UNIVERZITA J. SELYEHO – SELYE JÁNOS EGYETEM
FAKULTA EKONÓMIA A INFORMATIKY
GAZDASÁGTUDOMÁNYI ÉS INFORMATIKAI KAR

TROJROZMERNÉ CELLULÁRNE AUTOMATY

HÁROMDIMENZIÓS SEJTAUTOMATÁK

Semestrálna práca – Szemesztrális munka

Bc. Juraj Lukovics

2025

UNIVERZITA J. SELYEHO – SELYE JÁNOS EGYETEM
FAKULTA EKONÓMIA A INFORMATIKY
GAZDASÁGTUDOMÁNYI ÉS INFORMATIKAI KAR

VYUŽITIE FUZZY LOGIKY PRE RIADENIE SEMAFOROV

FUZZY LOGIKA ALKALMAZÁSA FORGALMI JELZŐLÁMPÁK IRÁNYÍTÁSÁRA

Semestrálna práca – Szemesztrális munka

Študijný program:	Aplikovaná informatika
Tanulmányi program:	Alkalmazott informatika
Číslo a názov študijného odboru:	18. - informatika
Tanulmányi szak száma és megnevezése:	18. - informatika
Školitelia:	prof. RNDr. Tibor Kmeť, CSc.
Témavezetők:	prof. RNDr. Tibor Kmeť, CSc.
Školiace pracovisko:	KINF - Katedra informatiky
Tanszék megnevezése:	KINF - Informatika Tanszék

Bc. Juraj Lukovics

Komárno, 2025

Tartalomjegyzék

Ábrajegyzék

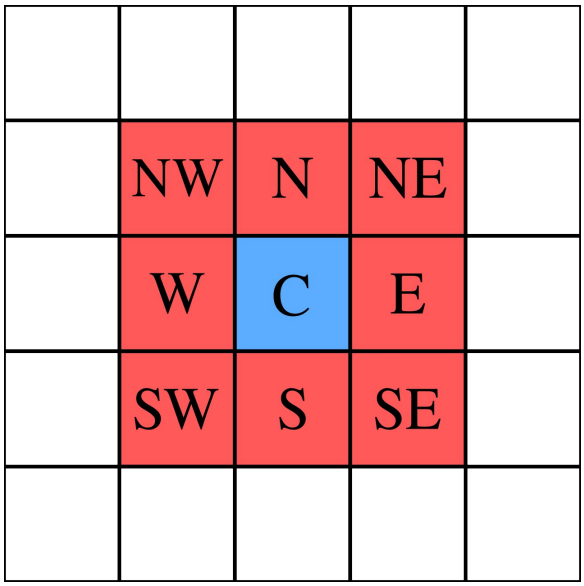
Bevezetés

Sejtautomatának nevezzük azokat a sejt halmazokat, amelyeknek szomszédjaik vannak. Minden szimulációs lépésnél a sejtek valamilyen állapotban vannak, ezek az állapotok általában véges számúak. Minden egyes szimulációs lépésnél minden vizsgált sejt állapota és az állapothoz meghatározott szabályok alapján változtatjuk a vizsgált sejtek állapotát. A sejtautomaták hasznosak, mivel néhány egyszerű szabály segítségével rendkívül komplex viselkedéseket tudunk lemodellezni.[2,5,6]

Kép 1: Conway-féle életjáték (Conway's Game of life) sejtautomata

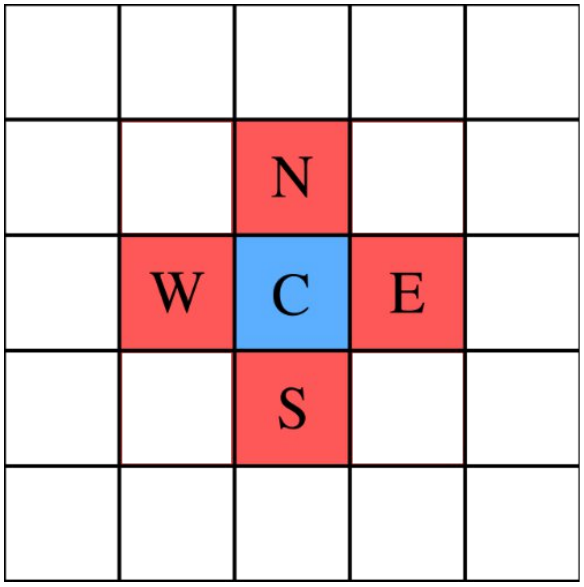
A legismertebb kétdimenziós sejtautomata a Conway-féle életjáték (Conway‘s Game of life), amely ismertségét Martin Gardner Scientific American tudományos magazinjában való megjelenésének köszönhette.[9] A megjelenés óta pedig kiderült, hogy képes számítások elvégzésére is.[1,4]

A sejteknek 2 állapota van, élő és halott; ezen felül még megkülönböztetünk 2 szabályhalmazt, az egyik azt adja meg, hogy a vizsgált sejt mennyi szomszédos élő sejttel éli túl a szimulációs lépést, a másik pedig azt adja meg, hogy mennyi élő szomszédos sejt kell ahhoz, hogy a vizsgált halott sejt élővé váljon. A szomszédság formája pedig Moore szomszédság.



