Életjáték

Programozás alapjai 3
Nagy házi feladat

Heizer Levente
IT9P0Z

Tartalomjegyzék

1. A feladat

A Programozás alapjai 3. tárgy nagy házi feladatának egy sejtautomata megvalósítását választottam, amelynek az alapötletét a Conway-féle *Életjáték* adta. A modell lényege, hogy egy NxN-es négyzetháló közrefog egy úgynevezett sejtteret. A sejttérben egy cella testesít meg egy-egy sejtet, amiknek két állapota lehet: élő vagy halott.
Ahogy az idő telik ezek a sejtek valamilyen szabály alapján változtatják az állapotukat. Ezt a szabályt az *Életjáték*ban a sejteket körbe vevő szomszédok száma határozza meg, így létrehozva különböző mintákat a rácson.

A feladat megoldása során a következőket kellett megvalósítani:

1. Könnyen kezelhető GUI a kezdőállapotok beállítására
2. Állapot elmentése és visszatöltése
3. Szimuláció megállítása és újraindítása
4. Gyűjtemény keretrendszer használata
5. Tesztelés-támogatás

A projektet *MVC architektúra* szerint csináltam, így *use-case*-ek és a user-manual ismertetése után, a dokumentációban a modell egyes rétegein keresztül fogom bemutatni az osztályokat.

2. Use-case-ek leírása

Cím	**Init simulation**
Leírás	A szimuláció beállítása
Aktorok	Felhasználó
Forgatókönyv	1. Random kezdőállapot generálása 1.1. Hányszor hányas legyen 1.2. Mekkora valószínűséggel legyen egy sejt élő 2. B../S.. szabály beállítása (Game Of Life, Flakes, Custom, stb.)
Alternatív forgatókönyv	1. A) Már korábban elmentett állapot betöltése

Cím	**Control simulation**
Leírás	A felhasználó szabályozza a szimulációt
Aktorok	Felhasználó
Forgatókönyv	1. Szimuláció elindítása 2. Szimuláció sebességének szabályozása 3. Szimuláció megállítás
Alternatív forgatókönyv	2. A) Másik szabály választása 3. A) Állapot elmentése 3. B) Kilépés a programból

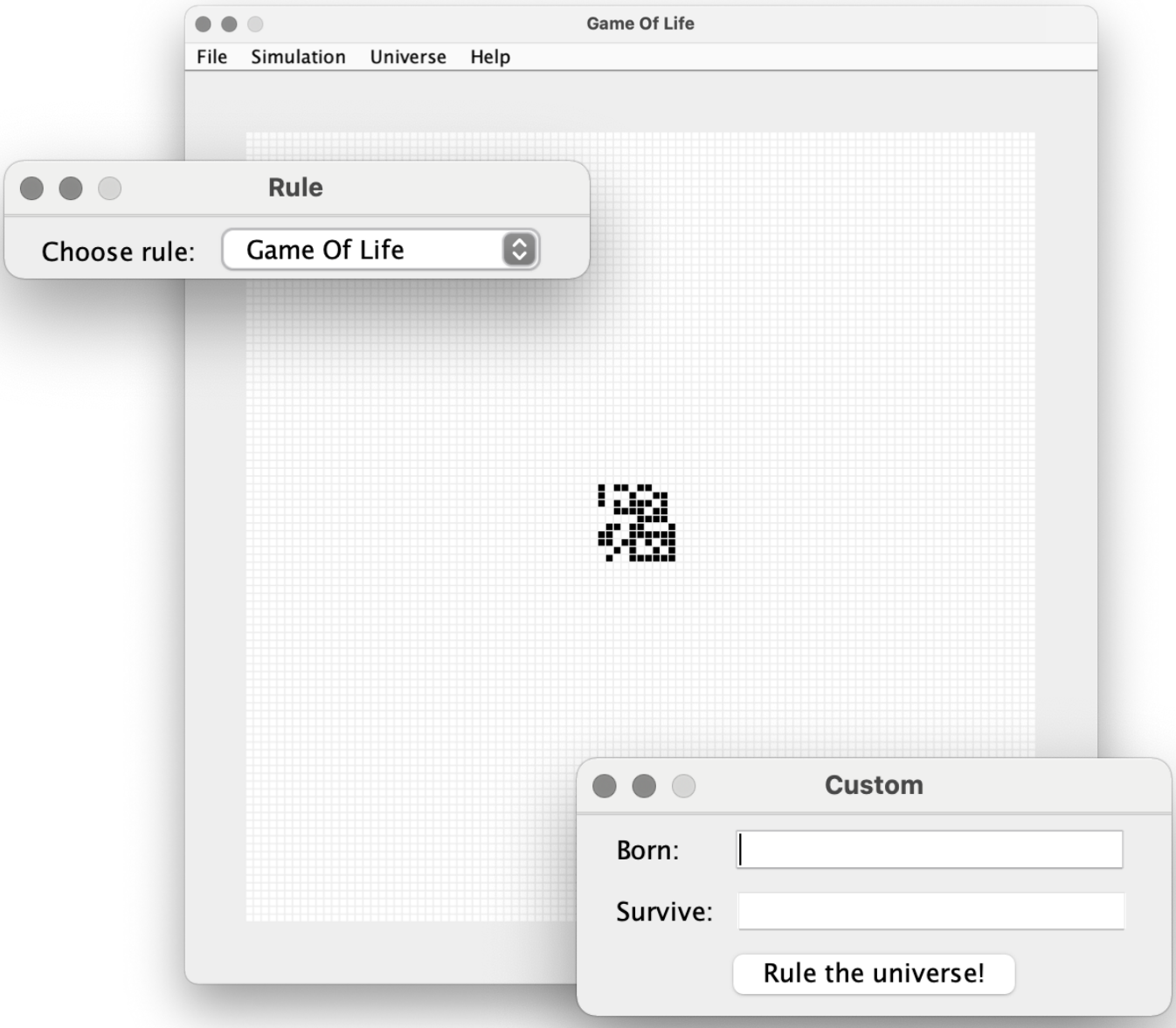
Cím	**View simulation**
Leírás	A felhasználó nézi a szimulációt
Aktorok	Felhasználó
Forgatókönyv	1. A rendszer kirajzolja a sejttér aktuális állapotát. 2. A felhasználó megtekinti a sejttér aktuális állapotát.

