus*, *szimbólum* vagy *karakter* formában kapnak helyet.

A kódban a *Pandas* és egy saját, "`string_to_int_matrix`" nevet kapott package kerül használatra. A *Pandas* package-ből kapott *DataFrame* adatszerkezete lehetővé teszi nagy mennyiségű strukturált adat (pl. CSV, Excel, SQL adatbázis) hatékony kezelését és tisztítását. Könnyen lehet szűrni, rendezni, csoportosítani és átalakítani az adatokat. A "`string_to_int_matrix`" fájl a kapott paraméter alapján string típusból készít integer mátrixot, amivel visszatér adatként.

Programkód 4.5. Integer mátrix konvertáló függvény

```
def convert_string_to_int_matrix(_value):
        . . .
        for i in range(len(board)):
                board[i] = [*board[i]]
        for r in range(len(board)):
                for c in range(len(board[0])):
                        board[r][c] = int(board[r][c])
        return board
```

Zárásként minden tábla és adott esetben a hozzájuk tartozó tulajdonságok eltárolásra jutnak. Típusonként szeparált módon tárolódnak fájlokban. Ehhez előre definiálom azok nevét, az algoritmus végeztével pedig oszlopokként belekerülnek a véglegesbe, ahonnan exportálódnak.

Programkód 4.6. Adat exportálás

```
new_csv_file_name = "new_example_sud_boardsSZD.csv"
 . . .
 # Update the DataFrame with the new columns
NEW_BOARD["new_column"] = datas
# Save the updated DataFrame to a new CSV file
NEW_BOARD.to_csv(new_csv_file_name, index=False)
```

4.3. Játék típusok generálása

4.3.1. Közép-Pont Sudoku

Struktúrális megállapítású típus, emiatt csak a megfelelő szerkezetű táblákat tárolja el a program. Kezdetben felvesz egy előre definiált 1-9-ig terjedő mintát, és a 3x3-as részcsoportok közepén keresi ezeket az értékeket. Az ellenőrzés végén, ha elsőnek megegyezik a hossza, majd a rendezett formában a kész mintával is egyenlőséget tesz, akkor eltárolásra kerül.

Programkód 4.7. Közép-Pont Sudoku-t generálás

```
PATTERN = "123456789"

. . .
center_dot_str = ""
. . .
# Condition for the properties in horizontal
if (row % 3 == 1) and (column % 3 == 1):
    if str(current_board[row][column]) not in center_dot_str:
        center_dot_str += str(current_board[row][column])
if len(center_dot_str) == len(PATTERN):
    if sorted(center_dot_str) == list(PATTERN):
        center_dot_numbers.append(all_table_values[table_row])
```

4.3.2. Páros-Páratlan

Az egyik legegyszerűbb algoritmus, hiszen egy tulajdonságú, és szekvenciális keresésű, mindemellett önálló értékekről történnek a megállapítások. Azért mondható egy tulajdonságúnak, mert valójában a jellemzők nem fedik egymást, - egy szám csak páros, vagy csak páratlan lehet – ráadás előny, hogy 1 sorban is lehet ezeket tárolni.

Programkód 4.8. Páros-Páratlan generálás

```
even_odd_str = ""
. . .
# Condition for the properties in horizontal
if current_board[row][column] % 2 == 0:
    even_odd_str += "e"
else:
    even_odd_str += "o"
even_odd_numbers.append(even_odd_str)
```

4.3.3. Szomszédos

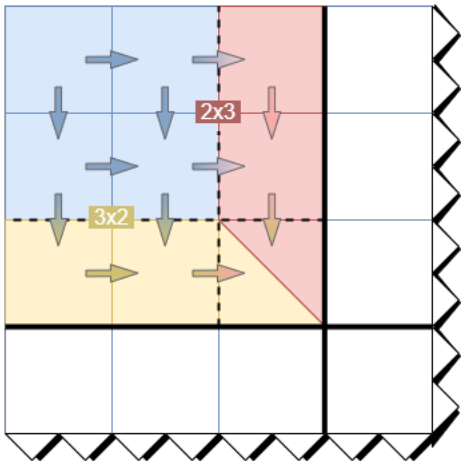
A jellemző akkor adott, ha kiválasztott elem, és az őt követőnek a különbsége 1. Az egyszerűség kedvéért a sorrendet nem tekintem, mely érték a nagyobb, hanem a várt differencia abszolút értéket veszem. Találat esetén 1, másképp 0 kerül a mátrixba.

Programkód 4.9. Szomszédos generálás

```
# Condition for the properties in horizontal
# Check the properties before it reach last element
if column != len(current_board[row])-1:
  # If the subtraction of the neighbours values is equal to 1
  if abs(current_board[row][column] - current_board[row][column+1]) == 1:
    horizontal_str += "1"
  else:
    horizontal_str += "0"
# Condition for the properties in vertical
if row != len(current_board)-1:
  if abs(current_board[row][column] - current_board[row+1][column]) == 1:
    vertical_str += "1"
  else:
    vertical_str += "0"
```

4.3.4. Relációs

Korábban említett módon, itt a relációk csak a blokkokon belül kapnak szerepet, azaz nem szekvenciális. A keresési szempont alapján az elem és az őt követő, vagy az egy sorral alatta megtalálható elem értéke ad relációt, melyik a nagyobb. A blokkok közötti kapcsolat keresésének elkerülése érdekében egy plusz feltétel épül a pozíciók figyelésébe. A részcsoportok 3x3-as méretét 2x3 (kék és piros színű terület) és 3x2-esre (kék és sárga színű terület) kell csökkenteni, ezt annak a bal felső sarkát választva kezdőhelynek (kék terület).



4.1. ábra: Blokk szintű tulajdonság a Sudoku tábla bal felső területén

Az ebben a mátrixban megtalálható értékeket fő értékeknek nevezem. A feltétel a következővel egészül ki, a fő értékek pozíciójának oszlop, később sor indexét 3-mal való osztás esetén 0 vagy 1 maradékot kell adnia, hogy vizsgálható legyen.