Kép 2: Moore szomszédság

Rengeteg módosítást lehet alkalmazni ezekhez az alapvető szabályokhoz, például a sejtek több állapotot vehetnek fel, más szomszédság alapján vizsgáljuk a szomszédos sejteket (pl. von Neumann), valamint a szomszédság sugarát is változtathatjuk.

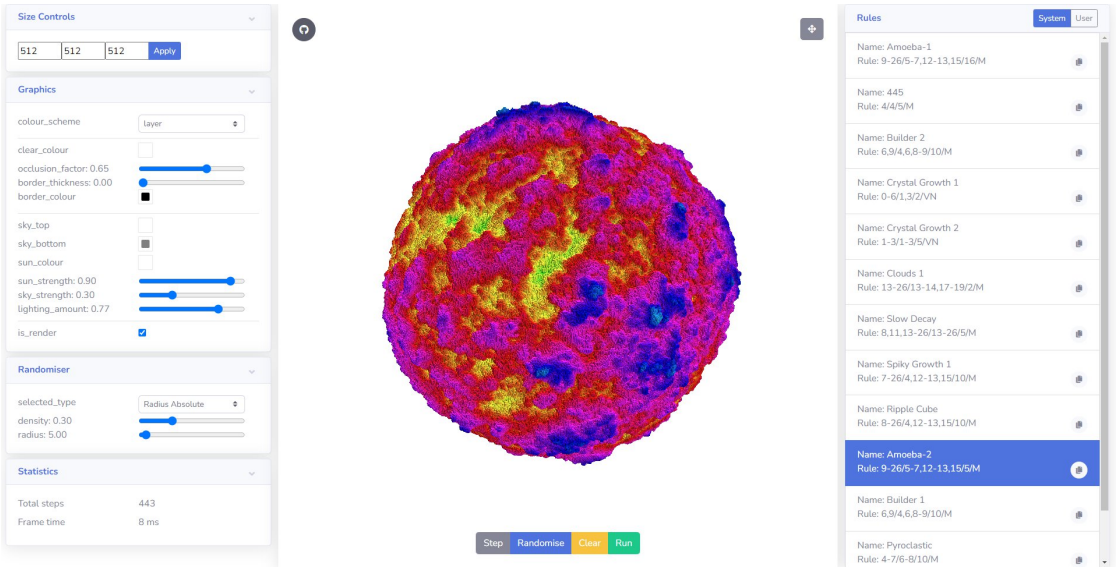


Kép 3: Von Neumann szomszédság

Eddigi megvalósítások

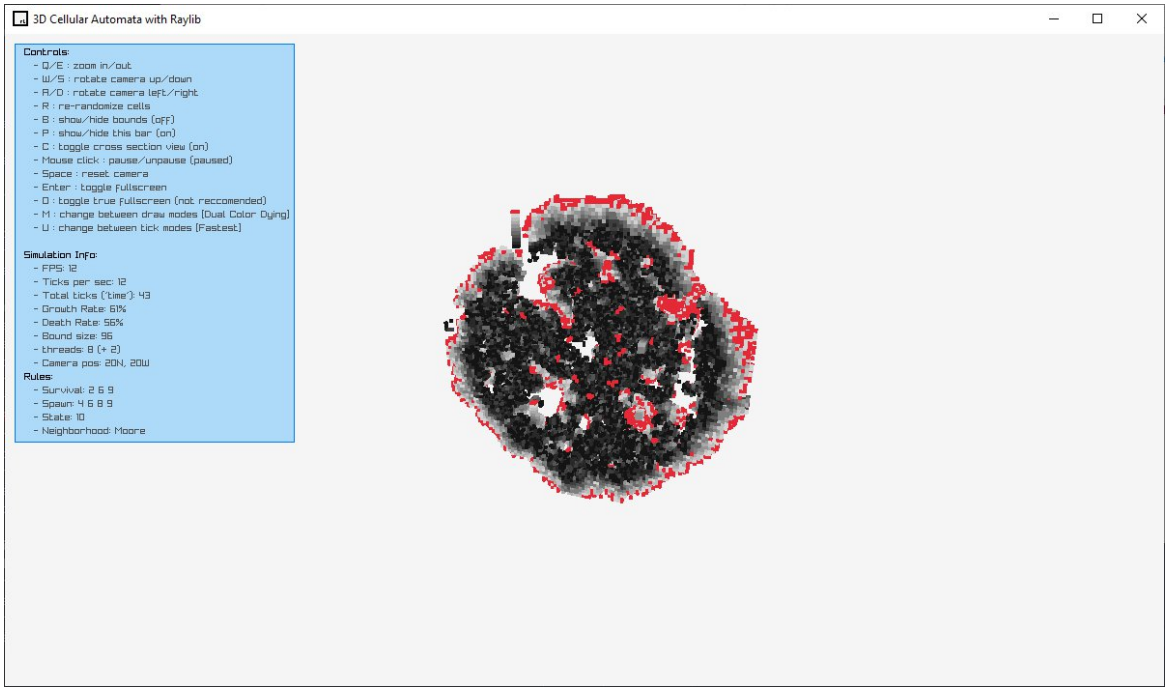
A munka elkészítése során találtam néhány megvalósítást, amelyek a funkciók implementálása szempontjából jó kiindulási alapot biztosítottak.