Cím	**Control app**
Leírás	A szimuláció kezelése a felhasználó beállításai alapján.
Aktorok	Controller
Forgatókönyv	1. A felhasználói beállítások alapján a sejttér kezdőállapotának inicializálása. 2. A szimuláció futtatása.
Alternatív forgatókönyv	2. A) A felhasználói utasítások érvényesítése a sejtteren az Action Listener-ek segítségével.

3. User-manuál

A program elindítását követően a képen látható ablak ugrik fel. A menüt a következők szerint strukturáltam:

- **File:** állapot mentése és betöltése
- **Simulation:** szimuláció elindítása, megállítása és sebességének szabályozása
- **Universe:** szabály beállítása, kezdő állapot beállítása és törlése a rácsról
- **Help:** leírások



Az egyes beállításokat legördülő menü segítségével lehet beállítani. Illetve a szabályoknál van egy *Custom* lehetőség is, amelynek választásakor a felhasználó egy egyedi számsorozat megadásával adhatja meg, hogy milyen esetekben éledjenek újra, illetve halljanak meg az egyes sejtek. A számsorozatokat pl. a következő formában kell megadni: 0,1,2,3,4

4. Osztályok leírása

Modell réteg

Universe

Felelősségek

Az univerzum állapotának és a generálás szabályainak tárolása és az új generáció létrehozásának megvalósítása.

Attribútumok

- **currentGen** : Boolean[][]	Az aktuális generáció állapotát tárolja, a logikai érték igaz és hamis állapota jelzi azt, hogy a sejt él vagy halott.
- **nextGen** : Boolean[][]	Az következő generáció állapotát tárolja, a logikai érték igaz és hamis állapota jelzi azt, hogy a sejt él vagy halott.
- **rule** : Rule	A sejtek életben maradásának, illetve születésének szabályát határozza meg az rendszerben.

