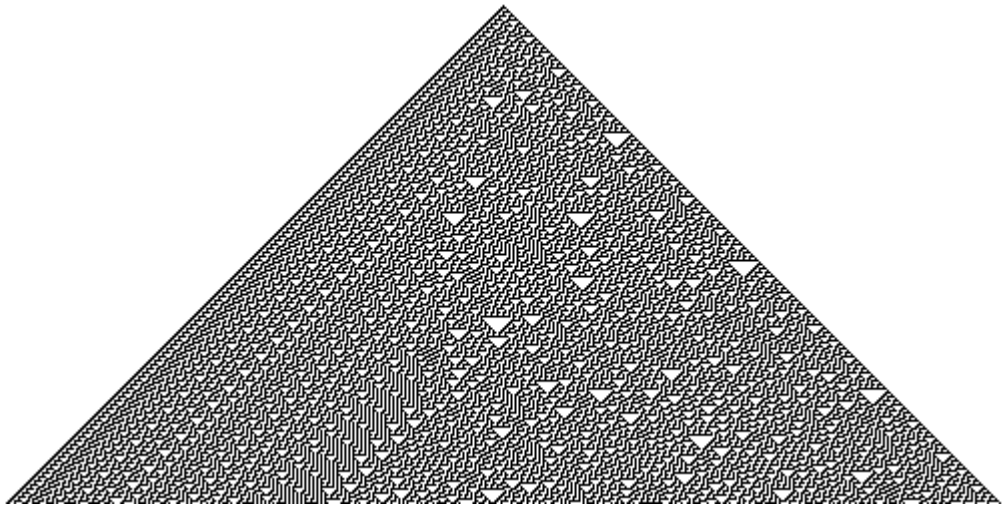


SEJTAUTOMATA DOKUMENTÁCIÓ

Szerző: Horváth Adrián Márk (PKFUUC)



1

¹ https://en.wikipedia.org/wiki/Cellular_automaton#/media/File:CA_rule30s.png

Célja

A sejtautomata program egy olyan program, ami képes egy adott szabály szerint szimulálni a sejtek alakulását. Maga a sejtautomata az automataelméletben tanulmányozott diszkrét számítási modell. A sejtautomatákat sejttereknek, tesszellációs automatáknak, homogén struktúráknak, celluláris struktúráknak és iteratív tömböknek is nevezik. Ezen számítási modellt több területen is előszeretettel használják, mint fizika, elméleti biológia és a mikrostruktúra modellezés. A programnak a célja, hogy egy ilyen szimulációt a lehető legpontosabban tudjon szimulálni.

