2021.11.25.

ÉLET JÁTÉK

DOKUMENTÁCIÓ

RÓNAI LILI

WAQO6I

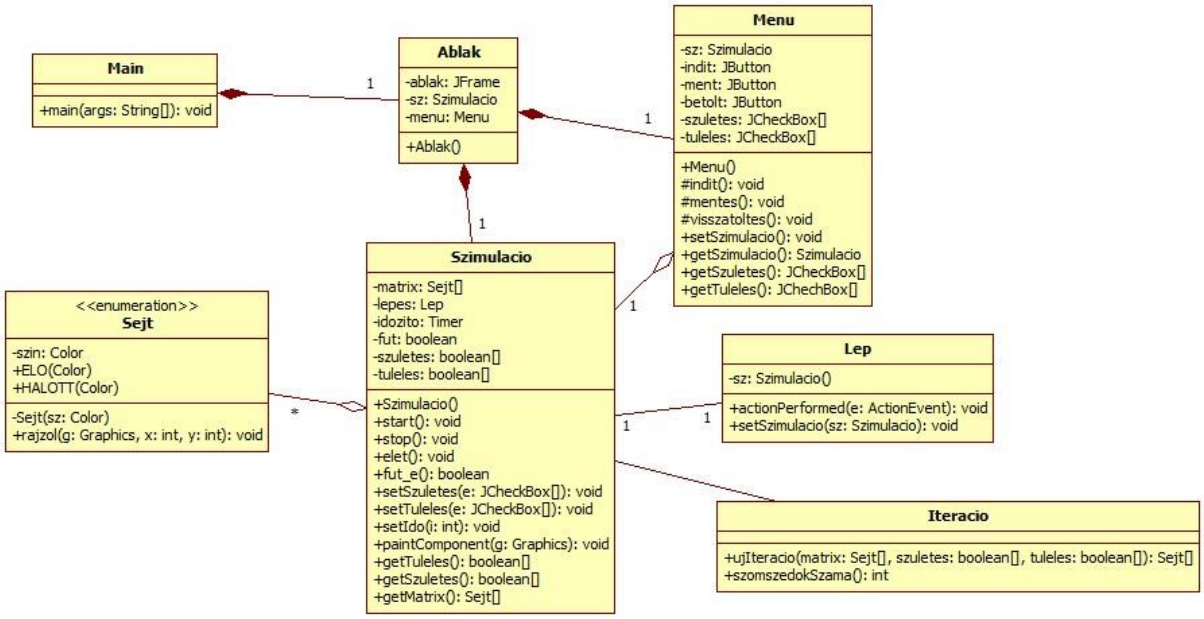
DOKUMENTÁCIÓ

Az életjátékot John Conway találta ki. Ez a legismertebb sejtautomata. Ezek az automaták úgy működnek, hogy egy 2D rács minden mezője egy adott állapotban van (élő vagy halott), majd minden egyes iterációban a környezettől függően változik a cellák állapota. A négyzetrács mezőit celláknak nevezzük. Egy cella környezete a hozzá legközelebb eső 8 mező. Egy cella szomszédjai a környezetében lévő sejtek. A kezdő állapotban random dől el, hogy melyik cellák élők, vagy üresek. Ezt követően körönként változik az állapotuk a celláknak. A Conway-féle klasszikus Game of Life szerint a következő iterációban azok a cellák maradnak életben, amelyeknek 2 vagy 3 élő szomszédja van, és azok az üres cellákban kelnek életre, amelyeknek pontosan 3 szomszédja van.

OSZTÁLYOK

Osztály	Felelősség
Main	Az ablak osztályt hívja.
Ablak	A JFramet hozza létre, illetve a menüt, és a szimulációt (ami később megjeleníti a táblát).
Menu	A gombok, textfield, és checkboxokat jeleníti meg, illetve a gombnyomásra történő dolgok függvényeit hívja.
Szimulacio	Tárolja a sejtek kettő dimenziós tömbjét, itt van implementálva a függvények nagyrésze, amik a menüben vannak meghívva. Illetve az élő és halott sejtekből alkotott két dimenziós tömb (mátrix) kirajzolásáért felel.
Lep	Az ActionListener interfészt implementálja, és meghívja a szimulacio elet függvényét. A szimuláció létrehozásával a timer egy Lep actionlistenert kap második paraméterként.
Iteracio	A felhasználó által beállított szabályok felhasználásával feladata, hogy meghatározza mely sejtek lesznek élők illetve halottak a következő iterációban.
Sejt	Ez egy enum osztály. ELO, vagy HALOTT lehet. Tárolja, hogy mely esetben milyen színű egy sejt, illetve egy sejt kirajzolásáért felel.