Metódusok

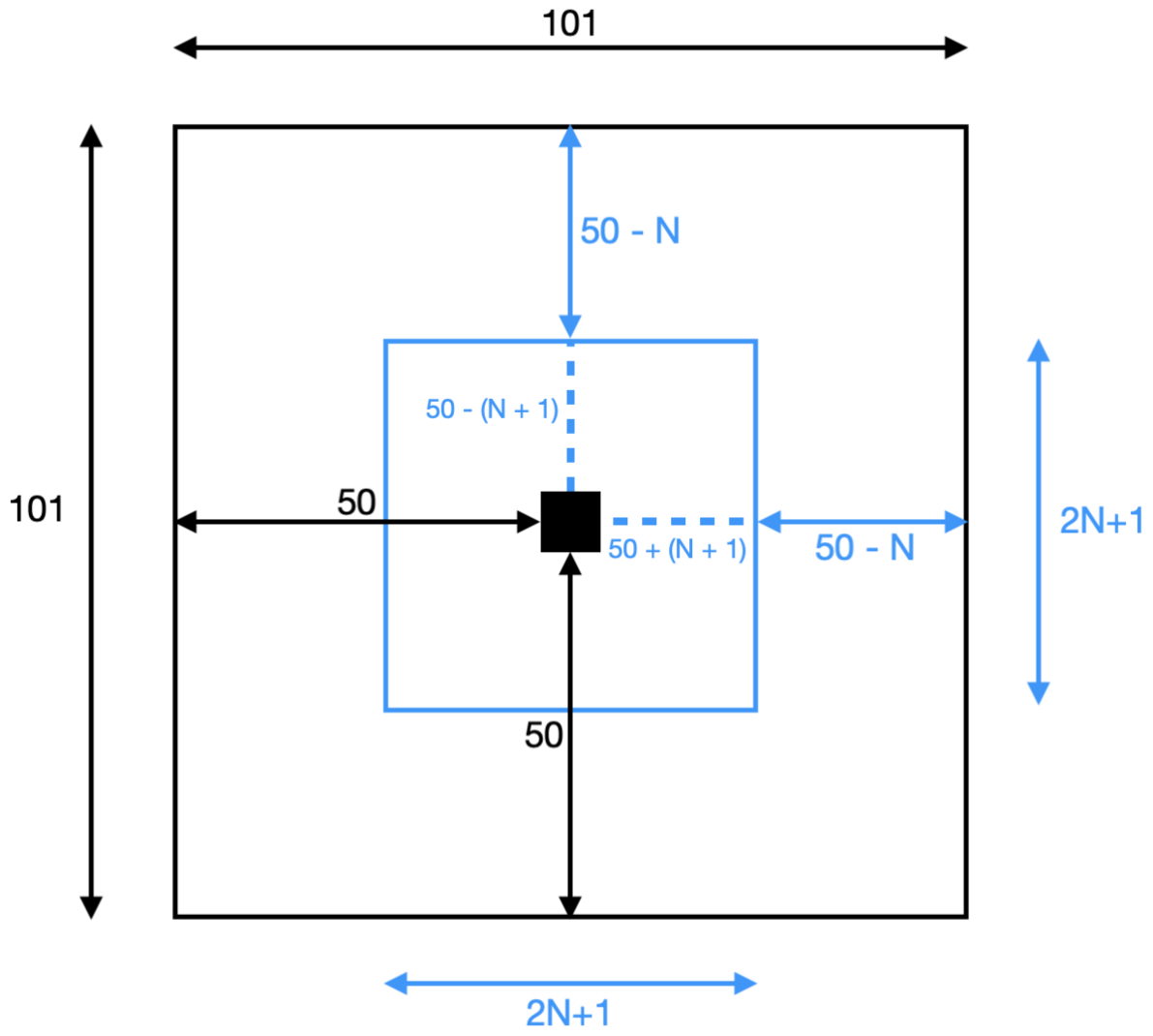
+ **Universe**(S : int, born : int[], survive : int[])	A Universe osztály konstruktora.
+ **getRule**() : Rule	A rule attribútum getter függvénye.
+ **setRule**(b : int[], s : int[])	A rule attribútum setter függvénye.
+ **setCell**(i : int, j : int)	Egy adott cella értékét állítja be.
+ **getCurrentGen**() : Boolean[][]	A currentGen attribútum getter függvénye.
+ **setCurrentGen**(state : Boolean[][])	A currentGen attribútum setter függvénye.
+ **getNextGen**() : Boolean[][]	A nextGen attribútum getter függvénye.
+ **clearUniverse**()	A currentGen attribútum összes cellájának értékét false-ra állítja.
+ **getSize**() : int	Függvény, ami visszaadja a currentGen ArrayList méretét.
+ **countNeighbours**(i : int, j : int) : int	Függvény, ami megszámolja egy adott cella élő szomszédainak számát.
+ **AliveCells**() : int	Függvény, ami visszaadja a currentGen élő celláinak számát.
+ **evolve**()	A rule attribútumban definiált szabály alapján konstruálja a következő generációt és felülírja a currentGen attribútumot.

+ **AutoFill**(probability : int, size : int)	A bemeneti paraméterként megadott értékék alapján generál egy véletlen állapotot a currentGen attribútumnak. probability: mekkora valószínűséggel legyen egy cella élő size: mekkora legyen a sejttérben generált rész

Megjegyzések

A **`currentGen`** és **`nextGen`** attribútumoknál a *`Boolean[][]`* típusmegjelölést csak helytakarékosság céljából használtam. A kódolás során a megnevezett attribútumoknak *`ArrayList<ArrayList<Boolean>>`* típust választottam, mivel az állapotok mentésénél és betöltésénél szerializációt terveztem használni.

A konstruktorban és az **AutoFill** metódusban lévő **`if (i > (50 - (N + 1)) && i < (50 + (N + 1)) && j > (50 - (N + 1)) && j < (50 + (N + 1)))`** feltétel azt teszi lehetővé, hogy a sejttérben lévő kezdő állapot a sejttér közepére kerüljön.



Rule

Felelősségek

Az születés és az életben maradás szabályainak tárolása és a szabályok érvényesítése.

Attribútumok

- **born** : int[]	Eltárolja, hogy milyen szomszédszámok esetén születik meg egy halott sejt.
- **survive** : int[]	Eltárolja, hogy milyen szomszédszámok esetén marad életben egy élő sejt.

Metódusok

+ **getBorn**() : int[]	A born attribútum getter függvénye.
+ **setBorn**(born : int[])	A bor attribútum setter függvénye.
+ **getSurvive**() : int[]	A survive attribútum getter függvénye.
+ **setSurvive**(survive : int[])	A survive attribútum setter függvénye.
+ **willBorn**(neighbours : int) : boolean	Függvény, ami ellenőrzi, hogy egy paraméterként megadott szomszédszámot tartalmaz-e a bor array.
+ **willSurvive**(neighbours : int) : boolean	Függvény, ami ellenőrzi, hogy egy paraméterként megadott szomszédszámot tartalmaz-e a survive array.