Az első William Yang, 3D-Cellular-Automata munkája volt, amely Javascript és WebGL segítségével lett elkészítve. A program háromdimenziós sejtautomatákat szimulál le és több állapot, illetve szabályok megadására is képes a programon belül a felhasználó.[7]



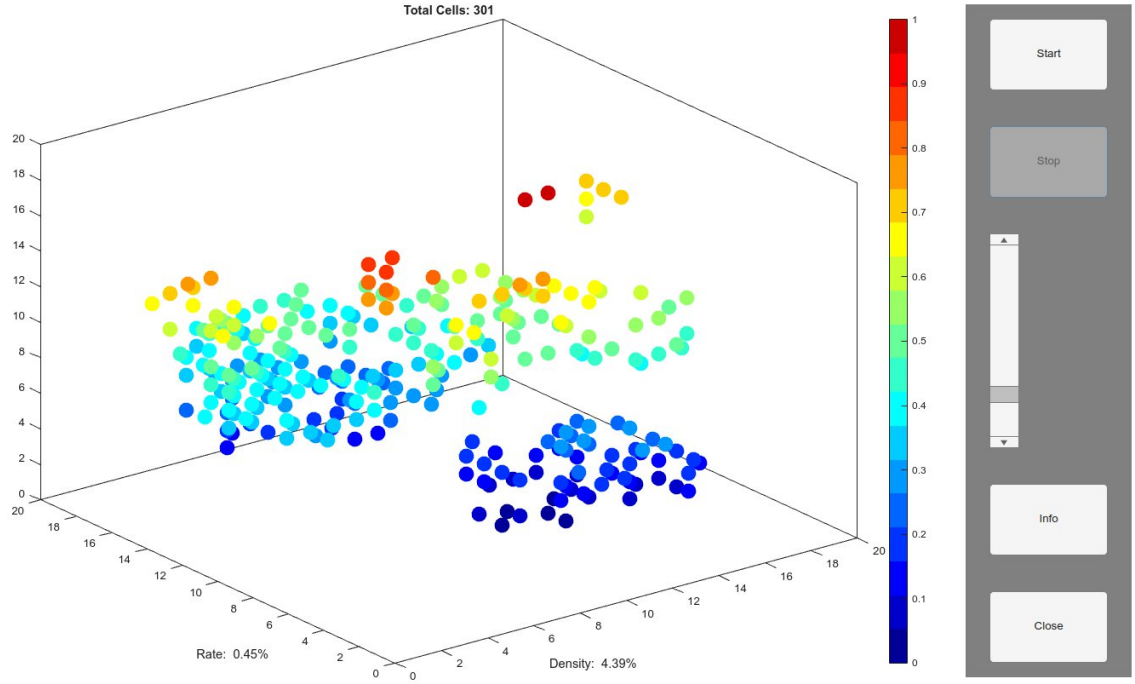
Kép 4: 3D-Cellular-Automata program (kipróbálható itt)

A második Millan Kumar, 3D-Cellular-Automata-Raylib munkája volt, amely C++ nyelven íródott és a Raylib grafikus könyvtárat használja.[3]



Kép 5: 3D-Cellular-Automata-Raylib program (letölthető innen)

A harmadik pedig Leandro Barajas, Conway‘s Game of Life in 3D munkája, amely Matlab-ban lett elkészítve. Eléggé hiányos, mivel szabályok nem adhatóak meg, valamint a programkódon belül sincsen megoldva más-más szabályok kezelése. A megjelenítés kicsit minimalisztikus, azonban emiatt elég jól skálázódik, tehát nagyobb méretű szimulációs tereket is jól tud kezelni.[8]