Programkód 4.10. Relációs generálás

```
# Checks properties only in the box with the column
  if column != len(current_board) - 1
        and (column % 3 == 0 or column % 3 == 1):
    # If the next element is greater
    if current_board[row][column] < current_board[row][column + 1]:
      horizontal_str += "<"
    else:
      horizontal_str += ">"
```

4.3.5. Kropki

Két tulajdonságú, szekvenciális keresést alkalmaz. Az átláthatóság érdekében szeparált formában, a 2 jellemzőt és a 2 irányban történő keresést 4 egymást követő ciklusban írtam meg. A játék típus sajátossága miatt a 2 operált érték megállapítására, hogy egymás kétszeresei-e, azt a módszert alkalmaztam, amelyben megállapítom melyik szám a nagyobb, majd az osztásban ezt választottam osztandónak. Ha hányadosnak a 2.0 float értéket kaptam, azaz maradék nélkül 2 egészet, akkor a tulajdonság string-be 2-es érték került, más esetben 0. Erre a megállapításra más módszer is használható, amelyben az osztás eredményére *vagy* operátorral teszek vizsgálatot, hogy 0.5 vagy 2.0 eredményt kap. Ebben az esetben nem számít a számok sorrendje.

Programkód 4.11. Kropki generálás

```
# Condition for the doubles properties in horizontal
greater = None
less = None
# Check the properties before it reach last element
if column != len(current_board[row])-1:
  # Stores which neighbour is greater
  if current_board[row][column] < current_board[row][column+1]:
   greater = current_board[row][column+1]
   less = current_board[row][column]
  else:
   greater = current_board[row][column]
   less = current_board[row][column+1]
  # If the division is 2.0, it found a properties for double
  if greater / less == 2.0:
   doubles_horizontal_str += "2"
  else:
   doubles_horizontal_str += "0"
```

A Kropki típusban az 1 és 2 értékek szomszédságánál mind a kettő tulajdonság teljesül. Mivel a Szomszédok típushoz képest ez egy összetettebb játék, így ilyenkor a kétszeresség re utaló jel vonatkozik rájuk. Fontos kiemelni, hogy ennek okán az utolsó 2 feltétel, amelyben az ilyen jellemzőket keresem, szükséges beleépíteni a feltételbe, hogy az említett számok esetén ezt ne ellenőrizze. Az összeadás kommutativitása miatt az 1 és a 2 számok összegét építettem a feltételbe, ezek kizárása érdekében.

Programkód 4.12. Kropki szomszédos generálás

```
# Current element and its neighbour's –in next column– additions isn't 3
if abs(current_board[row][column] - current_board[row][column+1]) == 1
    and (current_board[row][column] + current_board[row][column+1] != 3):
    neigh_horizontal_str += "1"
else:
    neigh_horizontal_str += "0"
```

4.3.6. Szamuráj

A Tervezés fejezetben megfogalmazott módon a Szamuráj játék kivitelezése összetettebb. Emiatt ennek a verziónak csak *Python*-ban írt generáló algoritmusa van a dolgozatomban megalkotva.

Tömören megfogalmazva szükség lesz 5 darab Sudoku táblára, amelyeknek a 4 sarkában helyet foglaló 3x3-as részcsoportjaik megegyeznek az ismert kritériumok alapján. A játékmezőkre eredeti, string formájukban is szükség van teljes egyezés esetén, így egy ilyen típusra visszakonvertáló fájl is meghívásra kerül. Egyszerű formában, korábbi package fordított nevét kapta "`int_matrix_to_string`". Tehát a táblák blokkjaira van szükség ebben a feladatban, ennélfogva importálásra kerül a saját "`get_board_box`" fájl/package. Meghatározott pozíció alapján visszaadja a 3x3-as részcsoport elemeit mátrix alakban.

Programkód 4.13. Pozíció szerinti mátrix generálás a blokkra

```
def get_board_box_values(position, board):
  box_x = position[1] // 3
  box_y = position[0] // 3
  box_list = []
  for i in range(box_y * 3, box_y * 3 + 3):
    for j in range(box_x * 3, box_x * 3 + 3):
      box_list.append(board[i][j])
  return box_list
```

Kezdetben elvárt egy alaptábla, amely középen helyezkedik el az öt közül. Ehhez viszonyítottan kell találni olyan táblákat, amelyek jobb alsó blokkjának 9 értéke megegyezik a kezdeti tábla bal felső blokkjával. Találat esetén keres szintén az alaptábla soron következő, jobb felső sarkához is egy táblát, amelynek már annak bal alsó sarokban lévő 9 számjegye egyezik ezzel. Az eljárás folytatólagosan a kifejtett szabályok alapján, akár sorrendet bontva, végzi a vizsgálatot.

Induláskor a részterületek pozíciói (a 4 sarok) kerülnek eltárolásra, majd érték nélkül létrejönnek a táblák számára a változók.

Programkód 4.14. Blokk pozíciók és tábla változók

```
top_left_box_pos = [0, 0]
top_right_box_pos = [0, 8]
bottom_left_box_pos = [8, 0]
bottom_right_box_pos = [8, 8]

base_board = None
top_left_board = None
top_right_board = None
bottom_left_board = None
bottom_right_board = None
```

A kezdőtábla 4 sarka *minta* név alatt külön-külön változókba generálódik le a *getBox()* függvénnyel a pozíciókat átadva.