Nézet réteg

GameOfLifeFrame

Felelősségek

A program grafikus felületének, menürendszerének és szimuláció grafikus megjelenítésének megvalósítása.

Attribútumok

- **game_controller** : GameOfLifeController	Kontroller a sejttér állapotának szabályozásához.
- **panel** : GameOfLifePanel	JPanel, ami a sejtteret valósítja meg a kezelőfelületen.
- **simulator** : Thread	A panel futását egy szálba foglalja.

Metódusok

+ **GameOfLifeFrame**(GameOfLifeController panel)	A GameOfLifeFrame konstruktora, ami létrehozza kezelőfelület menüjét és a panelt.
+ **getPanel**() : GameOfLifePanel	A GameOfLifePanel getter függvénye.

GameOfLifeFrame belső osztályai

- **StartActionListener**	A szimulációt lehet elindítani vele.
- **StopActionListener**	A szimulációt lehet megállítani vele.
- **SaveActionListener**	A szimuláció aktuális állapotát lehet elmenteni vele.
- **LoadActionListener**	Egy elmentett állapotot tölt be és jelenít meg a panelen.
- **AutofillActionListener**	Egy random kezdőállapotot hoz létre és jelenít meg a panelen.
- **ClearActionListener**	Az összes élő cellát halottra állítja.
- **SetRuleActionListener**	A sejtek születésének és életben maradásának szabályait lehet vele beállítani.
- **SetSpeedActionListener**	A szimuláció sebességét lehet vele szabályozni.
- **RulesLexiconListener**	A házi feladat leírásánál megadott szabályokat leíró honlapra hivatkozik.
- **CellularActionListener**	A házi feladat leírásánál megadott sejtautomatát ismertető wikipédia cikkre hivatkozik.

Megjegyzések

Az egyedi szabály beállításakor, ha a felhasználó nem 0 és 8 közötti számokat ad meg, akkor a program jelez, hogy rossz az input.

Az automatikus kitöltéssel és szabályok beállításával kapcsolatos belső osztályokról a következő fejezetben van szó: Action Listener-ek

GameOfLifePanel

Felelősségek

A sejttér grafikus megjelenítése az idő elteltével.

Attribútumok

- **cells** : JPanel[][]	JPanel-ekből álló 2D-s array a sejtek élő, illetve halott állapotának grafikus megjelenítéséhez. Ha egy sejt élő akkor, a panel fekete, ha halott, akkor fehér.
- **game_controller** : GameOfLifeController	Kontroller a sejttér állapotának szabályozásához.
- **size** : int	A kontrollerben eltárolt univerzum méretét tárolja.

Metódusok

+ **GameOfLifePanel**(GameOfLifeController controller)	GameOfLifePanel konstruktora.
+ **initUniverse**()	A panelt inicializálja.
+ **drawUniverse**()	Kirajzolja a panelra az univerzum aktuális állapotát.
+ **clearPanel**()	Törli a panelről az élő sejteket.
+ **run**()	A sejttér változását valósítja meg az idő elteltével a panelen.
+ **getCells**() : JPanel[][]	A cells attribútum getter függvénye.

Megjegyzések

Az **`overWriteUniverse`** metódusban az **`uni`** paraméternek *`ArrayList<ArrayList<Boolean>>`* a típusa.

Vezérlő réteg

GameOfLifeApp

Felelősségek

A Game Of Life app indítása.

Metódusok

+ **main**(args : String[])	

GameOfLifeController

Felelősségek

Az sejttér állapotának vezérlése. Kapcsolat kialakítása a modell és a grafikus megjelenítés között.

Attribútumok

- **size** : int	Az eltárolt univerzum mérete.
- **speed** : long	A szimuláció sebessége.
- **born** : int[]	Azok a szomszédszámok, amik esetén a halott sejt újraéled.
- **survive** : int[]	Azok a szomszédszámok, amik esetén az élő cella életben marad.
- **universe** : Universe	Referencia az univerzumra.

Metódusok

+ **getUniverseSize**() : int	Az univerzum méretének getter függvénye.
+ **getUniverse**() : Universe	Az univerzum getter függvénye.
+ **getUniverseArray**() : Boolean[][]	Az univerzum actualGen attribútumának elérése.
+ **setSpeed**(s : long)	A speed attribútum setter függvénye.
+ **getSpeed**() : long	A speed attribútum getter függvénye.
+ **isAlive**(i: int, j: int) : Boolean	Megadja, hogy az (i, j) sejt életben van-e.
+ **aliveCells**() : int	Megadja az aktuális állapot élő sejtjeinek számát.
+ **generateNextState**()	Meghívja a *Universe* osztály evolve metódusát, ezzel egy új állapotot generálva.

Megjegyzések

Az **`getUniverseArray`** metódusban az **`uni`** paraméternek *`ArrayList<ArrayList<Boolean>>`* a típusa.