Kép 6: Conway‘s Game of Life in 3D Matlab program

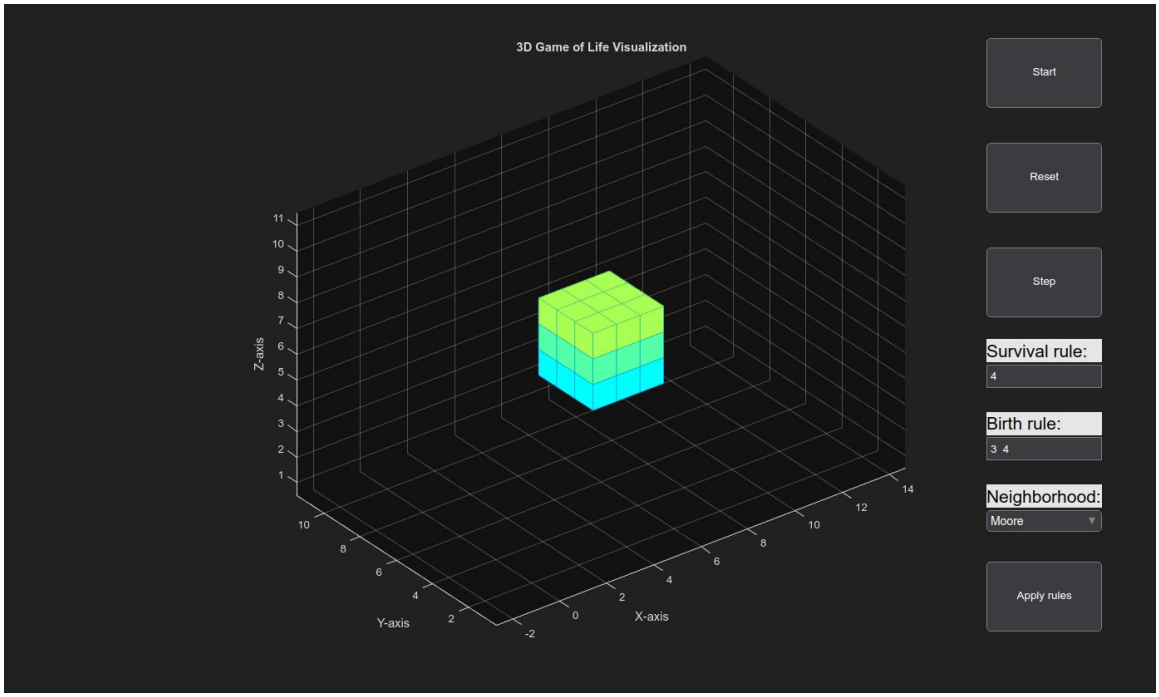
Felhasznált környezet

A program a Matlab programon belül lett elkészítve a Matlab szkriptelési nyelvén. Elsődleges cél volt, hogy a program könnyen használható legyen, valamint futási idő alatt módosítani lehessen a sejtautomata szabályait, illetve tetszőlegesen lehessen a szimulációt elindítani és leállítani.

A megjelenítéshez voxeles megközelítés lett alkalmazva, tehát minden élő sejtet egy voxelként jelenítünk meg. Sajnos a Matlab-on belül ez a fajta megjelenítés és gyorsan változó adatok nem kedveznek a teljesítménynek, így a program 11x11x11-es területen futtatja a szimulációt a felhasználói élmény maximalizálása érdekében. A sejtek két állapotot vehetnek csak fel, lehetnek élők (ekkor látszanak), vagy halottak (ekkor nem látszódnak).

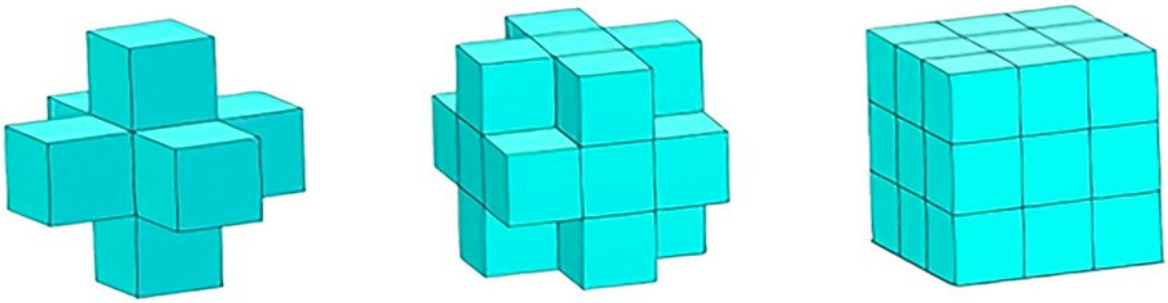
Program használata

A program a sejtautomata kezdő állapotában a középső sejtet és a Moore szomszédság alapján a szomszédos sejteket élő sejtté állítja be. Az alapvető szabály S4B3,4M; ez azt jelenti, hogy a vizsgált sejt akkor éli túl a jelenlegi iterációt, ha van 4 élő szomszédja és akkor válik egy halott sejt élővé, ha 3 vagy 4 élő szomszédja van, a szomszédokat pedig Moore szomszédság alapján vizsgálja, ami 3D-re kiterjesztve 26 sejtet fed le.



