# SameGame NP teljes puzzle megoldásának megközelítése különböző módszerekkel

Kárpáti Márk András

2022 Tavaszi félév

## 1 A SameGame játék

SameGame egy puzzle játék, amit Kuniaki Moribe talált ki chain shot! néven 1985-ben. Fujitsu FM-8/7 sorozatra volt kiadva, és egy havi lapban jelent meg egy számítógépes napi lapban. [16]. Később Eiji Fukumata a SameGame névvel újra kreaálta 1992-ben.

### 1.1 Szabályok

- pálya mérete 15x15
- 5 különböző fajta blokk szín van.
- Kezdetben a pálya összes mezején van valamilyen szín
- Egy lépés egy csoport eltávolításából áll.
- Egy csoport legalább 2 egyszínű blokk, ami érintkezik egymással, szomszédosak.
- Egy lépés $(n-1)^2$ pontot ér, ahol n az egy lépésben eltüntet blokkok száma
- Nincs idő korlát.
- Nincs lépés szám korlát.
- Minden lépés után a játék pálya "összeomlik" ha van üres hely az elemek lefelé mozdulnak, ha van egy egész üres oszlop, akkor balra felé tolódnak az oszlopok.
- A játék véget ér, ha elfogytak a blokkok vagy nem marad lehetséges lépés

## 1.2 SamGame komplexitása

A komplexitása, azt mutatja, hogy mennyire nehéz megoldani a játékot. Két fontos mértéke a játék-fa komplexitás és az állapottér komplexitás.[1] A játék-fa kompleszítás a levélcsomópontok száma a megoldási fában az induló pozícióból. Az állapottér kompleszítás az elérhető játék állások a kezdő pozicionál.

A SameGambe-ben a játék-fa komplexitás nagyjából $b^d$, ahol b az átlagos elágazási faktor és a d az átlagos játék hossza. b és d önjáték kísérletezéssel kideríthetők, 250 puzzle-ből az jött ki, hogy az átlagos játékhossz $d = 62.2$ lépés, az elágazás faktor pedig $b = 21.1$, ami $10^{82}$ komplexitású játék-fát eredményez. Az állapottér komplexitását lehet úgy számolni, hogy egy oszlopra nézzük meg. $10^{159}$[18]

Továbbá a játékot jellemzi, hogy melyik komplexitás osztályba tartozik. [12] A hasonló játék Clickomania bizonyítottan NP-teljes. [10] Ugyan akkor SameGame komplexitása lehet más. Minél több pontot ér a blokkok eltávolítása annál jobban eltérhet a Clickomania karakterisztikájától. A Clickomania-ban az egyetlen cél az összes blokk eltávolítása, amíg a SameGame-ben a nagy csoportú blokkok eltávolításáért is pont jár. Alább a bizonyítás, hogy pontozástól függetlenül is legalább NP teljes.

Ahhoz hogy bizonyítsuk a játékról, hogy NP teljes elég egy olyan játékra egyszerűsíteni, ami NP teljes. Ideális út találása helyet azt bizonyítjuk, hogy már egy adott megoldás optimalitásának ellenőrzése is NP teljes. A megoldás S úgy definiáljuk, hogy egy út a kezdő állapotból egy vég állapotba. Vagy (1)minden blokkot eltávolított a játék pályáról vagy (2)befejeződött még blokkokkal a pályán. Mind két esetben vizsgálni kell, hogy van-e megoldás, ami javítja a pontot, azzal hogy mindent eltávolít. Ha ilyen megoldás találása NP teljes, akkor a SameGame is NP teljes.

Mivel Clickomania olyan változata a SameGame-nek ahol nem jár pont a lépésekért, csak egy megoldást kell találni, ezért SameGame legalább olyan nehéz mint a Clickomania. Mivel a Clickomania NP teljes legalább 5 szín és 2 oszlop esetén[10], ezért SameGame legalább NP teljes.

# 2 Kereső algoritmusok puzzle-okhoz

Algoritmusok, amiket eddig használtak a SameGame megoldása során és azoknak a részletei.