OSZTÁLY DIAGRAMM



FÜGGVÉNYEK

ABLAK

public Ablak()

Létrehoz egy ablakot, egy szimuláció illetve menü panelt. Az új ablak neve Game of Life lesz, ami a Bezárás gomb megnyomására bezáródik, illetve nem lehet átméretezni.
Beállítja a tábla méretét, amit a sejtek kirajzolnak. A sejtek az ablak alsó részén helyezkednek majd el. A menü pedig az ablak felső részében kap helyet.

MENU

public menu()

A gombok, textfield, és checkboxokat jeleníti meg, illetve a gombnyomásra történő dolgok függvényeit hívja.

public setSzimulacio(Szimulacio sz)

Beállítja a paraméterként kapott szimulációt.

public Szimulacio getSzimulacio()

Visszaadja a szimulacio attribútum értékét.

public JCheckBox[] getSzuletes()

Visszaadja a szuletes tömböt.

public JCheckBox[] getTuleles()

Visszaadja a tuleles tömböt.

protected void indit()

A start/stop gomb megnyomására, ha a szimuláció fut már, akkor megállítja azt. Ha a start/stop gomb megnyomásakor a szimuláció nem fut még, akkor elindítja a menü konstruktora azt

protected void mentes() throws IOException

Szerializálja a mátrixot. A felhasználói könyvtárban az allapot.dat fájlba szeretném tárolni, ha ez a fájl nem létezik, akkor először létrehozom, majd az előadáson tanult módon szeriazálom.

protected void visszatoltes()

Visszatölti a korábban szerializált mátrixot. A felhasználói könyvtárban az allapot.dat fájlból töltjük vissza, ha létezik a fájl, akkor az előadáson tanult módon megoldom a szerializált mátrix visszatöltését. Előfordulhat, hogy nem létezik a fájl, de csak akkor, ha korábban nem volt mentve semmi, és ebben az esetben nincs is mit visszatölteni, szóval nem foglalkozom ezzel a helyzettel.

SEJT

A sejt enumnak két konstans értéke van, élő (amihez egy lila szín tartozik), vagy halott (ami pedig szürke).

private Sejt(Color sz)

A sejt konstruktora nincs meghívva, csak azért van rá szükség, hogy a színeket a konstansokhoz rendelhessem.

public void rajzol(Graphics g, int x, int y)

A paraméterként kapott adatokkal megrajzol egy sejtet, ahol x a mátrixban a sor száma és y a mátrixban az oszlop száma.

SZIMULACIO

public Szimulacio()

A konstruktor feltölti a mátrixot random élő vagy halott sejtekkel. A lepes attribútumnak meg kell kapnia a szimulációt annak létrejöttekor. 400x200 sejt jelenik meg a táblán. Ha a 0 és 1 között random sorsolt szám 0.5 feletti, akkor az adott sejt élő lesz, egyébként halott.

public void start()

Elindítja a timert. A fut boolean-t true-ra állítja, hiszen már elindult a szimuláció.

public void stop()

Megállítja a timert. A fut boolean-t false-ra állítja, hiszen már megállt a szimuláció.

public void elet()

Az iteráció metódusa kiszámolja, hogy a következő iterációban hogyan kell kinéznie a mátrixnak. Szóval a visszatérési értékét állítom be a matrixnak.

public boolean fut_e()

Visszaadja, hogy fut-e a szimuláció (tehát el van-e indítva a timer).

public void setSzuletes(JCheckBox[] e)

Beállítja a szuletes tömböt. Checkboxok tömbjét kapja paraméterként. Végignézem, hogy mely checkboxok voltak pipálva, ahol igen, ott true, egyébként false érték lesz a szuletes tömbben.

public void setTuleles(JCheckBox[] e)

Beállítja a tuleles tömböt. Checkboxok tömbjét kapja paraméterként. Végignézem, hogy mely checkboxok voltak pipálva, ahol igen, ott true, egyébként false érték lesz a tuleles tömbben.

public void setIdo(int i)

Paraméterként kapja a beállítandó időt. Ennyi időnként fogja a timer meghívni a lep action listenert.

protected void paintComponent(Graphics g)

Ezt a metódust nem hívom meg kézzel a programban egyszer sem, hiszen ez meghívódik magától, amikor a mátrix egy része megváltozik. Nem tudom, hogy miért van így, de az internet azt mondta, hogy fogadjam el, hogy ez így működik, és ne foglalkozzak vele.