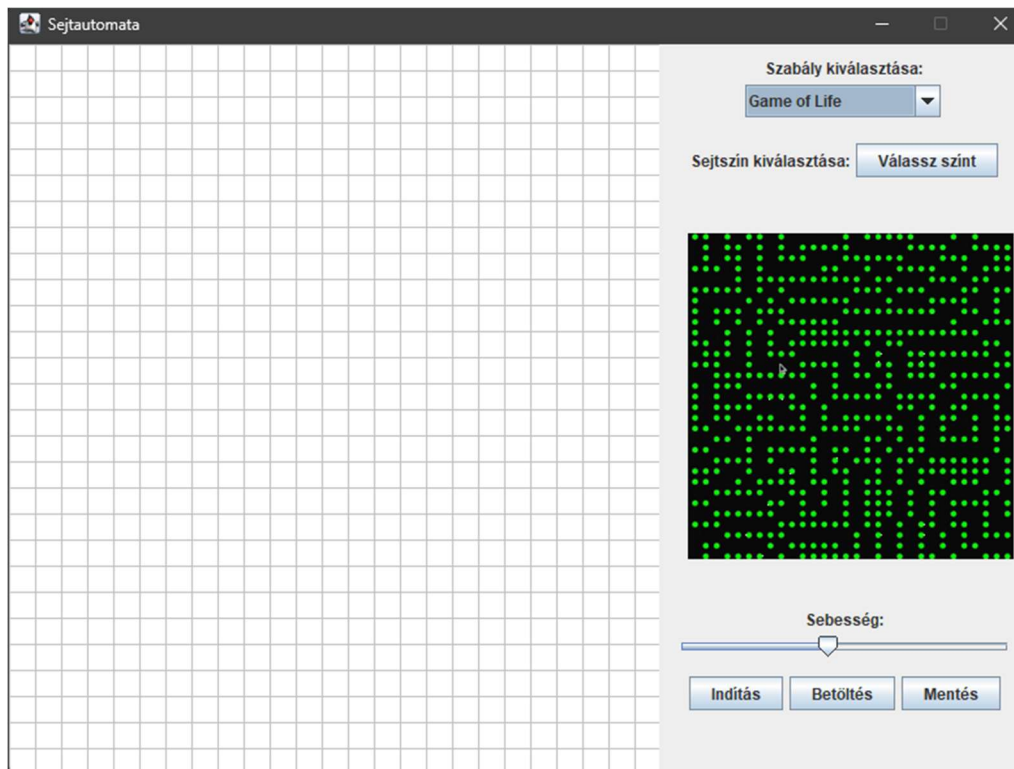
Szabályok

Maguk a sejtautomaták többféle szabály szerint tudnak modellezni. Csak párat említve ilyen a B1357/S1357 másnéven „Replicator”, B2/S vagy „Seeds”, vagy talán a leghíresebb a B3/S23 ami másnéven „Game of Life”. Maguk ezek a szabályjelölések két részből állnak. A „B” betűvel kezdődő rész azt jelenti, hogy „birth” az „S” betűvel kezdődő pedig „survival”. A „birth” az újjászületést hivatott jelezni, ami azt jelenti ha egy halott sejtnek egy adott mennyiségű szomszédja van akkor újjászületik. A „survival” pedig azt jelöli hogy az adott sejt körül ennyi élő sejt szükséges a túléléshez, különben elhal. Magukat a mennyiségeket azok a betűk után látható számok jelölik. Van azonban olyan eset amikor nincs szám a betű mögött, ilyenkor minden olyan esetben érvényes, amikor a feltételek teljesülnek. Ha több szám is van pl. B23, akkor a számok között „VAGY” kapcsolat van. Ezt úgy kell érteni, hogy előző példában például 2 vagy 3 sejt esetén születik újjá a sejt.

Több száz szabály létezik az ilyen sejtautomatákhoz, ezek közül hármat valósít meg a program. A Game of Life, Seeds, illetve a Replicator szabályt. A projekt úgy lett kialakítva, hogy a későbbi fejlesztések esetén könnyen lehessen új szabályokat implementálni.

Felület

A program indulásakor egy ablakon megjelennek a szimulációhoz szükséges eszközök. Bal oldalt az ablak 2/3-át elfoglaló mátrix panel jelenik meg ami a sejteket fogja megjeleníteni. A felhasználó könnyedén tud kijelölni négyzeteket (sejteket) és azokon a megfelelő gomb megnyomásával egy szimulációt is el tud indítani. Az alábbi kép szemlélteti, hogy a program, hogy is néz ki:



A képen jobb oldalt ezen kívül látszik hogy az egyes vezérlőgombok hogy helyezkednek el. A felső részen látható a szabály kiválasztó gomb, ami egy legördülő listában megjeleníti az eddigi implementált szabályokat (Game of Life, Seeds, Replicator). Alatta van egy szín választó gomb aminek a segítségével a sejtek színeit lehet tetszőlegesen kiválasztani. Középen látható egy design elem, egy internetről szedett sejtautomata GIF (<https://wp.nyu.edu/mickey/2023/04/05/cellular-automata/>) ami a program futása alatt folyamatosan megy. Ezen kívül ezen a vezérlőpanelen alul láthatóak a sebesség illetve az egyéb vezérlőgombok. A sebesség csúszkán lehet állítani a szimuláció sebességén. Az indítás gomb magának a szimulációnak az elindításáért felelős. A betöltés és mentés gombok pedig egy JSON fájlból történő beolvasást, illetve JSON fájlba történő mentést teszi lehetővé. A mentés és betöltés gomb megnyomása után egy fájl megnyitó/mentő Java felület jelenik meg, ahol a felhasználó könnyedén ki tudja választani hova szeretné menteni a mátrixot, illetve mit szeretne betölteni.