Programkód 4.15. A kezdő sarok minták eltárolása

```
for current_base in all_table_values:
  value = current_base
  base_boards = to_int_converter.convert_string_to_int_matrix(value)
  # Top left box of the base board
  top_left_pattern_box = getBox.get_board_box_values(
                         top_left_box_pos, base_boards)
  # Top right box of the base board
  top_right_pattern_box = getBox.get_board_box_values(
                         top_right_box_pos, base_boards)
  # Bottom left box of the base board
  bottom_left_pattern_box = getBox.get_board_box_values(
                         bottom_left_box_pos, base_boards)
  # Bottom right box of the base board
  bottom_right_pattern_box = getBox.get_board_box_values(
                         bottom_right_box_pos, base_boards)
```

A blokkok készen állnak a kereséshez. A teljes adathalmazon, azaz az összes táblán végez keresést a nagyobb siker érdekében. Sarkonként történik a folyamat, egyenként külön ciklussal. Az általam meghatározott sorrend a következő:

- **Bal felső**
- **Jobb felső**
- **Bal alsó**
- **Jobb alsó**

Találat esetén a vizsgált táblát tárolja el, majd a sarokra vonatkozó ciklus leáll, és folytatja a következővel. Mindez addig folytatódik, amíg a 4 helyen nem kap eredményt, vagy a táblák elfogynak.

Programkód 4.16. A kezdő sarok minták eltárolása

```
# Search for the matching top right base box
# to bottom left box of the other boards
top_left_board = None
top_right_board = None
bottom_left_board = None
bottom_right_board = None

for current_board in tables_dictionary.values():
 bottom_right_matching_box = getBox.get_board_box_values(
         bottom_right_box_pos, current_board)
 if top_left_pattern_box == bottom_right_matching_box:
 # Get the index/values of the board
   top_left_board = current_board
   print("I found top left!")
   break
 for current_board in tables_dictionary.values():
 bottom_left_matching_box = getBox.get_board_box_values(
         bottom_left_box_pos, current_board)
  if top_right_pattern_box == bottom_left_matching_box:
  # Get the index/values of the board
    top_right_board = current_board
    print("I found top right!")
    break
 . . .
 if top_left_board and top_right_board
       and bottom_left_board and bottom_right_board:
   base_board = current_base
   break
```

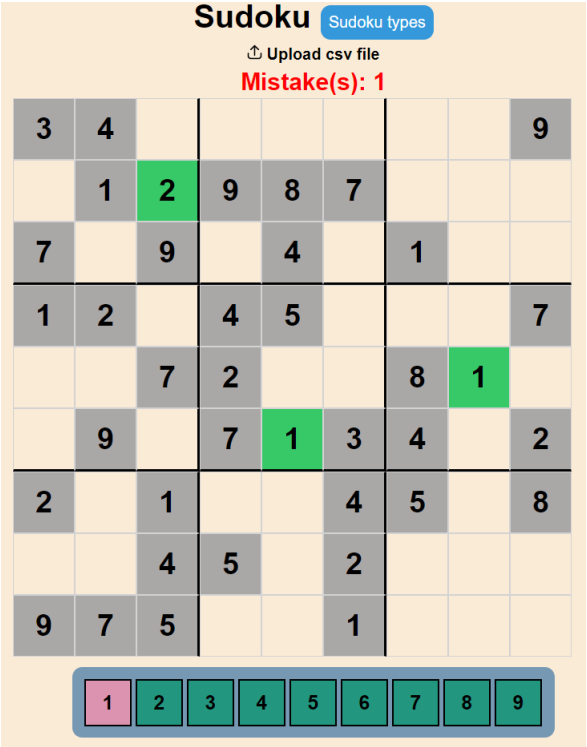
A keresés végén, ha mind az 5 tábla változó tartalmaz értéket, akkor sikeresen talált egy Sudoku tábla kombinációt az adathalmazban. Végül ezek eltárolásra kerülnek a CSV fájlba.

A Szamuráj Sudoku 5 Sudoku játékmező kombinációját használja. Ennek megtalálására tehát sokkal kisebb a valószínűség, mint egy egyszerűbb, Közép-Pont változatú Sudoku-ban. Érdemes ezért ennél nagy adathalmazt használni.

A fejezet végén, a játék kézi generálásához, hogy ne táblák közötti összefüggést és hasonlóságot kelljen megfigyelni, egy részben ehhez kapcsolódó algoritmust taglalok. A Szamuráj játék készítésének sikeresebb eredménye érdekében a kezdő tábla sarkait eltárolva generálható azokból az értékekből 4 új tábla. Ezért biztosan egyeznek az említett helyeken a 9x9-es mezők.

4.4. Webes megvalósítás

A sudoku játék játszására készítek egy weboldalt, amely a korábban megírt kódokból generált adatokat használja fel a táblák megjelenítésére. Taglalni fogom általánosan az oldal kódját, és a játék típusokhoz egyedi kódrészt. Mivel elsősorban a különböző sudoku játékok generálási algoritmusai voltak a célom, így a webes formában nem minden típus került megvalósításra.



4.2. ábra: Webes sudoku

4.4.1. A weboldal szerkezete

A fő váznak a tartalma HTML, a stílusa CSS, a működése JavaScript kóddal került megírásra. Az oldalon található egy cím, a játék típus nevét visszaadva, egy menü, amelyből válthat a játékos más aloldalakra, ahonnan a többi változat érhető el. A játékhoz szükséges elemek a feltöltés gomb, amellyel egy csv fájl-t vár az oldal feltöltésre, verziótól függően. Majd a hibák számát visszajelző sor. A játékmező alatt a használható számjegyek palettája van, amelyből a kívánt értéket kiválasztva lehet az üres cellákba próbálni. Minden játék fajtának saját HTML és JavaScript kódja van. Strukturálisan kevés dologban térnek el, többnyire a szkriptben van különbség, amely a megjelenített táblák tartalmát generálja a feltöltött adatokból. Igyekeztem a stílust minden oldalhoz és HTML elemhez egy fájlban megvalósítani. Emiatt vannak megjelenítést szolgáló class-ok, amelyek csak megadott típusban kerülnek az elemekre. Például a Relációs játékban alkalmazott jellemzők jelölése kisebb betűméretet használ, szemben a Szomszédos vagy Kropki, fehér vagy fekete kör szimbólumaival.