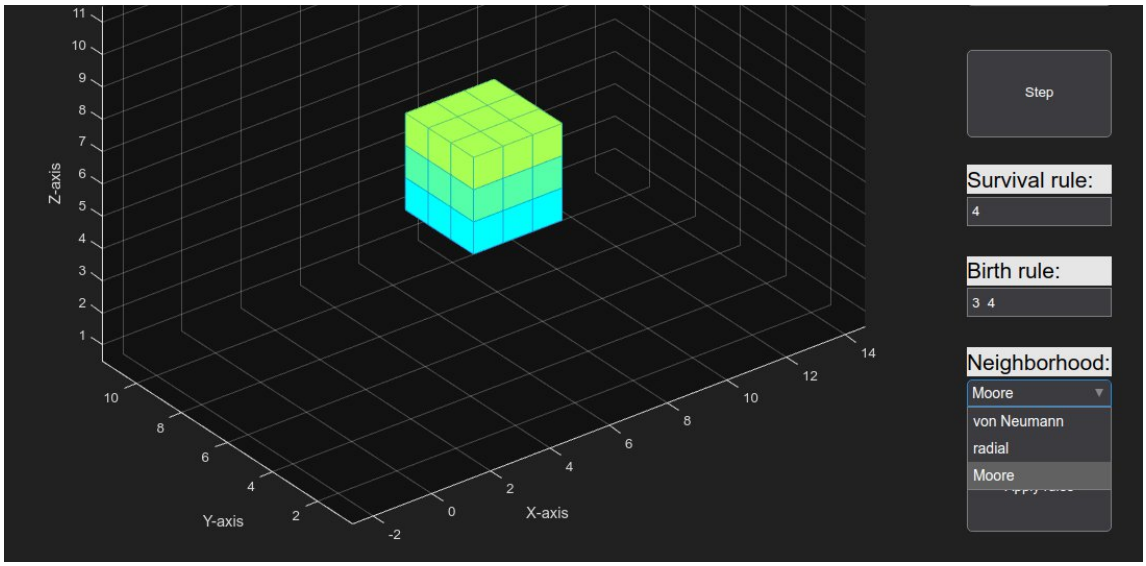
Kép 7: A program alapállapotban

A program 3 fajta szomszédsági ellenőrzést támogat, ezek a von Neumann, csillag/sugaras (radial) és a Moore.



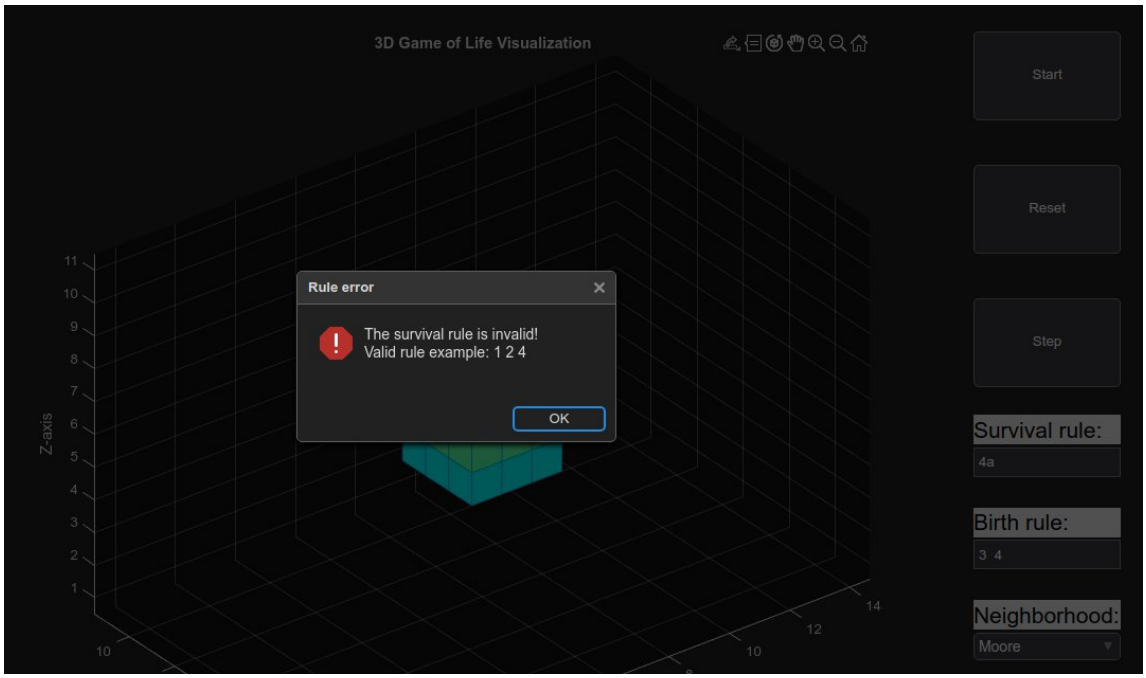
Kép 8: Von Neumann, csillag/sugaras (radial), Moore szomszédság