Működése

A projekt elkészítése után egy sejtautomata-1.0.jar fájlba csomagolta be a futtatható állományt, amit, ha a felhasználó megnyit akkor elindul a program. A program az elején egy üres mátrixot tölt be. Ebbe a mátrixba tud belenyúlni a felhasználó, ha a négyzetekre rákattint. Amennyiben nincs kijelölve egy négyzet (sejt) sem akkor a program egy hibaüzenetet dob fel, hogy nincs élő sejt kijelölve. Ha kijelölt sejteket a felhasználó akkor van lehetősége szabályt választani a jobb felső sarokban. A Game of Life szabály van alapból beállítva. Ezután a felhasználónak van lehetősége sejtek a színeit is kiválasztani a „Válassz színt” gomb segítségével. Majd betudja állítani a szimuláció sebességét is akár, de ezt a szimuláció futása közben is tudja módosítani. Az „Indítás” gombra kattintva pedig el is indul a szimuláció. A futás során az „Indítás” gomb átváltozik egy „Megállítás” gombbá, amit megnyomva a szimuláció leáll. A „Betöltés” gombra kattintva a felhasználó be tud tölteni JSON formátumú mátrixokat, illetve a „Mentés” gombot megnyomva pedig az aktuális mátrixot tudja kimenteni egy JSON formátumú fájlba. A program bezárásához az jobb felső sarokban levő X-et megnyomva be is záródik a program.

Projekt felépítése

A projektet az IntelliJ Community 2024.3-as programban lett elkészítve. A projekt felépítéséhez egy ismert projektmenedzsment eszköz lett felhasználva a Maven. A Maven szabályai alapján külön mappában találhatóak a fő program osztályai/interfészei, illetve külön mappában vannak a tesztelési osztályok is. Ezen kívül magát a projektet egy GitHub repoval lett összekötve, amivel folyamatos verziókövetést lehetett megvalósítani. A README.md fájlban röviden leírja mit csinál a program, amit a GitHub-on a projekthez tartozó repoban meg is jelenít. A projekt fájl szerkezetét az alábbiakban ismertetem:

- src: Ebben a mappában találhatóak meg a projekt forrásfájljai.
 - main: Fő program osztályai, illetve egyéb fájlok.
 - java: Itt találhatóak meg a fő Java osztályok két csomagra bontva.
 - automatonSimulation: A szimuláció működéséhez szükséges osztályokat tartalmazza.
 - rules: Az egyes szabályok logikáját definiáló osztályokat tartalmazza.
 - resources: Az egyéb forrásállományokat tartalmazza, pl. GIF fájl, illetve egy teszteléshez használt valid_matrix.json előre elkészített mátrix.
 - test: A teszteléshez szükséges osztályokat tartalmazza, amit a Maven rendszer lefuttat.
 - java: Itt találhatóak meg maguk a tesztelési osztályok, ami JUnit egységtesztek segítségével teszteli a program működését.
- pom.xml: Mivel rendszer alapján lett felépítve a projekt, ezért szükséges ez a fájl, ami leírja milyen függőségek szükségesek a projekt megfelelő működéséhez, ezenkívül egyéb projekt szintű beállításokat tartalmaz.
- Egyéb JSON fájlok: A tesztelések során olyan funkciók is tesztelve lettek, amik fájlt hoznak létre, illetve töltenek be, és ezek a fájlok itt jelennek meg, szám szerint 4 ilyen JSON fájl van. Több nem fog létrejönni, csak ezeket fogja manipulálni az egyes tesztosztályok.