Tény, hogy az ezekre való megállapítás könnyedén megtörténhetne a JavaScript kódba is, mivel az értékekre itt is tudjuk a korábbi műveletet alkalmazni. Ellenben

ezzel megbontanám azt a rendszert, amellyel szeparált módon kezelem a táblák adatait egy feladatként, és a másikban végezném el a weboldalon való megjelenítését.

A JavaScript kód minden oldal működéséhez az alábbi függvényeket használja:

- *handleFileSelect()*
- *setGame()*
- *selectNumber()*
- *selectTile()*

A kód elején definiálja és inicializálja a változókat, várja a feltöltéshez az Upload gombra való kattintással a CSV fájl importálását a verzióhoz mérten. Ebben az esetben a *handleFileSelect()* függvény kerül meghívásra, a kapott tartalmat felbontja sorokra, amelyek egyenként megfelelnek egy sudoku tábla értékeinek, valamint fennálló lehetőséggel a hozzá tartozó tulajdonságokkal együtt. Ezek közül választ egy véletlenszerűt, kiíratja a konzolra annak azonosítóját, és a változókba szeparált módon eltárolja a sorhoz tartozó említett adatokat, a későbbi játék felhasználásához.

Programkód 4.17. Fájlkezelés

```
function handleFileSelect(event) {
  const file = event.target.files[0];
  const reader = new FileReader();
  reader.onload = function (e) {
    const csv = e.target.result;
    const rows = csv.split('\n');
    const board_index=Math.floor(Math.random() * (rows.length - 1)) + 1;
    console.log("You play board: ", board_index);
    const row = rows[board_index].split(",");
    board_value = row[0].trim(); // values
    . . .
```

Ha előállt az alap adatsor, akkor ebből készül a tábla mátrixa, és a megoldás, amelyet alkalmaz az összehasonlításhoz a játék alatt.

Programkód 4.18. Megoldott tábla eltárolása

```
for (let index = 0; index < board.length; index++) {
  board_solution.push(board[index]);
}
```

A hiányos tábla olyan módon valósul meg, hogy az alap számjegyeket 0-ra cserélem. A kiválasztás véletlenszerű, de a hiányzó elemek száma előre meghatározott 60 darab. Több nehézségi szint a kivételre kerülő értékek számával is befolyásolható. Ezen függvény zárásakor az adatok előkészítve, átadásra kerülnek a *setGame()* nevű függvénynek, a játék megjelenítéséhez.

Programkód 4.19. Hiányos tábla létrehozása

```
// Changing the give number (60 as default) of numbers to 0.
while (num_of_empty > 0) {
  var rand_row = Math.floor(Math.random() * 9); // 0—8
  var rand_col = Math.floor(Math.random() * 9);
  var rand_item = board[rand_row][rand_col];
  if (rand_item !== "0") {
    board[rand_row] = board[rand_row].replace(rand_item, "0")
    num_of_empty — — ;
 } }
setGame(board);
```

A tábla cellái külön kezelhető div, típusos esetben fieldset elemekből állnak. A paletta létrejöttét követően a 9x9-es terület kerül generálásra. Az adatszétválasztási eljárás és az ellenőrzési esemény mellett a kód ezen része különbözik verziónkként. Itt van biztosítva a csempék számára a class-ok hozzáadása és értékek felvétele, mielőtt a táblához kerülnek gyerek objektumként.

Programkód 4.20. Fájlkezelés

```
function setGame(board) {
 // Digits 1—9
 for (let i = 1; i <= 9; i++) {
 //<div id="1" class="number">1</div>
 let number = document.createElement("div");
 number.id = i
 number.innerText = i;
 number.addEventListener("click", selectNumber);
 number.classList.add("number");
 document.getElementById("digits").appendChild(number);
 }

 // Board 9x9
 for (let r = 0; r < 9; r++) {
  for (let c = 0; c < 9; c++) {
   //<div id="0—0" class="tile">1</div>
   let tile = document.createElement("div");
   tile.id = r.toString() + "—" + c.toString();
   . . .
   tile.addEventListener("click", selectTile);
   tile.classList.add("tile");
   document.getElementById("board").append(tile);
   } } }
```

Egyéb fontos függvényt, a selectTile()-t emelném ki, amely egy kattintásra történő esemény. Ez a metódus vizsgálja a próbálni kívánt szám helyességét. Nem üres cella esetén visszatér, leáll az esemény, ellenkező esetben az eltárolt megoldás mátrixban ugyanazon helyen levő értékével kerül összehasonlításra. Helyesség esetén a csempe háttérszíne zöld, más esetben piros színre vált, 0.5 másodperc várakozás után pedig az alapot kapja vissza. A hibák száma az utóbbi esetben növekszik.

Programkód 4.21. Csempeválasztási függvény

```
function selectTile() {
  const errorStr = "Mistake(s): ";
  if (numSelected) {
    if (this.innerText != "") {
      return;
  }
    . . .
  if (board_solution[r][c] == numSelected.id) {
    this.innerText = numSelected.id;
    this.style.background = "#37c968";
  }
  else {
    this.style.background = "red";
    setTimeout(() => { this.style.background = "";
              this.style.transition = "1s"; }, 500);
    errors += 1;
    document.getElementById("errors").innerText = errorStr + errors;
} } }
```

4.4.2. Játékok vizualizálása

Közép-pont

Az alap Sudoku után elsőnek ezt a változatot szeretném bemutatni a tulajdonság függetlensége, vagyis a struktúrájából adódó játéklehetősége miatt. Ennél sem szükséges még jellemzőket tároló adatokat importálni. Viszont mivel a játéktípusból adódóan nem lehet bármilyen táblából kivitelezni, ezért szükséges a számára létrehozott CSV fájlt használni. Egyszerűen a 3x3-as blokkok középső csempéje kap plusz class-t, amely a CSS alapján kap kiemelt színt.