StateIO

Felelősségek

A sejttér állapotának mentése és a már mentátt állapot visszatöltése.

Attribútumok

- **CSV_SEPARATOR** : String	Segédváltozó, ami meghatározza, hogy mi *.csv fájlok szeparátora.

Metódusok

+ **writeToCSV**(universe : Boolean[][])	Statikus függvény, ami ArrayList tartalmát egy *actualstate.csv* fájlba ír.
+ **readFromCSV**(file : String) : Boolean[][]	Statikus függvény, ami ArrayList-et ad vissza, aminek tartalmát a paraméterként megadott fájlból beolvasott adat szerint inicializál.

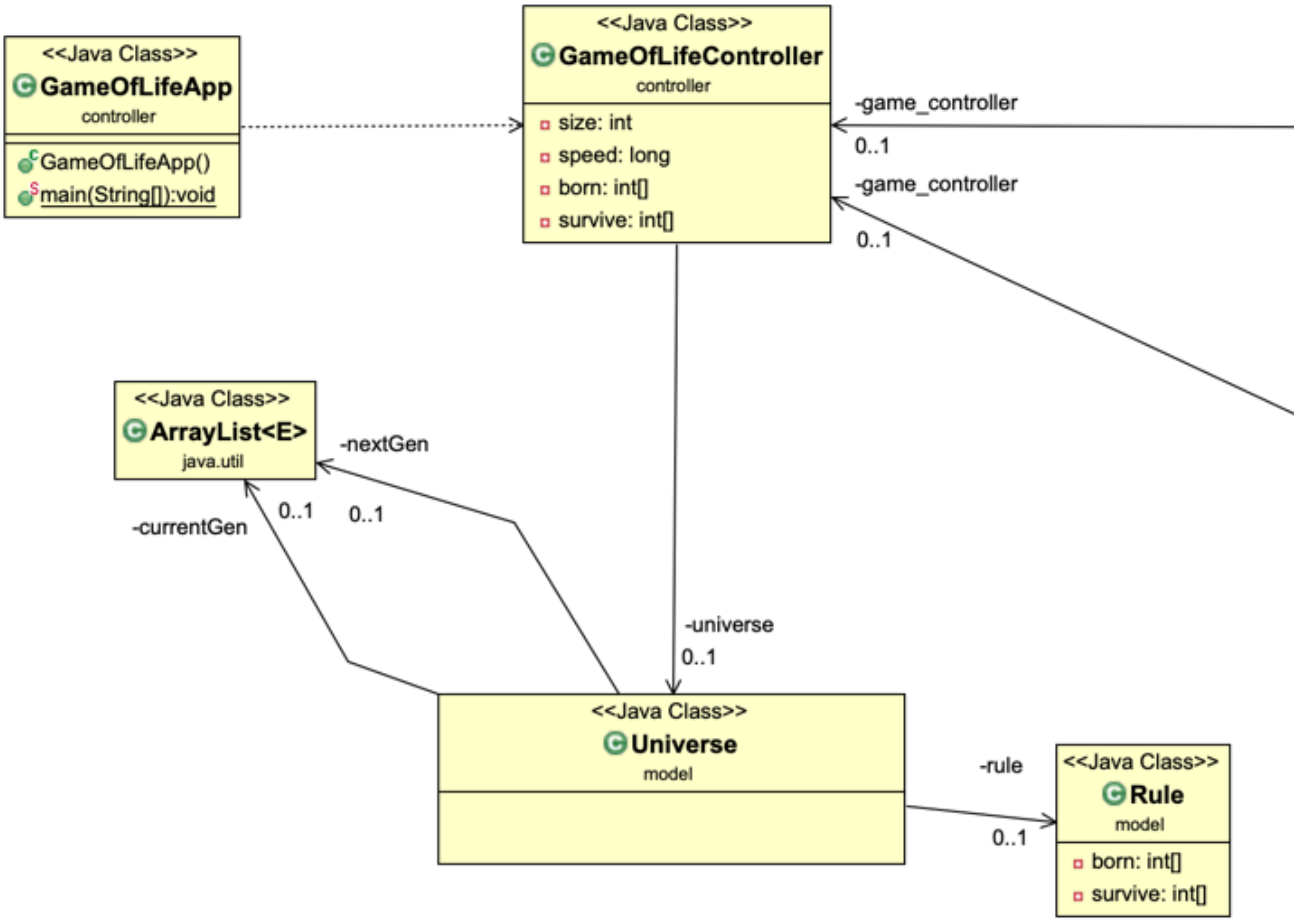
Megjegyzések

A sejttér állapotának mentése `*.csv` kiterjesztésű fájlokba történik. Amiben a sejtek aktuális állapotát (élő vagy halott) 1 vagy 0 értékű bitek reprezentálják.