A programon belül a felhasználónak lehetősége van a szimuláció elindítására, illetve leállítására. A szimuláció előreléptetésére egy iterációval, alaphelyzetbe beállítani a szimulációt, valamint túlélési és születési szabályokat megadni, illetve szomszédsági ellenőrzést beállítani.



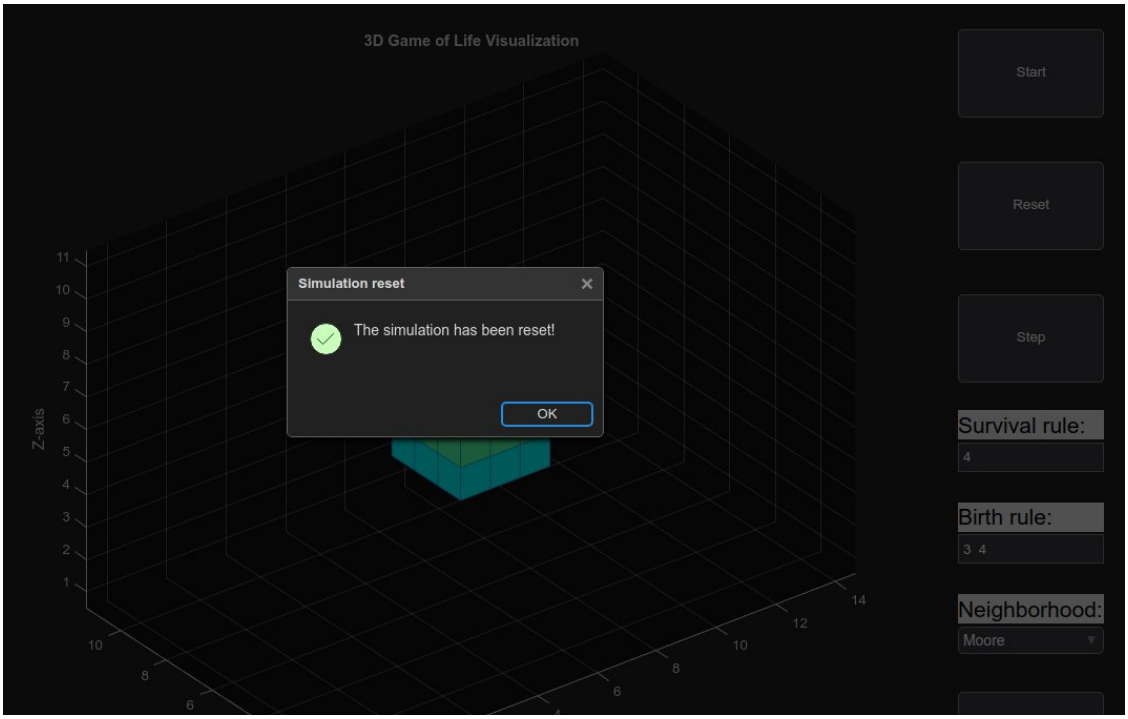
Kép 9: Szabályok beállításai a programon belül

A szabályokat úgy tudja a felhasználó megadni, hogy a szabály felirat alatti mezőjébe beírja a szomszédok számát szóközzel elválasztva, pl. „3 14“ – ez a szabály 3 vagy 14 élő szomszédnál lesz érvényes. A program képes kezelni érvénytelen formátumú szabályokat pl. „4a“ – ekkor a program hibát fog dobni a felhasználónak. A program nem dob hibát abban az esetben, ha érvényes számot ad meg a felhasználó, azonban az a szám kívül esik a szomszédsági mátrix által vizsgált sejtek mennyiségén. A szabályok alkalmazásához az „Apply rules“ gombra kell kattintani.