## 2.1 DBS keresés

A viszonyítási alap (Benchmark) a SameGame kutatáshoz 20 pocizóra van standardizálva. Az első SameGame programot [2] írta. Ez a program egy nem dokumentált DBS-t (Mélységi keresés költségvetéssel) használ. Amikor a program

eléri azt a mélységet, ahol elfogy a kerete, akkor egy mohó algoritmus lefut. Ez a program 72,816 pontot ért el 2-3 óra kereséssel pozíciónként.

## 2.2 Tipikus módszerek puzzle játékokhoz A* IDA*

Klasszikus megközelítése a puzzle-k megoldásának **A*** [11] és az **IDA*** [15]. Az A* egy Legjobbat először keresés, ahol minden csúcs el van tárolva egy listában. Ez a lista rendezve van egy elfogadható kiértékelő függvénnyel. Minden iterációban az első elemet eltávolítjuk, és a gyereke elemeit hozzá adjuk a rendezett listához. Ezt addig ismételjük, amíg a célállapot nem kerül a lista legelejére. IDA* az egy iteratívan mélyülő változata az A* keresésnek. Úgy használ mélységi bejárást, hogy nem kell eltárolni az egész fát a memóriában. Addig keres mélységi eljárással, amíg nem ér levébe és a kiértékelő függvény értéke nem ér el egy küszöböt. Ha a keresés eredmény nélkül tér vissza, akkor a küszöböt növeljük.

Mind két módszer nagyon erősen függ a kiértékelő függvény minőségétől. Még ha a függvény egy elfogadható alsó becslés, akkor is elég pontosnak kell lennie. Klasszikus puzzle játékoknál mint a Nyolcas játék, annak nagyobb változatai [15] és a Sokoban [13] ez a megközelítés nagyon jól működik. Hiszen ott egy jó alul-becslő függvény a Manhattan távolság. A fő feladata ennek területnek a kiértékelő függvény javítása pl mintaadatbázisokkal. [9] [8]

Ezek a megközelítési módok elbuknak a SameGame esetén, mert nem annyira egyszerű egy elfogadható függvényt (nem becsli túl az árat) készíteni, ami még mindig elég pontos. Erre az egyik próbálkozás, hogy pontokat asszociálunk a táblán lévő csoportokhoz. Ez vissza is adja azt a pontszámot, amit nagyjából kapnánk, ha föntről lefelé haladva eltávolítanánk ezeket a csoportokat. De ha egy optimális pontszámot szeretnénk elérni a SameGame-ben, akkor a jelenlegi helyzethez egy "túl becslő" lenne szükségünk, hiszen nem a legrövidebb utat keressük, hanem a legtöbb pontot szeretnénk elérni. Egy elfogadható becslő függvényt kapunk, ha úgy tekintjük, hogy az összes ugyan olyan színű blokkot egyszerre el tudjuk távolítani a pályáról. Ezt lehet javítani azzal, hogy mínusz pontot ad, ha ezzel a táblát nem tudjuk teljesen kiüríteni. Azonban ez messze van az igazi ponttól, amit egy helyzetben lehet szerezni, és nem működik jól az A* és IDA*-val sem. A tesztek azt mutatják, hogy ezzel a becslővel ellátott A*, IDA* Szélességi kereséshez hasonlóan viselkednek.[18] A probléma az, hogy a gyerek heurisztikus értéke sokkal kisebb lesz mint a szülője, hacsak nem egy nagy csoportot távolított el. Sejtések szerint ugyan ettől szenvednek a mélységi keresés, az Elágazás és korlátozás algoritmus is. [20] Ezekből kifolyólag jelent kihívást a SameGame a puzzle-k kutatásában.