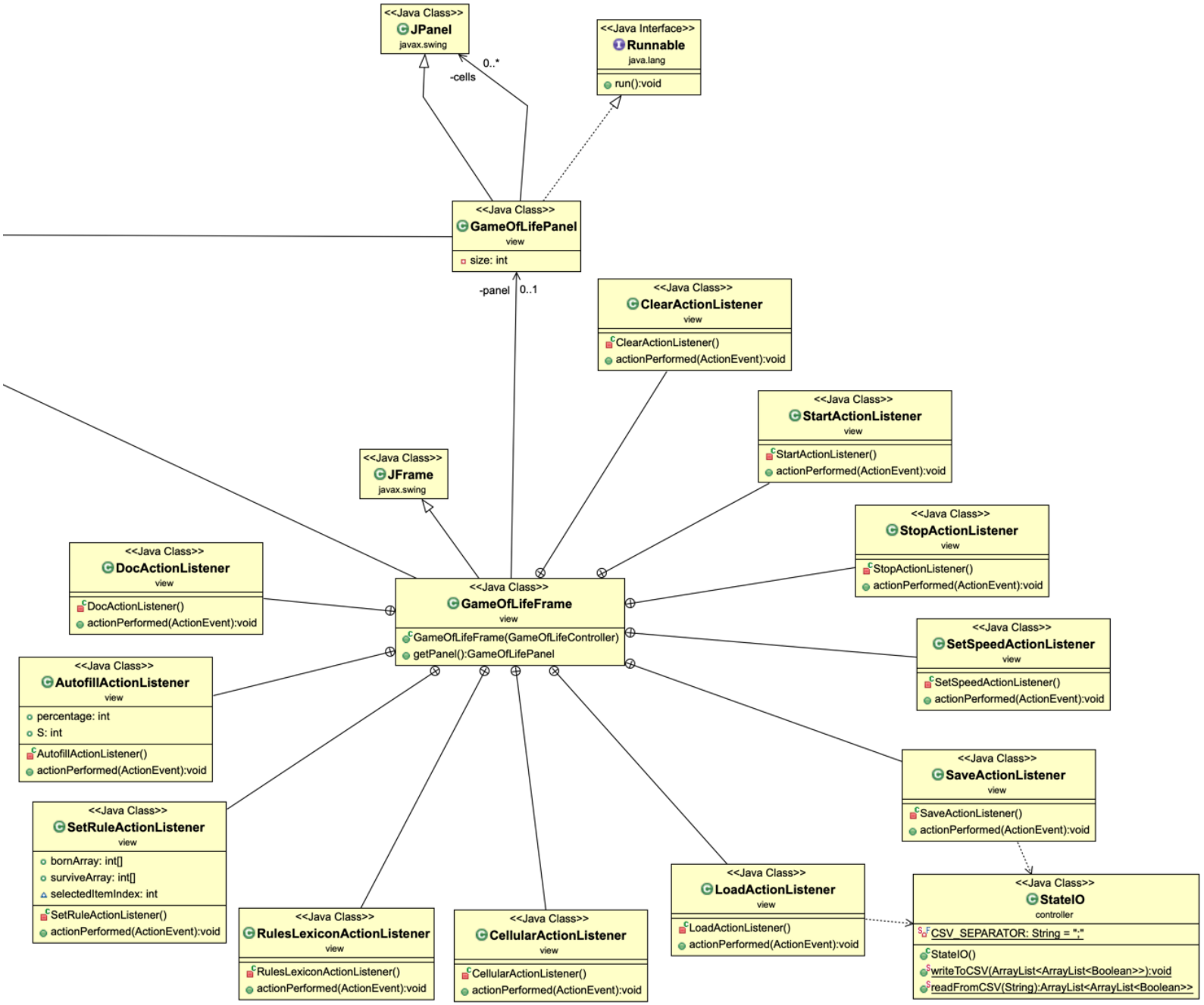
Illetve a **`writeToCSV`** és a **`readFromCSV`** metódusokban a bemeneti és a return paraméterek `ArrayList<ArrayList<Boolean>>` tipusúak.

5. Osztálydiagram

Az osztálydiagrammot kódból generáltam *Object Aid UML Explorer*[1] segítségével. Néhány függőséget eltávolítottam az átláthatóság érdekében, de a tömörített fájlban a teljes osztálydiagram is megtalálható.