LEP

public void actionPerformed(ActionEvent e)

Egy ActionEvent-et kell kapnia az actionPerformed metódusnak. A szimuláció élet metódusát hívja.

public void setSzimulacio(Szimulacio sz)

Az ablak konstruktorában létrehozott szimulációt kell kapja paraméterként. A menu hívja, amikor létrehoz egy Lep-et. Beállítja a szimuláció attribútumot.

ITERACIO

public static Sejt[][] ujIteracio(Sejt[][] matrix, boolean szuletes[], boolean tuleles[])

A szimuláció elet metódusa hívja, feladata kiszámolni, hogy a mátrix mely sejtjei fognak élni, illetve halni a következő iterációban a megadott szabályok mellett. Létrejön egy új mátrix, ezt fogjuk feltölteni a sejtek új állapotával. A mátrix minden elemére kiszámolom, hogy hány szomszédja van, majd ha az adott sejt élő, és ha a tulélésnél beállított számok valamelyikével megegyezik a szomszédai száma, akkor a következő iterációban is életben marad. Ha nem egyezik meg egyik beállított szám sem a szomszédok számával (tehát egyszer sem lépett be az if ágba az előző for cikluson belül), akkor halott lesz a sejt a következő iterációban. Ellenben, ha a sejt halott volt, de a születésnél beállított számok valamelyikével megegyezik a szomszédok száma, akkor következő iterációban élővé válik a sejt. Ha nem egyezik meg egyik beállított szám sem a szomszédok számával (tehát egyszer sem lépett be az if ágba az előző for cikluson belül), akkor halott lesz a sejt a következő iterációban.

Paraméterként kapja a jelenlegi mátrixot. Megkapja, hogy hány szomszéd esetén születik egy sejt, illetve, hogy hány szomszéd esetén él túl egy sejt.

Az újonnan elkészített mátrix-szal tér vissza.

public static int szomszedokSzama(int x, int y, Sejt[][] matrix)

A paraméterként kapott sor előtti sortól az utána lévő sorig (ez 3 sor), és a paraméterként kapott oszlop előtti oszloptól az utána következő oszlopig (ez 3 oszlop) megnézzük, hogy az éppen aktuális sejt, ahol a ciklus van elő-e, ha igen, akkor a szomszédokat számláló int-et egyel növelem. Előfordulhat, hogy olyan sejt szomszédait szeretnénk megkapni, ami a mátrix szélén van, ebben az esetben ArrayIndexOutOfBoundsException-t kapunk, de ignorálom és inkább folytatom a következő iterációval. A ciklus végén még meg kell nézni, hogy a paraméterként kapott helyen lévő sejt élő volt, mert az előzőekben azt is beleszámoltuk, akkor a szomszédok közé, és ki kell vonni a számból. Egyébként így már visszatérhetünk a számmal.

Paraméterként kapja, hogy hogy a mátrix mely sorában, illetve mely oszlopában lévő sejt szomszédait kell megszámolni. Ezen felül természetesen a mátrixot is megkapja.

Visszaadja, hogy hány szomszéddal rendelkezik az adott sejt.

FELHASZNÁLÓI KÉZIKÖNYV

A táblánt 400x200 sejt látható. Ebből a lilák, amik élő sejtek, a szürkék pedig a halottak. A program minden újabb futtatásakor random dől el, hogy mely sejtek milyen állapotúak a táblán.

Szabályok

A menü jobb felső részében checkboxokkal lehet állítani, hogy hány szomszéd esetén éljen túl illetve szülessen egy új sejt. Ha egy adott számú chekcbox selected, akkor a sejt adott számú szomszéd esetén túlél/születik, egyébként nem. A checkboxokat nem lehet változtatni, ha a szimuláció el van indítva.

Idő

Két iteráció között eltelt időt lehet ebben a text field-ben állítani. A text field csak számokat fogad el. Ha például 50 van bele írva, akkor indítás után 50 ms-onként fognak frissülni a sejtek. Az időt nem lehet változtatni, ha a szimuláció el van indítva.

Mentés

A mentés gomb a tábla aktuális állapotát tudja elmenteni.

Visszatöltés

Ezzel a legutóbb elmentett állapot töltődik vissza a táblára.

Start/Stop

A Start gomb megnyomásával elindul a szimuláció a beállított szabályokkal, és idővel. Futás közben ez utóbbiakon változtatni nem lehet, azonban menteni és visszatölteni viszont igen.
Amint a szimuláció elindul a Start gomb Stop gombbá változik, és akkor a funkciója a szimuláció megállítása lesz.