Programkód 4.22. Csempeválasztási függvény

```
function setGame(board) {
  . . .
  if (r % 3 == 1 && c % 3 == 1) {
    tile.classList.add("tile-center")
  }
  . . . }
```

Páros-Páratlan

Mivel ennél a Páros-Páratlannál már van a jellemzőkre utaló adatsor, így a kezelése kibővül az eddigi, alapokhoz képest. Kezdetben az alkalmazáshoz létrehozok egy egész szám típusú index változót, az adatok szétválasztásánál pedig egy string-et. Változatos ezeknek a használata, de a legtöbb esetben vízszintes sorban vannak az attribútumok megállapítása, ezért nem szükséges mátrix típusra konvertálni, ahogyan a cellaértékekkel is tettem.

Programkód 4.23. Páros-páratlan tulajdonság tárolás

```
function handleFileSelect(event) {
  . . .
  board_value = row[0].trim(); // values
  even_odd = row[1].trim(); // even-odd properties
  . . .
  setGame(board, even_odd);
}
```

A játékban a létrejött adatok felhasználásához a páros és páratlanságát adó karakterláncot iterálom végig. Ha a string értéke a léptetett index által elérve „e”-t kap páros, ellenkező esetben páratlan class-t ad az elemnek.

Programkód 4.24. Osztályok átadása

```
function setGame(board, even_odd) {
  . . .
  properties_index = 0
  if (even_odd[properties_index] == "e") {
    tile.classList.add("tile-even")
  } else{
    tile.classList.add("tile-odd")
  }
  . . . }
```

Szomszédos

Egy tulajdonságú a vízszintes és függőleges tengelyre tett alkalmazással. Egyenként index és string változó kerül definiálásra. A reprezentálás egy újabb értéklánc hozzáadásával valósul meg.

Programkód 4.25. Szomszédos tulajdonság tárolás

```
function handleFileSelect(event) {
 . . .
 board_value = row[0].trim(); // values
 neighbour_hor = row[1].trim(); // neighbour prop horizontal
 neighbour_verti = row[2].trim(); // neighbour prop vertical
 . . .
 setGame(board, neighbour_hor, neighbour_verti);
}
```

Említett módon, a tulajdonsággal ellátott játéktípusoknál az alap div elemek helyett fieldset kerül használatra, hogy megfelelő módon lehessen a gyermek objektumaikat transzformálni. Először ezeket a gyerek elemeket definiálom. Csempénkként ellenőrzöm, hogy az attribútum létezik-e és hogy az elem horizontális esetben nem az utolsó eleme-e a sornak. Utóbbi feltétel lehetővé teszi, hogy a jobb szélen helyezkedő elemeket egyáltalán ne vizsgálja. Az oszlopok szintjén ezt az utolsó sorban fogja figyelni. Igaz esetekben az iránynak megfelelő osztályokat és a fekete körrel, mint tartalommal ellátott div elemeket adja hozzá. Az indexek léptetése minden lefutáskor megtörténik, kivéve, ha az irány utolsó lehetőségéről van szó, azaz a tábla szélén, ahol egyébként sincs kezelendő érték.

Programkód 4.26. Osztályok átadása

```
function setGame(board, neighbour_hor, neighbour_verti) {
 . . .
 hori_index = 0
 verti_index = 0;
 let propRight = document.createElement("div");
 let propBottom = document.createElement("div");
 // Condition to create properties to the right of the cell
 if (neighbour_hor[hori_index] == 1 && c != 8) {
  propRight.innerText = "blackdot";
  propRight.classList.add("prop-right");
  tile.appendChild(propRight);
 }
 // Condition to create properties to the bottom of the cell
 if (neighbour_verti[verti_index] == 1 && r != 8) {
  propBottom.innerText = "blackdot";
  propBottom.classList.add("prop-bottom");
  tile.appendChild(propBottom);
 }
 . . .
}
```

Kropki

A kettős tulajdonsága révén ehhez dupla annyi változó szükséges. Az importálási adat rekordját 5 részre osztom.

Programkód 4.27. Osztályok átadása

```
function handleFileSelect(event) {
 . . .
 board_value = row[0].trim(); // values
 doubles_hor = row[1].trim(); // doubles prop horizontal
 doubles_verti = row[2].trim(); // doubles prop vertical
 neighbour_hor = row[3].trim(); // neighbour prop horizontal
 neighbour_verti = row[4].trim(); // neighbour prop vertical
 . . .
 setGame(board, neighbour_hor, neighbour_verti,
        doubles_hor, doubles_verti);
}
```

Az elemek létrehozása hasonló a Szomszédos változathoz. Itt ügyelni kell arra, hogy az egyik változóban 2-es, a másikban 1-es számmal vannak jelölve az attribútumok. A jelölés megkülönböztetésére más-más szín van alkalmazva. Kétszeresség esetén fehér, szomszédos különbség esetén fekete kör jellemzi, ahogy azt már látható volt az előző játékban.