Osztályok/Interfészek

A főbb osztályok az src/main/java mappában találhatóak. Itt két csomagra lettek szétosztva az osztályok annak érdekében, hogy átláthatóbb legyenek. Az osztályok két csomagra lebontva az alábbiak:

automatonSimulation:

- CellularAutomaton: ez az osztály azt hivatott szolgálni, hogy a mátrixot tárolja a program, ezen kívül pedig a mátrixon különféle manipulációkat lehessen csinálni.
 - Tagváltozók:
 - List<List<Boolean>> matrix: tárolja a sejtek állapotát (true – élő, false – halott)
 - Rule currentRule: a jelenleg kiválasztott szabály tárolja
 - Metódusok:
 - CellularAutomaton(int, int): a sejtek mátrixát inicializálja
 - void update(): a mátrixot frissíti az aktuális szabály alapján
 - boolean getCellState(int, int): egy cella (sejt) állapotát kérdezi le két paraméter alapján (sor, oszlop), majd visszaadja a sejt állapotát (boolean).

- void setCellState(int, int, boolean): egy sejt állapotát egy megadott értékre állítja be egy adott sorban és oszlopban.
 - List<List<Boolean>> getMatrix(): visszaadja a sejt állapotait tartalmazó mátrixot
 - void setRule(Rule): egy adott szabályra beállítja az aktuális szabályt.
 - void reset(): minden egyes sejt állapotát halottnak nyilvánítja (false-ra állítja).
 - void validateCell(int, int): egy adott sor és oszlop indexét megvizsgálja hogy érvényes koordinátát írnak-e le.
 - Rule getRule(): visszaadja az aktuális szabályt.
- ControlPanel: ez az osztály azért felel hogy megfelelően létrejöjjenek a megfelelő gombok és objektumok a vezérlőpanelen (a program jobb oldalán):
 - Tagváltozók:
 - JButton startStopButton: az „Indítás” és „Megállítás” gomb, ami a szimuláció indításáért és megállításáért felelős.
 - Metódusok:
 - ControlPanel(CellularAutomaton, MainWindow): ez felel a főbb komponensek létrehozásáért a vezérlőpanelen, ezen kívül pedig a CellularAutomaton objektumon a kiválasztott szabályt beállítja, szín kiválasztó működését definiálja, GIF beillesztését elvégzi, sebesség csúszkát létrehozza, illetve gombokat létrehozza.
 - void updateStartButtonText(String): a startStopButton gomb (Indító/Megállító) szövegét állítja át egy megadott String értékre.
 - Rule createRule(String): egy megadott szabály név alapján visszaad egy a megadott szabálynak megfelelő szabály (Rule) objektumot.
- MainWindow: ez fogja össze az egész programot és ebben is található meg a main metódus is. Ebben történik meg a szimuláció tényleges indítása, mátrix példány létrehozása stb.:
 - Tagváltozók:
 - CellularAutomaton automaton: ez tárolja magát a mátrixot, egy példánya a CellularAutomaton osztálynak. Alapból 30x30-as mátrixot hoz létre.
 - Timer simulationTimer: egy időzítő, ami arra hivatott hogy a sebesség csúszkán megadott értékeknek megfelelően tárolja a szimuláció gyorsaságát.
 - MatrixPanel matrixPanel: egy MatrixPanel példány, a mátrixhoz tartozó panelt tárolja.
 - ControlPanel controlPanel: egy ControlPanel példányt tárol, ez tárolja a vezérlőpanelhez tartozó objektumokat.
 - boolean isSimulationRunning: ez írja le, hogy a szimuláció fut-e. Alapból false értékű.
 - Metódusok:
 - MatrixPanel(): létrehozza a program fő ablakát, vele együtt az összes szükséges panelt.
 - void runSimulationStep(): ez a metódus frissíti a mátrixot, majd a frissített mátrix alapján újra kirajzolja a mátrixot.
 - void startSimulation(): ennek a feladat az hogy elindítsa a szimulációt., majd a Indítás/Megállítás gombot megváltoztatni.