# 3 Egy játékosos Monte-Carlo fakeresés

Továbbiakban Monte-Carlo tree search-t MCTS-nek fogom rövidíteni. MCTS egy olyan legjobbat először keresés, aminek nincs szüksége értékelő funkcióra. MCTS egy kereső fát épít, úgy hogy a leveleknél Monte-Carlo szimulációt használ. A fa minden csomópontja egy játék állást reprezentál és általában tartalmazza az rész-fa átlagos pontját, és hogy mennyiszer lett meglátogatva. MCTS a fakeresések egy családját elkotja, ami a társasjátékok területére vonatkozik. [3] [7] [14]

Általánosan a MCTS négy lépésből áll, amíg ki nem fut az időből. [4] (1)Egy *kiválasztási stratégia* , ami a fa bejárására szolgál, a gyökértől a levélig. (2) Egy *szimulációs stratégia* ami a játék befejezésére szolgál a keresési fa levélpontjától. (3) Egy *kiterjesztési stratégia*, ami megmondja, hogy mennyi és melyik gyerek vannak eltárolva, mint ígéretes levelei a fának. (4) Végül az MC eredményét gyökér felé vivő stratégia, a *Visszaterjesztő stratégia*

A MCTS-re alapozva, a játékokhoz adaptált változatának használata, *Single-player Monte-Carlo tree search (SP-MCTS)* [17]. A következő részben MCTS és SP-MCTS különbségeiről lesz szó.

## 3.1 SP-MCTS 4 lépés

A négy lépés iteratívan megy, amíg ki nem fut az időből. Ezek a lépések: (1) kiválasztás, (2) kijátszás, (3) bővítés és (4) visszaterjesztés.

## 3.2 Kiválasztás

Kiválasztás az egy stratégiai feladat, hogy egy adott csomópont gyerekét kiválassza. Ez irányítja az egyensúlyt a kiaknázás és felfedezés között. A kiaknázás során, olyan lépésekre fókuszál, amik az eddig a legjobb eredményhez vezettek. A felfedezés a kevésbé ígéretes lépések foglalkozik, amiket lehet még fel kell fedezni az eddigi értékelésük bizonytalansága miatt. MCTS-nél minden csomópont, ami a gyökérből indul ki kell választani, amíg elérünk egy olyan pozícióba, ami még nincs benne a fában. Több stratégia is van erre a feladatra. [3] [7] [14]

Kocsis és Szepesvári [14] egy UCT (upper confidence bounds)/(felső bizalmi határ) ajánl kiválasztáshoz. Az SP-MCTS-hez egy módosított UCT verziót használtak. A p csomopot i gyerek kiválasztásánál olyan lépést választ, ami a következő függvényt maximalizálja.

$$v_i + C \times \sqrt{\frac{\ln n_p}{n_i}} + \sqrt{\frac{\sum r^2 - n_i \times v_i^2 + D}{n_i}}$$

Az első két kifejezés az eredeti UTC-ből áll. Ahol $n_i$ hogy az i-dik csomópontban hányszor lett meglátogatva, $i$ jelöli a gyereket és $p$ szülő az adott felső bizalmi

határ az átlagos játékérték $v_i$. A puzzle számára egy harmadik kifejezést, ami reprezentálja a lehetséges eltéréseket a gyerek csomóponttól [3] [7]. Ez tartalmazza a megoldások négyzetének összegét $\sum r^2$ amit eddig elért a gyerek csomópont kijavítva a várt eredménnyel $n_i \times v_i^2$. Egy magas konstans érték D-t hozzá adnak, ami biztosítja, hogy ritkán tekintett csomópontok is ígéretesnek tűnjenek. Mivel az eltérés nem tartomány specifikus, hasznos lehet más tartományokban, ahol pontok variációja fontos funkció (pl.: Puzzle játékok, ahol pontszerzés a lényeg). Alább 2 különbséget írok le a puzzle és a két játékos játékok között, ami befolyásolhatja a kiválasztási stratégiát.

Első, az alapvető különbség a puzzle- és a két játékos játék között *értéktartomány*. Két játékos játékban a végeredményt általában vagy vesztés, döntetlen győzelem vagyis (-1,0,1). Egy csomópont átlagos pontja a [-1,1] tartományban van. Egy puzzle-ben tetszőleges pontot el lehet érni ami nincs benne az előre beállított intervallumban. Például a SameGame-ben vannak pozíciók amik 5.000 pont feletti pontot ér. Egyik megoldás lehet egy konstansokat $C$ és $D$ olyanra állítunk be amik lehetséges némelyik intervallumokban (például [0,6000] SameGame esetén). Másik megoldás, hogy az értékeket vissza vezetjük az előbb említett intervallumba [-1,1], de ehhez kell ismerni a maximum értéket. Amikor nem ismerjük, akkor egy elméleti felső határt lehet használni. SameGame esetében egy ilyen határ lenne, ha az összes blokk ugyan olyan színű. Ennek a nagy határnak a következménye lehet, hogy a legtöbb eredmény a 0 környékén helyezkedik el. Vagyis $C$ és $D$ teljesen más értékekkel kell ellátni.