[1] https://www.objectaid.com/home

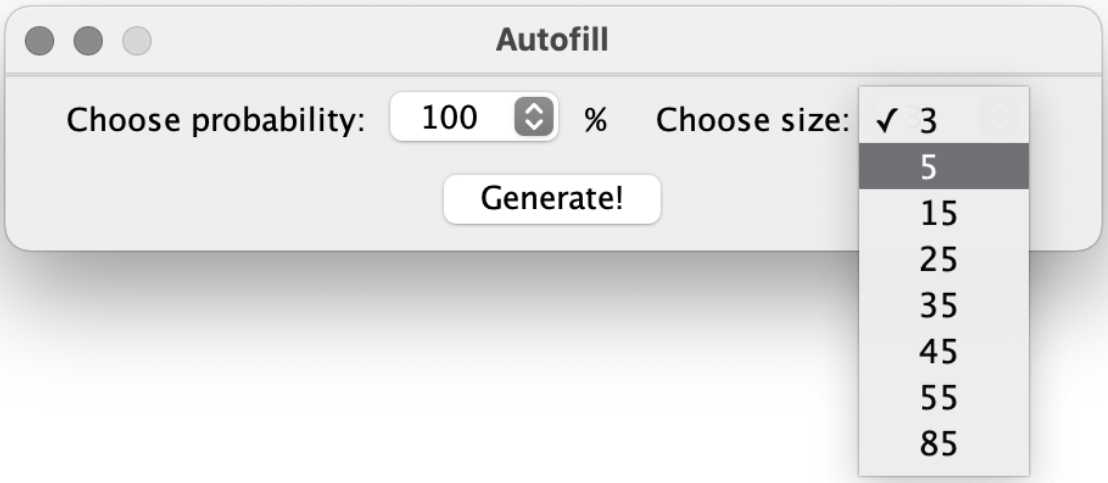

<<Java Class>>
JPanel
javax.swing
<<Java Interface>>
Runnable
java.lang
run():void
0..*
-cells
<<Java Class>>
GameOfLifePanel
view
size: int
-panel
0..1
<<Java Class>>
ClearActionListener
view
ClearActionListener()
actionPerformed(ActionEvent):void
<<Java Class>>
StartActionListener
view
StartActionListener()
actionPerformed(ActionEvent):void
<<Java Class>>
JFrame
javax.swing
<<Java Class>>
StopActionListener
view
StopActionListener()
actionPerformed(ActionEvent):void
<<Java Class>>
DocActionListener
view
DocActionListener()
actionPerformed(ActionEvent):void
<<Java Class>>
GameOfLifeFrame
view
GameOfLifeFrame(GameOfLifeController)
getPanel():GameOfLifePanel
<<Java Class>>
SetSpeedActionListener
view
SetSpeedActionListener()
actionPerformed(ActionEvent):void
<<Java Class>>
AutofillActionListener
view
percentage: int
S: int
AutofillActionListener()
actionPerformed(ActionEvent):void
<<Java Class>>
SaveActionListener
view
SaveActionListener()
actionPerformed(ActionEvent):void
<<Java Class>>
SetRuleActionListener
view
bornArray: int[]
surviveArray: int[]
selectedItemIndex: int
SetRuleActionListener()
actionPerformed(ActionEvent):void
<<Java Class>>
LoadActionListener
view
LoadActionListener()
actionPerformed(ActionEvent):void
<<Java Class>>
StateIO
controller
CSV_SEPARATOR: String = ";"
StateIO()
writeToCSV(ArrayList<ArrayList<Boolean>>):void
readFromCSV(String):ArrayList<ArrayList<Boolean>>
<<Java Class>>
RulesLexiconActionListener
view
RulesLexiconActionListener()
actionPerformed(ActionEvent):void
<<Java Class>>
CellularActionListener
view
CellularActionListener()
actionPerformed(ActionEvent):void

6. Action Listener-ek

Ezt a két GameOfLifeFramehez tartozó ActionListener-t, azért emelem ki, mert ezek működéséhez további ActionListener-ek implementálásra volt szükség.