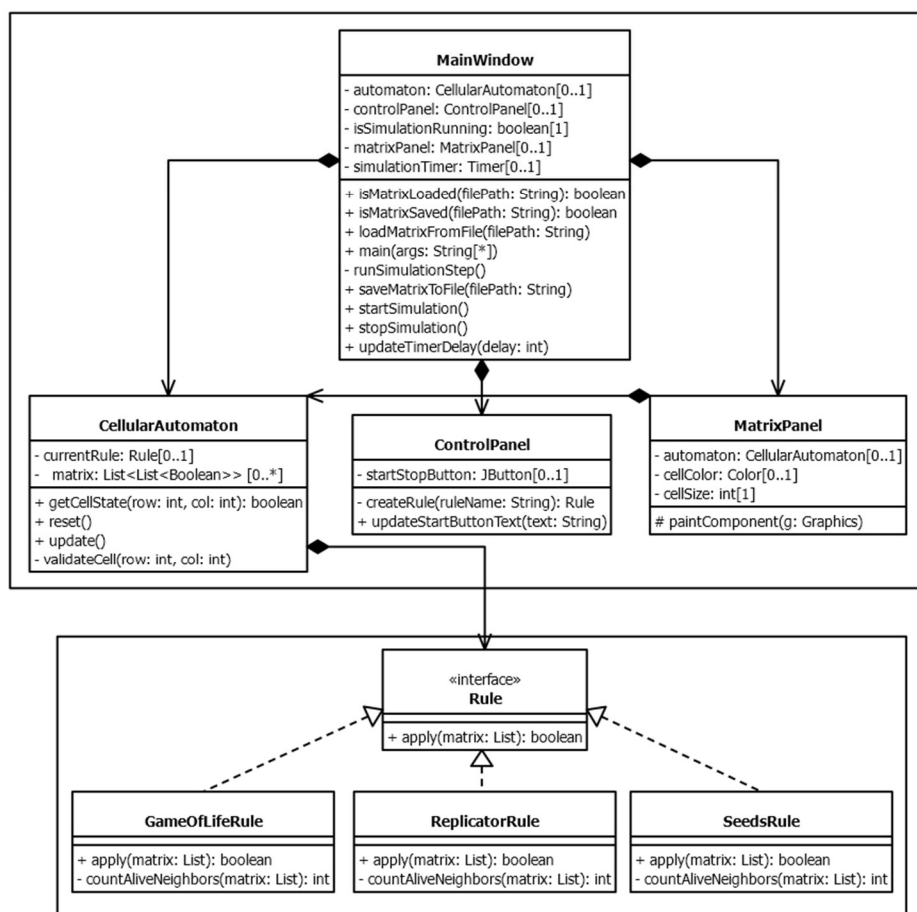
- void stopSimulation(): mint az előző metódus, csak ez megállítja a szimulációt.
- boolean isRunning(): visszaadja boolean értékként hogy a szimuláció fut-e vagy sem.
- void updateTimerDelay(int): beállítja milyen időközönként történjen a mátrix frissítése, az időt egy int paraméterként kapja meg.
- void saveMatrixToFile(String): ez arra szolgál, hogy a paraméterként megkapott helyre kimentí a programban található mátrixot JSON formátumba.
- void loadMatrixFromFile(String): olyan mint az előző metódus, csak fordítva, ez egy JSON formátumú fájlból beolvassa a mátrixot, majd a programban található mátrixot ez alapján frissíti.
- boolean isAnyCellSelected(): megnézi van-e élő sejt a mátrixban, majd a döntés eredményét boolean-ként visszaadja.
- MatrixPanel getMatrixPanel(): visszaadja a programban található MatrixPanel objektum referenciáját.
- int getTimerDelay(): lekérdezi hogy mi az aktuális gyorsaság amin a sebesség csúszkán lett kiválasztva.
- boolean isMatrixSaved(String): ez egy tesztelésnél használt metódus, ami arra szolgál, hogy megnézi megfelelően van-e lementve a mátrix a paraméterként megadott helyére. Egy boolean értéket ad vissza a döntés értelmében.
- boolean isMatrixLoaded(String): hasonló mint az előző, csak a mátrix betöltését vizsgálja meg, hogy megfelelő-e amit az adott helyről (String) töltött be.
- void main(String[]): A program belépési pontja.
- MatrixPanel: a mátrixot tartalmazó panelt és a hozzátartozó viselkedését írja le ez az osztály:
 - Tagváltozók:
 - CellularAutomaton automaton: egy CellularAutomaton példányt tárol.
 - Color cellColor: a mátrixban található sejtek színét tárolja. Alapból a fekete színt tárolja.
 - int cellSize: egy sejt nagyságát tárolja, ami alapból 20x20-as nagyságú.
 - Metódusok:
 - MatrixPanel(CellularAutomaton): ez valósítja meg azt, hogy lehessen kattintani magán a mátrixon és be is színezzé azt a megadott színnek megfelelően.
 - void paintComponent(Graphics): ez a metódus felelős azért, hogy a mátrix összes cellája (sejt) ki legyen színezve a megfelelő színnel.
 - Color getCellColor(): ez lekérdezi az aktuális színt, majd visszaadja azt.
 - void setCellColor(Color): ez egy megadott színre állítja az aktuálisan használt színt.