A másik különbség, hogy puzzle esetén nem kell az ellenfelünk döntésének bizonytalanságán gondolkozni. Vagyis a játék az ellenfél akadályozása nélkül számolható [5]. Emiatt, nem csak az átlag pont használható, hanem az eddigi lépések maximumok pontja is. Kézi beállítás alapján a maximális pontot $W$ 0.02 értékül súllyal adták hozzá az átlag ponthoz.[18]

## 3.3 kijátszás

A kijátszás akkor kezdődik meg, amikor olyan pozícióba megyünk, ami még nem része a fának. Lépések véletlen szerűen vannak kiválasztva, amíg a játék nem ér véget. A választás minőségének érdekében szinte randomizálva vannak kiválasztva, vagyis heurisztikát használva.

Két megoldást említettek TabuRandom és a TabuColorRandom-t. Mind két stratégia, egy nagy csoportot próbál létrehozni egy színből. SameGame-ben egy nagy csoport létrehozása előnyös. Az ötlet az, hogy a kiválasztott színt addig nem választhatja, amíg van másik lehetőség. Ezzel a stratégiával automatikusan nagy csoportok formálódnak abból a színből. Az újítás a TabuColorRandomban, hogy a leggyakrabban előforduló szint hagyja ki. Ez növelheti a nagyobb csoportok kialakulását a kijátszásban. Még ehhez hozzá vették epszilon mohó politikát, hogy az előbb említett stratégiától eltérjen.[19] Mielőtt a szimulált

stratégiát használnánk epszilon eséllyel egy véletlenszerű lépés van használva.

Egy másik alternatíva rulett kerék kiválasztás használata a MCTS-hez. Ez nem csak egy csoport maximalizálására törekszik, hanem a többiére is. Mivel számításilag sokkal drágább ez a módszer mint a TabuClorRandom, nem növeli SP-MCTS eredményét [19].

## 3.4 kiterjesztés

A kiterjesztési stratégia dönti el, hogy melyik csomópontok vannak eltárolva a a memóriában. Coulom [7] azt javasolta, hogy egy gyerekkel bővítsük kijátszásonként. Ezzel a stratégiával a kibővített csúcspont összhangban van az első olyan pozícióval, ami nincs benne a fában.

## 3.5 Vissza terjesztés

Ebben a lépésben a kijátszás végeredménye vissza van terjesztve a gyökérig. Az irodalomban több stratégia is van ajánlva[7] [3]. Az egyik legjobb eredmény SP-MCTS elért az a kijátszások átlagos értékén alapult.

## 3.6 Utolsó lépés kiválasztás

A négy lépés addig van ismételve, amíg ki nem fut az időből. Amikor ez megtörténik, egy végső lépés kiválasztás történik, ami eldönti, hogy mit kell lépni. Két standard létezik erre, ami a a döntésüket a gyökér csúcspont gyerekeire alapozza. (1) Azt a gyereket választjuk aminek a legnagyobb az átlag pontja és (2) annak gyereknek a választása, ami legtöbbször volt kijátszva. Puzzle játékok számára az a legfontosabb hogy a legmagasabb pontot találjuk meg. Ezért érdemesebb a legnagyobb átlag pontú gyereket választani.

Az alapvető különbség miatt, hogy ez egy játékosos játék, ezért lehetséges egy nagy keresést indítani a kezdeti pozícióból majd az összes lépést kijátszani. Ezzel a megközelítéssel megfontolásra kerül az elején minden lépés, addig amíg az időből nem fut ki. Érdemes lehet vizsgálni, hogy ez a megközelítés jobban teljesít-e, mintha lépésenként szabnánk neki időt.