Programkód 4.28. Osztályok átadása

```
function setGame(board, neighbour_hor, neighbour_verti) {
 . . .
 hori_index, verti_index = 0;
 let propRight = document.createElement("div");
 let propBottom = document.createElement("div");
 // Condition to create properties to the right of the cell
 if (doubles_hor[doubles_hori_index] == 2 && c != 8) {
  propRight.innerText = "whitedot";
  propRight.classList.add("prop-right");
  tile.appendChild(propRight);
 }
 // Condition to create properties to the bottom of the cell
 if (doubles_verti[doubles_verti_index] == 2 && c != 8) {
  propBottom.innerText = "whitedot";
  propBottom.classList.add("prop-bottom");

  tile.appendChild(propBottom);
 }
 // Condition to create properties to the right of the cell
 if (neighbour_hor[hori_index] == 1 && c != 8) {
  propRight.innerText = "blackdot";
  propRight.classList.add("prop-right");
  tile.appendChild(propRight);
 }
 . . . }
```

Relációs

3x3-as blokkjaiban adott tulajdonságok ismét más folyamatot igényelnek a webes megjelenítéshez a korábban megismertekhez képest. A tulajdonságok felvétele nem különbözik, csak azok használata. A setGame() függvényben a relációs jelet biztosító objektumok kapnak egy új osztályt, hogy nagyobb betűmérettel jelenjenek meg. A csempékre a feltétel biztosítja, hogy csak a megfelelőek, pontosabban a blokkok bal felső 2x2-es részterületek kapjon ilyen HTML tag-et. Az indexek inkrementálása szintén ezektől függően történik meg. Mivel a relációs jelek vertikálisan nem egyértelműen kivehetőek mire utalnak, így az azt formáló osztályt, a *prop-bottom*-t, transzformálom, vagyis elforgatom 90 fokkal.

Programkód 4.29. Osztályok átadása

```
function setGame(board, neighbour_hor, neighbour_verti) {
 . . .
 hori_index, verti_index = 0;
 let propRight = document.createElement("div");
propRight.classList.add("prop-right-rel");
 let propBottom = document.createElement("div");
 // Condition to create properties to the right of the cell
 if (doubles_hor[doubles_hori_index] == 2 && c != 8) {
  propRight.innerText = "whitedot";
  propRight.classList.add("prop-right");
  tile.appendChild(propRight);
 }
 // Condition to create properties to the bottom of the cell
 if (doubles_verti[doubles_verti_index] == 2 && c != 8) {
  propBottom.innerText = "whitedot";
  propBottom.classList.add("prop-bottom");
  tile.appendChild(propBottom);
 }
 // Condition to create properties to the right of the cell
 if (neighbour_hor[hori_index] == 1 && c != 8) {
  propRight.innerText = "blackdot";
  propRight.classList.add("prop-right");
  tile.appendChild(propRight);
 }
 . . .
}
```

A megvalósításról szóló fejezet végén kiemelem, hogy látható módon, a típusokhoz készült kódok alkalmazzák és kiterjesztik a korábbiakat, magasabb vagy újabb szintre emelik azokat.

5. fejezet

Tesztelés

5.1. Típusok eredményei

Az általam *Python* kódban, majd *JavaScript*-tel weben megjelenített oldalon található játékok működését szeretném bemutatni a Tesztelés fejezetben.

Előkészületnek, hogy legyen adat, amelyből a játék táblái készülnek el, mindenképp le kell futtatni a `sud_fill.py` fájlt, amely egyedi 9x9-es táblákat generál le. Amennyiben valamelyik, nem hagyományos típusú Sudoku-val szeretnénk játszani, az annak adatait generáló *Python* programot kell lefuttatni. A következő lista ebben a sorrendben mutatja az információkat [játék verziók : adatot készítő fájlok : végül a weboldalon használandó adatot tároló fájlok]:

- Közép-Pont : `center-dot-sdk.py` : `center_dot_sud_boardsSZD.csv`
- Páros-Páratlan : `even-odd-sdk.py` : `even-odd_sud_boardsSZD.csv`
- Szomszédos : `neighbours-sdk.py` : `neigh_sud_boardsSZD.csv`
- Kropki : `kropki-sdk.py` : `kropki_sud_boardsSZD.csv`
- Relációs : `relations-sdk.py` : `relation_sud_boardsSZD.csv`

Minden oldalon megtalálható a Menü, amelyen keresztül a többi Sudoku típusra tud váltani a játékos.

5.1. ábra: Az oldal menüje

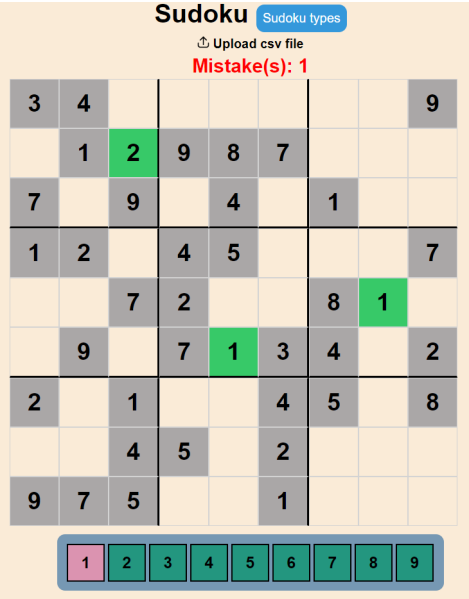
A kiválasztott játék oldalán a játékosnak az Upload gombra kattintva fel kell tölteni egy típusnak megfelelő CSV fájlt, amelyből az oldal legenerálja a táblát a játékhoz.

A Sudoku szabályait követve megfejtheti a kapott 9x9-es mezőt. A kívánt számjegyet az alsó palettából tudja kiválasztani egérkattintással, majd a tábla cellájába kattintva van lehetősége tenni. A megfejtéshez minden üres mezőt, vagy másnéven csempét tartalmazó számot meg kell találni!

5.1.1. Hagyományos

9x9 Sudoku

A hagyományos sudoku verzióban az 1-től 9-ig használható számjegyek sorban, oszlopban és 3x3-as vastagon elválasztott vonal menti területen csak egyszer fordulhatnak elő. A webes kialakításban az elképzelt formában kiválaszthatóak a számjegyek, és megfelelően ellenőrzi a helyességet. Pozitív találat esetén zöld csempeszínnel jelzi azokat, hibázáskor pedig piros felvillanás mutatja az adott helyen.