rules:

- Rule: ez egy interfész, a szabályok általánosítását írja le, ami arra szolgál, hogy az egyetlen metódusát minden ezt az interfészt megvalósító osztály (szabály) a saját szabályainak megfelelően tudja változtatni.

- `boolean apply(List<List<Boolean>>, int, int)`: egy adott szabály alkalmazását végzi el egy megadott mátrix sorában és oszlopában.
- **GameOfLifeRule**: ez az osztály határozza meg a Game of Life szabály működését.
 - `boolean apply(List<List<Boolean>>, int, int)`: a szabálynak megfelelő módosítást végzi a mátrix egy adott sorában és oszlopában.
 - `int countAliveNeighbors(List<List<Boolean>>, int, int)`: megszámolja, hogy az adott mátrix adott sorában és oszlopában található sejtnek a szomszédságában (a középső sejt körüli 8 sejt) hány élő sejt van.
- **ReplicatorRule**: ez az osztály határozza meg a Replicator szabály működését.
 - `boolean apply(List<List<Boolean>>, int, int)`: a szabálynak megfelelő módosítást végzi a mátrix egy adott sorában és oszlopában.
 - `int countAliveNeighbors(List<List<Boolean>>, int, int)`: megszámolja, hogy az adott mátrix adott sorában és oszlopában található sejtnek a szomszédságában (a középső sejt körüli 8 sejt) hány élő sejt van.
- **SeedsRule**: ez az osztály határozza meg a Seeds szabály működését.
 - `boolean apply(List<List<Boolean>>, int, int)`: a szabálynak megfelelő módosítást végzi a mátrix egy adott sorában és oszlopában.
 - `int countAliveNeighbors(List<List<Boolean>>, int, int)`: megszámolja, hogy az adott mátrix adott sorában és oszlopában található sejtnek a szomszédságában (a középső sejt körüli 8 sejt) hány élő sejt van.

Az osztályok viszonyát az alábbi UML osztálydiagram szemlélteti:



Függőségek

A projekten belül a pom.xml fájlban több függőség is meg lett adva, hiszen vannak olyan csomagok, amik fontosak a projekt megfelelő futása érdekében. Maga a projekt Java JDK 22 fordítót vár el, ami a pom.xml fájlban meg is van szabva. A program több plugint is használ, ilyen a maven-jar-plugin ami arra szolgál hogy a JAR file készítése során a megfelelő classpath-t állítsa be. Ezen kívül a JaCoCo plugint is tartalmazza, ami arra szolgál, hogy a projekt Code Coverage-t lehessen vizsgálni, ami a tesztelés folyamán fel is lett használva, hogy minél részletesebb tesztelést lehessen csinálni.