## 3.7 Véletlenszerű újrakezdések

Megfigyelések alapján [18] SameGame-ben érdemes mély fákat generálni. Azonban, ha a legígéretesebb lehetőséget tárjuk fel, akkor SP-MCTS egy lokális maximumba szorulhat. Ezért véletlenszerű újra indítással és a generátorokhoz más (seed)-t használva ez a probléma megoldható. Mivel nincs információ megosztva

a keresések között. Ez hasonlít a gyökér párhuzamosításhoz [6].

Gyökér párhuzamosítás az egy hatékony módja, több mag használatának [6]. Azonban, Egyszálú rendszerben lehetne amellett érvelni, hogy ez megfelelő módja a lokális maximum elkerülésének. Mert valójában nincs igazi párhuzamosítás, ezt nevezték véletlenszerű újrakezdésnek.

# 4 Program képességei

A cél az, hogy minél jobb algoritmust találjak, a játék megoldására majd ezt implementáljam. Ehhez elkezdtem a játékot újra megalkotni.

Eddigi funkciói a programnak:

- Teljesen véletlenszerű pálya generálás, ahol a pálya mérete és a blokkok típusának számossága megadható
- Pálya beolvasása megadott .csv formátumban.
- Pálya kiírása a beolvasáshoz megfelelő .csv formátumban.
- Ellenőrzi, hogy maradt-e még lépés lehetőség.
- Pálya "összeomlását" elvégezni.
- Kiválasztani egy csoportot, és az összes tagját egyszerre eltüntetni.
- a pontszámot kiszámolni.

A pálya .csv formátumú fájlokban van tárolva, aminek első sora oszlop és sorok száma. Ezt követi a pálya az összes adat ';' karakterrel van elválasztva.

# References


[1] Louis Victor Allis et al. *Searching for solutions in games and artificial intelligence*. Ponsen & Looijen Wageningen, 1994.

[2] D. Billings. Personal communication with s.

[3] Guillaume Chaslot, Jahn-Takeshi Saito, Bruno Bouzy, JWHM Uiterwijk, and H Jaap Van Den Herik. Monte-carlo strategies for computer go. In *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium*, pages 83–91, 2006.

[4] Guillaume M JB Chaslot, Mark HM Winands, H Jaap van den Herik, Jos WHM Uiterwijk, and Bruno Bouzy. Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation*, 4(03):343–357, 2008.

[5] Guillaume Maurice Jean-Bernard Chaslot Chaslot. *Monte-carlo tree search*, volume 24. Maastricht University, 2010.

[6] Guillaume MJ-B Chaslot, Mark HM Winands, and HJVD Herik. Parallel monte-carlo tree search. In *International Conference on Computers and Games*, pages 60–71. Springer, 2008.

[7] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[8] Joseph C Culberson and Jonathan Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

[9] Ariel Felner, Uzi Zahavi, Jonathan Schaeffer, and Robert C Holte. Dual lookups in pattern databases. In *IJCAI*, pages 103–108, 2005.

[10] RUDOLF Fleischer, LARS Jacobsen, and JIAN MUNRO. The complexity of clickomania. In *More Games of No Chance—Combinatorial Games at MSRI, 2000*, volume 42, pages 389–404. Cambridge University Press Cambridge, England, 2002.

[11] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[12] David S Johnson. A catalog of complexity classes. In *Algorithms and complexity*, pages 67–161. Elsevier, 1990.

[13] Andreas Junghanns. *Pushing the limits: new developments in single-agent search.* University of Alberta, 2000.

[14] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[15] Richard E Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1):97–109, 1985.

[16] K. Moribe. Chain shot! gekkan ascii.

[17] Maarten PD Schadd, Mark HM Winands, HJVD Herik, Guillaume MJ-B Chaslot, and Jos WHM Uiterwijk. Single-player monte-carlo tree search. In *International Conference on Computers and Games*, pages 1–12. Springer, 2008.

[18] Maarten PD Schadd, Mark HM Winands, Mandy JW Tak, and Jos WHM Uiterwijk. Single-player monte-carlo tree search for samegame. *Knowledge-Based Systems*, 34:3–11, 2012.

[19] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[20] Nageshwara Rao Vempaty, Vipin Kumar, and Richard E Korf. Depth-first versus best-first search. In *AAAI*, pages 434–440, 1991.