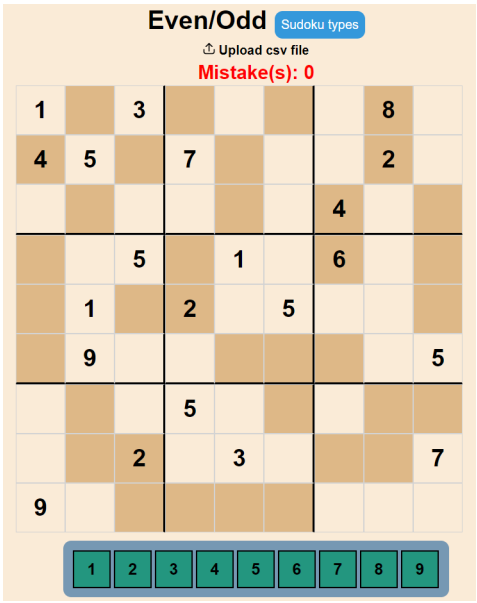


5.2. ábra: 9x9 Sudoku

5.1.2. Önálló

Páros-Páratlan Sudoku

Páros-Páratlan Sudoku egyes mezőiben levő szám páros vagy páratlan tulajdonságától függően más háttérszínűek. Az alap funkciók mellett a típusnak kialakított adatfájlt jól alkalmazza.

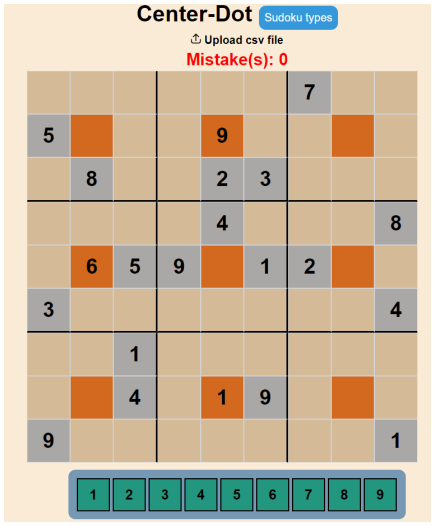


5.3. ábra: Páros-Páratlan Sudoku

5.1.3. Struktúrális

Közép-Pont Sudoku

A Közép-Pont Sudoku-ban a 3x3-as vastag vonallal határolt területek középső mezőiben is a számsor elemei csak egyedien vannak. A játék igazolja a szerkezetileg helyesen kiválasztott táblákat.

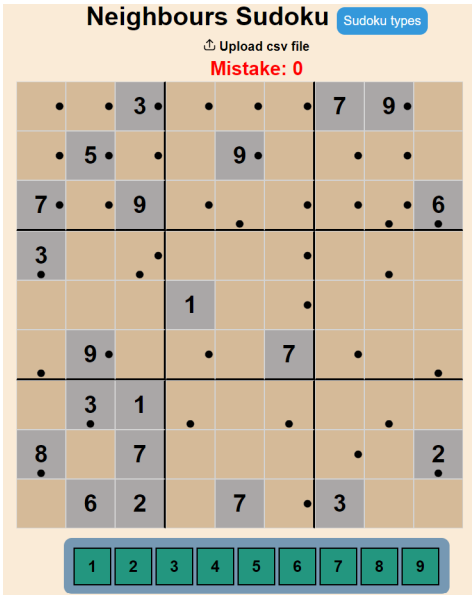


5.4. ábra: Közép-Pont Sudoku

5.1.4. Tulajdonságok

Szomszédos Sudoku

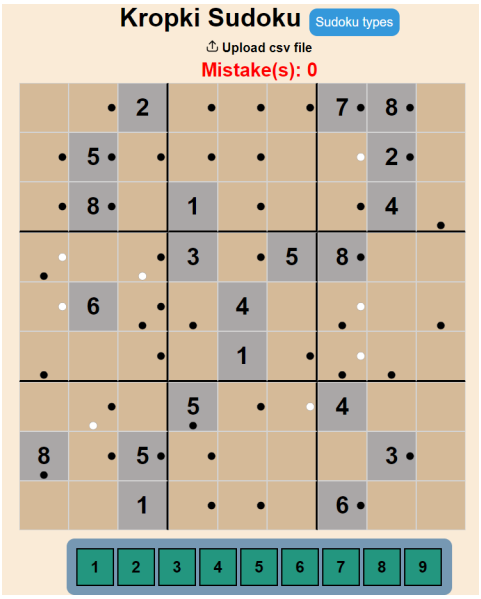
Szomszédos Sudoku azon változat, ahol fekete kört van kettő szomszédos szám között, ha azok különbsége 1. A jelölések jól látható módon kivehetőek és egyertelműen jelzik, mely cellák között áll fent a kapcsolat.



5.5. ábra: Szomszédos Sudoku

Kropki Sudoku

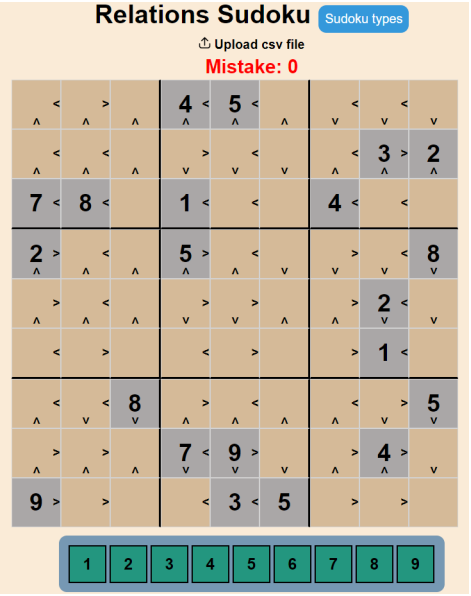
Kropki Sudoku egy, a mezők közötti duplaságot fehér körrel jelző játétípus, mely tartalmazza a Szomszédos tulajdonságokat is fekete kör formájában. Kontrasztos színnel, kivehetően ábrázolja az új tulajdonságot. Az összetett adathalmazát kiválóan alkalmazza.