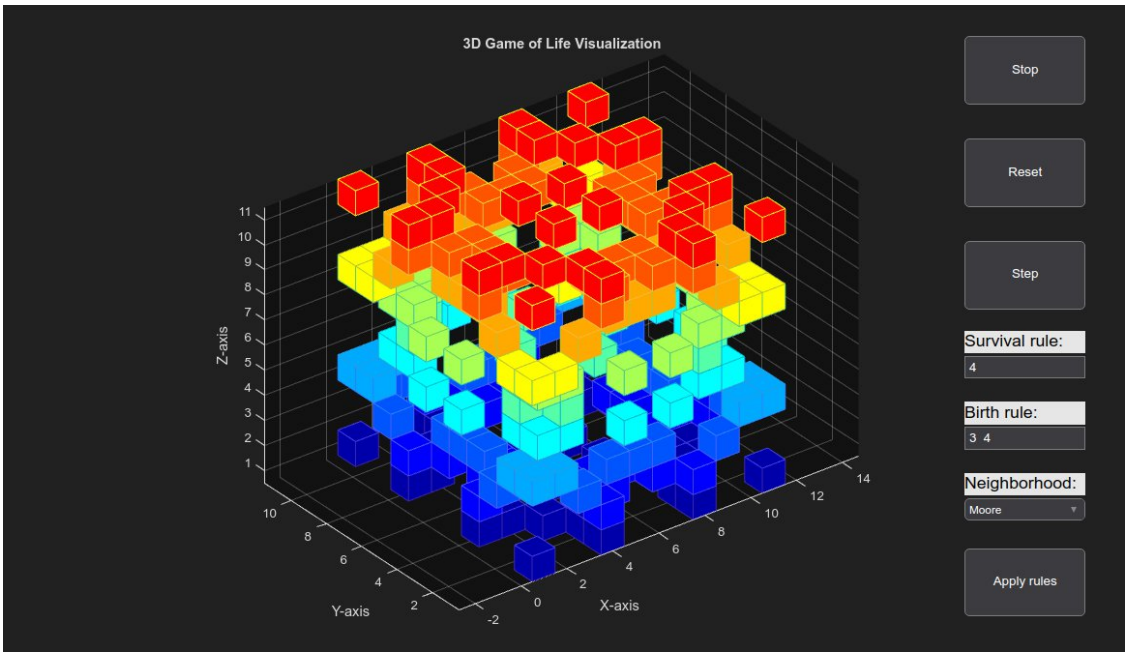
Kép 10: Program hibát dob érvénytelen szabály megadásakor

A szimulációt a „Reset“ gombbal tudjuk alaphelyzetbe állítani, ekkor a megadott legutolsó érvényes szabály nem vész el, csak a sejtek állnak vissza a kiindulási állapotba.



Kép 11: Szimuláció alaphelyzetbe állítása

A szimulációt a „Start“ gombbal tudjuk elindítani és a „Stop“ gombbal pedig megállíthatjuk. A „Step“ gombbal a szimuláción végigfuttatunk egy iterációt.



Kép 12: A program működés közben

Program működése

A program az első indításkor beállít néhány alapvető értéket, ezek közé tartozik a voxelek nagysága, a szimulált tér nagysága és az inicializált sejtállapotok. Ekkor feltölt a program egy háromdimenziós tömböt a sejtértékekkel (1 – élő sejt; 0 – halott sejt). Ezután, ha nincsen megadva szabály, akkor az alap S4B3,4M szabályt alkalmazza a program. Végül pedig beállítja a program gombjait és feliratait és hozzájuk rendeli a megadott függvényeket.

Azért, hogy ne duplikálódjon az ablak, illetve ne duplikálódjanak a voxel objektumok, minden futás előtt felvesszük az összes voxelt a grafikus ablakba, majd ezek átlátszóságát állítjuk aszerint milyen sejt állapot tartozik az adott voxelhez.

```
for i = 1:size( cells, 1 )
    for j = 1:size( cells, 2 )
        for k = 1:size( cells, 3 )
            [x,y,z] = cube( i, j, k, radius);
            hold on ;
            cell_voxels( i, j, k ) = surface( x, y, z, 'FaceColor', color( k, : ) );
            hold off ;
        end
    end
end
for i = 1:size( cells, 1 )
    for j = 1:size( cells, 2 )
        for k = 1:size( cells, 3 )
            if cells( i, j, k )==1
                set( cell_voxels( i, j, k ), 'FaceAlpha', 1.0 );
                set( cell_voxels( i, j, k ), 'EdgeColor', 'flat' );
            else
                set( cell_voxels( i, j, k ), 'FaceAlpha', 0.0 );
                set( cell_voxels( i, j, k ), 'EdgeColor', 'none' );
            end
        end
    end
end
```