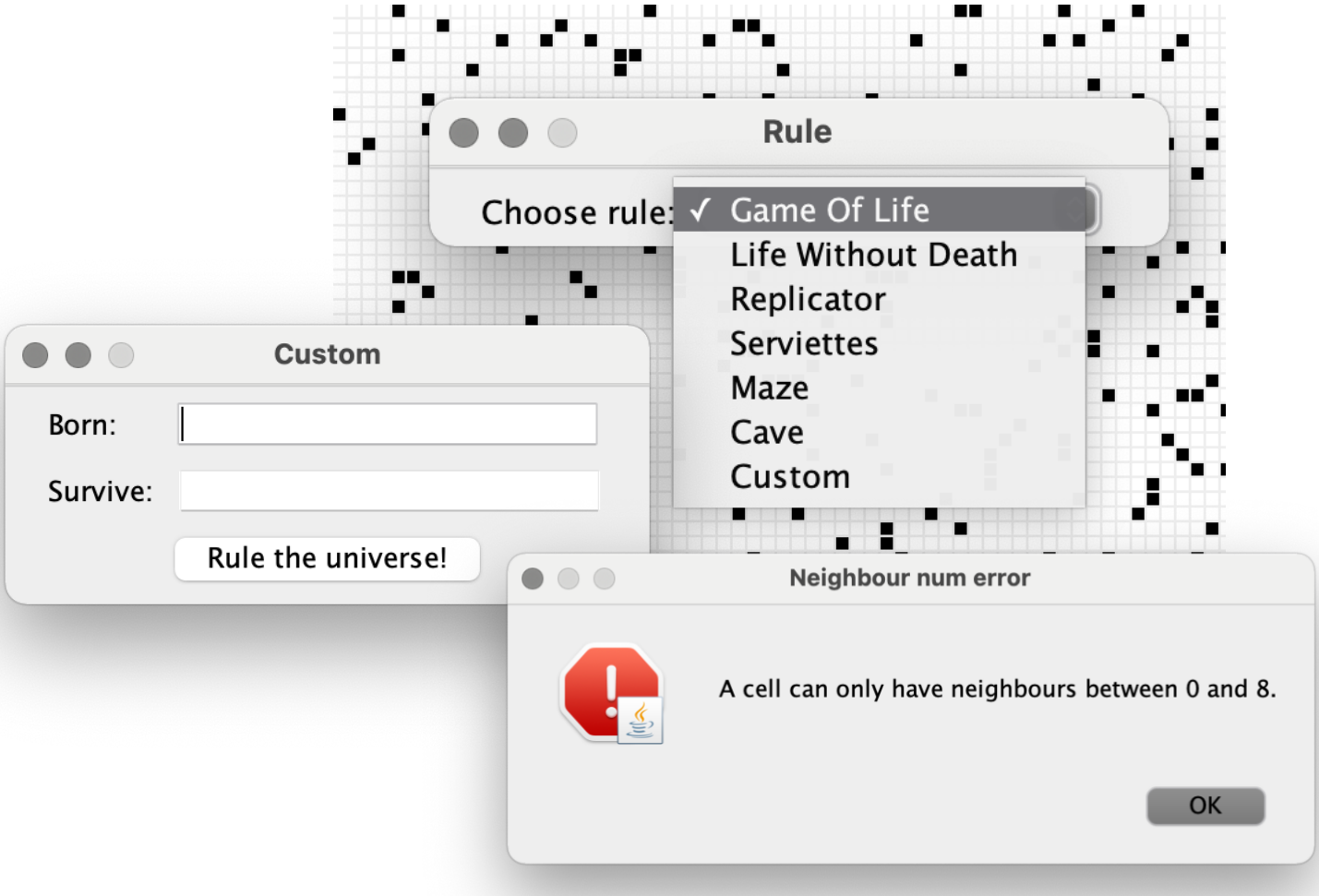
AutofillActionListener

Ennek a funkciónak az a lényege, hogy lehessen valamilyen kezdőállapotot generálni a szimulációhoz. Itt két JComboBox-on keresztül be lehet állítani, hogy egy cella milyen valószínűséggel legyen élő, illetve hogy a kezdő állapot mekkora részt foglaljon el a sejttérből. Ha ezek a beállítások megvannak, akkor a Generate! JButton érvényesíti a kiválasztott beállításokat.



SetRuleActionListener

Az sejtek életben maradásának és születésének szabályait beállító ActionListener-ben is egy JComboBox által megvalósított legördülő menüből lehet kiválasztani a szabályt. Itt van egy olyan lehetőség is, amivel egyedi szabályokat lehet meghatározni. Itt figyelembe kell venni, hogy a sejttérben egy sejtnek maximum 8 szomszédja lehet. Így, ha a lehetséges szomszédszámok megadásánál olyan számot adunk meg, ami nem esik bele a [0, 8] zárt intervallumba a felhasználónak egy hibaüzenet jelzi, hogy rosszul adta meg a szabályokat.



7. Fontosabb algoritmusok

Az új generáció generálása

A sejttér topológiai szempontból egy tórusz, ezért a szomszédok számítása során a 2D-s tömb szélein elhelyezkedő sejteknek az ellentétes oldalak szélein lévő sejtek is a szomszédaik. Ezt a **Universe** osztály **countNeighbours** metódusában figyelembe kellett venni.

Az új generáció létrehozását a **evolve** metódus végzi. Ez végigfut az **actualGen** attribútum sorain, ahol a következők mennek végbe:

1. Létrehoz egy új ArrayList-et a **nextGen** attribútumban.
2. Egy for ciklussal végig fut a cellákon, ahol minden cellának megszámolja a szomszédjait, majd attól függően, hogy az aktuális sejtet reprezentáló elemnek igaz vagy hamis értéke van a következő két eset szerint, ad hozzá igaz és hamis értékeket a **nextGen** tömbhöz:
    a. Ha a sejt él, akkor a **Rule** attribútumban definiált szabályok szerint ellenőrzi, hogy a szomszédszám benne van-e a **survive** tömbben. Ha igen sejt életben marad, ha nem a sejt meghal.
    b. Ha a sejt nem él, akkor azt ellenőzi, hogy a szomszédszám benne van a **born** tömbben, ha igen, akkor a sejt életre kell, ha nem, akkor sejt élettelen marad.
3. A **nextGen** felülírja az **actualGen** tömböt.

A GameOfLifePanel run metódusa

A **GameOfLifePanel** delegációval hoztam létre a `Runnable`-ből, ezért implementálni kellet hozzá az az interfész `run` metódusát. A programban ez a `run` metódus rekurzióval teszi lehetővé, hogy a szimulációban a sejttér állapota folyamatosan változzon.

A metódus futása során a következők történnek:

1. Univerzum következő állapotának generálása.
2. Az új állapot kirajzolása a panelra.
3. Szálkezeléssel késleltet, annyi ms-nyit, amennyit a felhasználó beállított.
4. Rekurzió: `run` újra hívása, ha van még elő sejt.