5.6. ábra: Kropki Sudoku

Relációs Sudoku

A Relációs Sudoku az egymással szomszédos számokra ad relációs segítséget a 3x3-as blokkon belül. A verzió jellemzőit elforgatja, és eltérő mérettel, kellemes, de nem sűrített hatást adva a játéktérnek.



5.7. ábra: Relációs Sudoku

6. fejezet

Összefoglalás

A Sudoku elterjedése és felkapottsága miatt számtalan típussal létezik. Ezeknek a megfejtéséhez és létrehozásának algoritmikusai a készítő által egy elképesztő feladat. Alkotóként rengeteg lehetőség állt rendelkezésemre az információk megszerzéséhez, a tanulás és a megvalósítás formáira, a mai technikai eszközök mellett.

A kitűzött feladat elvégzését erősen segítette az alkotás és a kíváncsiság, hogy milyen kivitelezésben lehet elérni a célt. A játék algoritmusait és webes felületét élvezettel teszteltem és írtam, szem előtt tartva azok dokumentálhatóságát és átláthatóságának megtartását. A típusok kutatása alatt megismertem újakat, és időközben építettem bele a dolgozatomba. Természetesen a tesztelés része érdekelt a legjobban, vajon működnek-e a játékok, megfelelően kezeli az adatokat és összeáll az egész egy olyan felületté, ami az üzenetem tárgyát képezik: a tudás alkalmazását és a játékos gondolkodást kamatoztatva.

A rögös út akadályai olyan tényezők voltak, amelyek a komplexebb játéktípusokhoz köthetőek. A keresési eljárások és a kapott eredmény tesztelése okozott nehézséget. A legtöbb esetben a programokat egy esetre készítettem el, hogy egyáltalán működnek-e, majd ezt követően valósítottam meg általánosan. A bonyolultabb játékoknál hátráltató tényező volt az ellenőrzésre vonatkozóan, hogy több tábla összekapcsolásakor kevesebb esély volt ezek feltárására. Ez úton kézzel vittem be olyan mező adatokat, amelyekre kaptam a pozitív visszajelzést a programtól, hogy sikeresen megtalálta ezeket, tehát az algoritmus működik.

A jövőben szeretnék generáló algoritmusokat készíteni a többi meglévő típusra, újakat megalkotni és webes formában megjeleníteni őket.

7. fejezet

Summary

Due to the spread an popularity of Sudoku, there are numerous types. Deciphering and making the algorythms was an incredible task. As a creator I had plenty of opportunity to gain information, to learn and improve, which was helped by the modern technical tools.

For the goal of the pinned task was greatly aided by the creativity and curiosity to see what kind of implementation could achieve the goal. I enjoyed testing and writing the algorythms also the web interface of the game, keeping in mind the documentability and transparency. While researching the types, I learned about new ones and in the meantime incorporated them into my thesis. Obviously, the testing part was the most interesting. I had pure curoiosity whether the games worked, do they handle the data properly, and come together into an interface that is the subject of my message: applying knowledge and engaging playful thinking.

While writing the more complex game types there were obstacles on the bumpy road. Testing the searching methods and results caused difficulties. In most cases, I prepared the programs for a single case to see if they even work , and then implemented them in general. For the more complex games, the fact that there was less chance of detecting them when linking several boards was a hindrance to testing. This way I manually entered field data for which I received positive feedback from the program that it had successfully found them, thus proving that the subcorrectness worked.

In the future, I would like to create generating algorithms for the other existing types, create new ones and display them in web form.

Irodalom

[1] MatLab szoftver
https://www.mathworks.com/products/matlab.html

[2] Maple szoftver
https://www.maplesoft.com/products/Maple/

[3] C programozási nyelv
https://en.wikipedia.org/wiki/C_(programming_language)

[4] Java programozási nyelv
https://www.java.com/en/

[5] Google Colab
https://colab.research.google.com/

[6] Python programozási nyelv
https://www.python.org/

[7] PyCharm IDE
https://www.jetbrains.com/pycharm/

[8] HTML nyelv
https://en.wikipedia.org/wiki/HTML

[9] CSS nyelv
https://en.wikipedia.org/wiki/CSS

[10] JavaScript nyelv
https://en.wikipedia.org/wiki/JavaScript

[11] Különböző Sudoku táblák száma
https://pi.math.cornell.edu/~mec/Summer2009/Mahmood/Count.html

[12] Minimális számú cella megfejthetősége
https://phys.org/news/2012-01-mathematicians-minimum-sudoku-solution-problem.html

[13] Killer-Sudoku kombinációk száma
https://godoku.com/play/killer/combinations/

[14] Sudoku megoldó algoritmus

`https://www.techwithtim.net/tutorials/python-programming/sudoku-solver-backtracking`

CD Használati útmutató

A CD tartalma a dolgazatot LaTeX-ben megírt fájlok a `latex_files` mappája, valamint a témát képző programoké a `program_files`. Az előbbi mappában PDF formátumban megtalálható a szakdolgozat `dolgozat.pdf` néven.

A *Python* programkódok a `generators_and_datas`, a web futtatásához szükségesek a `web_files` mappában vannak tárolva.

A *CSV* és *Python* fájlok egy mappában vannak tárolva. Az adatokat tartalmazó *CSV* fájlok üresek, csak az oszlopok neveit tartalmazzák.

A generáláshoz elsőnek a `sudoku_fill.py`-t kell futtatni. A kívánt adathalmaz megtekinthető az aktuális `sud_boards_SZD.csv` fájlban. Típusonként a névszerinti Python fájlt szükséges futtatni.

A weboldal használata során valamelyik HTML oldalt kell megnyitni. A játszani kívánt és megnyitott oldalon az Upload CSV file gomb megnyomása szükséges. Ekkor feltölthető a korábbi mappából a típushoz tartozó kért fájl.