A voxelek megjelenítéséhez szükségünk van a voxeleket alkotó háromszögek csúcsaira, amelyet a cube(x,y,z,radius) függvény ad meg. Ez a függvény úgy működik, hogy az x,y,z pozició lesz a voxel középpontja, majd ettől radius távolságra felveszi a csúcsokat, majd az így keletkezett kocka háromszögeinek visszaadja a csúcsait, amelyet a surface() Matlab függvény felrajzol a grafikus ablakba.

```
% Cube vertices that needs to be drawn

% The cube's center is on the [posX, posY, posZ] point and the other points

% are drawn by the value of the width apart

function [x,y,z] = cube(posX,posY,posZ,width)

x = [ posX + width, posX + width, posX + width, posX + width, posX + width; posX +
width, posX + width, posX - width, posX - width, posX + width; posX + width, posX +
width, posX - width, posX - width, posX + width; posX + width, posX + width, posX +
width, posX + width, posX + width ];

y = [ posY + width, posY - width, posY - width, posY + width, posY + width; posY +
width, posY - width, posY - width, posY + width, posY + width; posY + width, posY -
width, posY - width, posY + width, posY + width; posY + width, posY - width, posY -
width, posY + width, posY + width ];

z = [ posZ - width, posZ - width, posZ - width, posZ - width, posZ - width; posZ -
width, posZ - width, posZ - width, posZ - width, posZ - width; posZ + width, posZ +
width, posZ + width, posZ + width, posZ + width; posZ + width, posZ + width, posZ +
width, posZ + width, posZ + width ];

end
```

A szimuláció indításakor a program elmenti a program állapotát az isStarted változóba (ha el van indítva a szimuláció, akkor az értéke 1, ha nincs, akkor 0), majd ezt exportálja a guidata() Matlab függvény segítségével a grafikus ablakba. A szimuláció futása során a program megnéz minden sejt szomszédját a cells tömbben, majd az új állapotot a new_cells tömbben rögzíti, majd az ellenőrzés végén a cells tömböt feltölti a new_cells tömb értékeivel.

A tömb értékeit try catch blokkok segítségével érjük el, ezáltal elkerülve azt, hogy a tömb „szélén“ érvénytelen elemeket címezzünk. Ezeket a nem létező sejteket halottként kezeljük, tehát a default_val változóba 0-t rakunk.

```
% "Safe" array indexing

function value = getValue(array,x,y,z,default_val)

try

    value = array( x, y, z );

catch

    value = default_val;

end

end
```

Továbbfejlesztés, javítási lehetőségek

A Matlab-os implementáció legnagyobb hátránya az, hogy nincsen mód grafikai optimalizációk implementálására, valamint a több-szálas futás rendkívül limitált más programozási nyelvekhez képest. Ehhez a feladathoz egy játékmotor ideális lenne, mivel nézőpont alapján nem szükséges minden voxel háromszögét kirajzoltatnunk, valamint a két voxel közötti háromszögeket sem, ezáltal sokkal kevesebb erőforrást igényelne a megjelenítés.

A szabályok megadásánál érdemes lenne még intervallumok implementálására, így például nem kellene kiírni a felhasználónak a „1 2 3 4“ szabályt, hanem elég lenne csak „1-4“ szabályként megadnia, valamint még jobb hibakezelés is jó lenne a szabályokhoz.

Plusz lehetőségként jó lenne még ha a sejtautomata voxeles struktúráit ki lehetne exportálni valamilyen 3D modellfájlként, illetve lehetne megadni saját kezdőállapotot egyszerű szkripteléssel, vagy akár saját 3D modell, vagy 3D mátrix szövegfájl feltöltésével.

Kézenfekvő lehet még továbbfejleszteni a programot olyan módon, hogy több állapotot is tudjon a sejtautomata kezelni, valamint a voxelek színezését lehessen módosítani és például sejt állapot alapján legyenek színezve, vagy egyéb más szabály alapján.

Felhasznált irodalom

[1] GOUCHER, A.P. Universal Computation and Construction in GoL Cellular Automata. In ADAMATZKY, A.Ed. Game of Life Cellular Automata [online]. London: Springer, 2010. s. 505–517. [cit. 2026-01-03]. ISBN 978-1-84996-217-9Dostupné na internete: <https://doi.org/10.1007/978-1-84996-217-9_25>.

[2] ILACHINSKI, A. Cellular Automata: A Discrete Universe. . [s.l.]: World Scientific, 2001. 846 s. ISBN 978-981-238-183-5.

[3] KUMAR, M. LelsersLasers/3D-Cellular-Automata-Raylib [online]. . 2025. .

[4] LINDGREN, K. Universal Computation in Simple One-Dimensional Cellular Automata. In . .

[5] SCHIFF, J.L. Cellular Automata: A Discrete View of the World. . [s.l.]: John Wiley & Sons, 2011. 280 s. ISBN 978-1-118-03063-9.

[6] WOLFRAM, S. A New Kind of Science. . [s.l.]: Wolfram Media, 2002. 1286 s. ISBN 978-1-57955-008-0.

[7] YANG, W. williamyang98/3D-Cellular-Automata [online]. . 2025. .

[8] Conway's Game of Life in 3D. In [online]. 2026. [cit. 2026-01-03]. Dostupné na internete: <https://www.mathworks.com/matlabcentral/fileexchange/4892-conway-s-game-of-life-in-3d>.

[9] Mathematical Games - The fantastic combinations of John Conway's new solitaire game "life" - M. Gardner - 1970. In . 1970. .