Több függőség is meg lett adva amit a projekt futtatása során automatikusan a Maven letölt. Ilyen függőség a jackson-databind, jackson-core, jackson-annotations amik a JSON fájl formátum kezelést hivatottak kezelni. Ezen kívül a junit-jupiter pedig a JUnit amivel a tesztek lettek elvégezve.

Fájlok kezelése

Több fájlt is kezel a program, ilyen GIF fájl is, amit a program indulásakor a program be is tölt az src/main/resources mappából. Ezen kívül a tesztelésnél a resources mappában lévő valid_matrix.json fájlt is felhasználja. A program fájl betöltést és fájl mentést is megvalósít.

Mentésnél és betöltésnél egyaránt JSON formátumú fájlokat vár és készít, amihez a külső Jackson könyvtárat használja, hogy jól megtörténjen az adatok feldolgozása. Hiba esetén, például nem sikerült a fájl betöltése vagy nem sikerült a fájl mentése azt jelzi és kezeli is a program.





Hibakezelés

A program futása során többször is kell hibát kezelni, hiszen vannak olyan metódusok, amiknek a paraméterei például egy koordinátát határoz meg, ezért le kell kezelni azt, ha rossz koordináta kerül be. Illetve a hibás fájlkezelés esetén is lekezelet a program a hibákat. A vezérlőpanelen levő gomboknál is kezeli a problémákat, bár a Swing biztosítja hogy ilyen ne essen meg, inkább a tesztek során lett jobban ez a funkció felhasználva.

Tesztelés

A projekt teszteléséhez JUnit lett felhasználva, annak is az 5-ös verziója. A Maven szerkezetének megfelelően a teszteléshez használt osztályok az src/test/java mappában találhatóak meg. A tesztek futtatásához az **mvn test** parancsot kell kiadni, ami a Maven segítségével az összes tesztet lefuttatja a projekten. Több funkció tesztelése is implementálva lett, ami a Code Coverage-n látszik is, amihez a JaCoCo plugint használtam:

sejtautomata

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
automatonSimulation		90%		72%	30	90	35	259	6	40	0	5
rules		97%		91%	5	43	0	41	0	9	0	3
Total	123 of 1 467	91%	33 of 166	80%	35	133	35	300	6	49	0	8

Mindkét csomagon átlagosan legalább 80%-os Code Coverage el lett érve. Ebbe beleszámít az is, hogy a Swing komponensei nem minden esetben lett letesztelve, mert azt feltételezések alapján működő képesen kell működnie.

A teszteléshez az alábbi tesztelési osztályok lettek implementálva:

- CellularAutomatonTest: ez a teszt osztály a CellularAutomaton osztály egyes metódusait vizsgálja meg teszteli le.
- ControlPanelTest: ez a ControlPanel osztály működését teszteli le, azon belül is főleg az osztályhoz tartozó metódusokat.
- GameOfLifeRuleTest: ez az egyik szabály, a GameOfLifeRule-hoz tartozó metódusokat teszteli le, hogy megfelelően működik-e a szabály logikája.
- MainWindowTest: ez a MainWindow osztály tesztelését végzi el, annak metódusain főleg.
- MatrixPanelTest: ez a MatrixPanel osztályon végzi el a szükséges teszteket.
- ReplicatorRuleTest: ez a Replicator szabályhoz tartozó ReplicatorRule osztályra végzi el a szükséges teszteket.
- SeedsRuleTest: ez pedig a Seeds szabályhoz tartozó SeedsRule osztályhoz végzi el az egyes teszteket.

A tesztelés folyamán több ablak is megjelenik, annak érdekében, hogy mindegyik teszt egymástól függetlenül működjön és lehessen tesztelni az egyes funkciókat. Összesen 26 teszt lett definiálva, amit a projekt végén lefuttatva sikeresnek nyilvánított a Maven.