



# Tutorial

Az életjáték

Fables implementációja

## Készítették:

Erdélyi Viktor, Iványi Márton Dávid és Legéni

Richárd Olivér

Az ELTE-IKKK részére



**Budapest, 2007. november**

# 1 Tartalomjegyzék

2	Bevezető	3
2.1	Ágens-alapú modellezés	3
2.2	MASS	3
2.3	FABLES ismertető	4
3	Telepítés	7
3.1	Rendszerkövetelmények	7
4	Bevezetés	7
4.1	John H. Conway Életjátéka	8
5	Új projekt létrehozása	9
6	Az IME használata	11
6.1	A modell struktúrája	11
6.2	A Fables kód fordítása	14
7	A Charting Wizard	16
7.1	Egy Grid2D grafikon létrehozása	16
8	A modell futtatása	19
9	Dokumentumok generálása	23
10	Többszöri fordítás és futtatás	25
11	Zárszó	25
12	Melléklet: A teljes forráskód	26

## 2 Bevezető

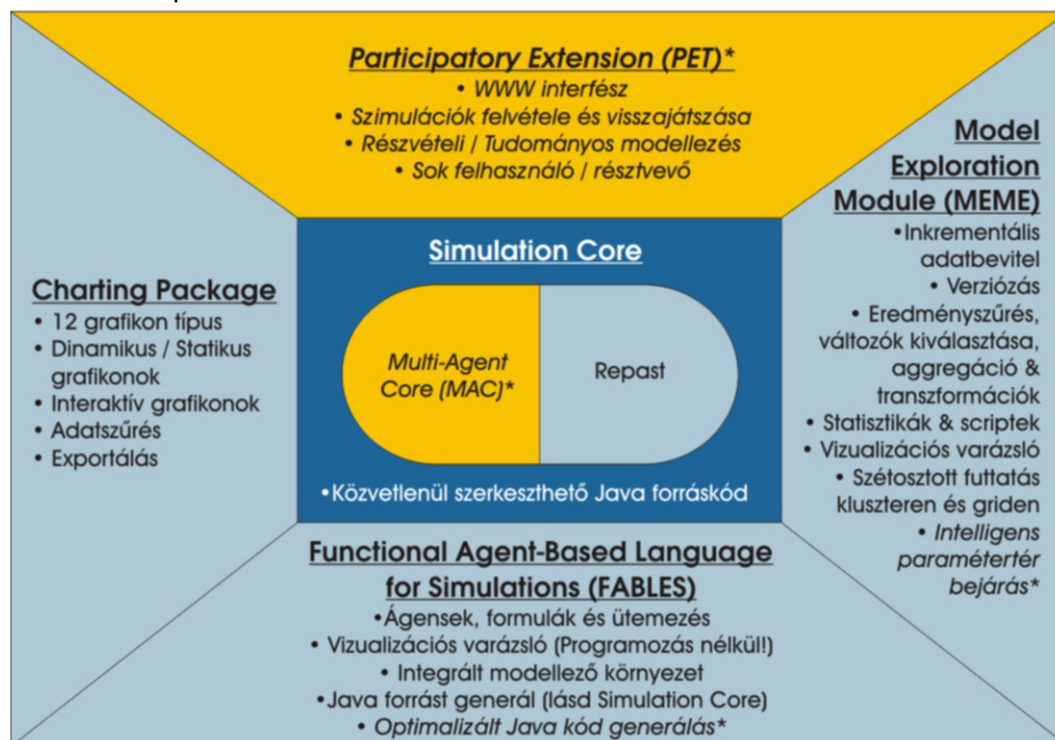
### 2.1 Ágens-alapú modellezés

Az ágens-alapú modellezés a számítógépes szimulációk egy - komplex társadalmi rendszerek modellezésére különösen alkalmas - új ága. Alapvetése az, hogy az egyént modellezzük, tökéletlenségeivel (pl. korlátozott kognitív és számítási képességek), egyéni jellegzetességeivel és egyedi interakcióival együtt. A modell tehát "alulról felfelé" épül - elsősorban a mikro szabályokra koncentrálva, de a makró jelenségek kialakulását kutatva. A részvételi szimuláció, mint az ágens-alapú szimuláció alfaja egy olyan metodológia, amely az emberi szereplők és a mesterséges ágensek együttműködésére alapoz. Ezek a megoldások a képzési és döntés-támogatási területeken igen hasznosak.

### 2.2 MASS

A Multi-Agent Simulation Suite (MASS) egy szoftvercsomag, amely lehetővé teszi a felhasználó számára, hogy az ágens-alapú modellezés megoldásait változatos területeken alkalmazza anélkül, hogy komoly programozási ismereteket kelljen elsajátítania.

A programcsomag négy, egy ún. szimulációs mag köré szerveződő alkalmazásból áll össze. A MASS rendelkezik egy saját maggal (ez a MAC), illetve képes futni Repast alapokon is. A több szimulációs magon való futtathatóság biztosítja, hogy a modellek mag-függetlenek legyenek, így a használható magok számát a jövőben bővíteni kívánjuk. A Functional Agent-Based Language for Simulation (FABLES) egy olyan programozási nyelv és modellezési környezet, amely kifejezetten ágens-alapú modellek fejlesztését szolgálja. A Model Exploration Module (MEME) paraméter terek bejárását, a kinyert adatok feldolgozását és megjelenítését hivatott támogatni. A Participatory Extension (PET) egy opcionális web-alapú környezet multi-ágens és részvételi szimulációk futtatásához. A MASS negyedik része, a Vizualizációs Csomag, nem jelenik meg önálló programként, a többi szoftverben használt grafikonok és vizualizációk implementációit tartalmazza.



1. ábra - Multi-Agent Simulation Suite

## 2.3 FABLES ismertető

A FABLES egy olyan programozási nyelv, amely kifejezetten ágens-alapú modellezéshez készült. A nyelv fejlesztésénél a fő célkitűzés az volt, hogy a lehető legkevesebb programozási ismeret mellett lehessen jól működő szimulációkat készíteni. Ezért a nyelv rendelkezik a szimulációs komponensek deklarálásához szükséges szintaktikai elemekkel, mint például az ágens tulajdonságok, vagy ütemezés.

A nyelv leválasztja egymástól a modellt és annak grafikus megjelenítését. A modellezőnek a modell struktúráját és viselkedését kell leírnia forráskódban, míg annak vizualizációját interaktív varázsló segítségével állíthatja össze, minden programozás nélkül. Több mint 15 fajta grafikon és egyéb vizualizáció használható, elérhetőek beépített operátorok, melyek segítségével a szimulációs eredmények általános statisztikai jeleníthetőek meg, illetve a haladó felhasználóknak lehetőségük van rövid scriptek használatára a vizualizációkban.

A FABLES forrásfájl a modell leírását tartalmazza. A modellek konstansok, változók, függvények és ágensek használatával írhatóak le. Ezek az elemek a modell alapvető részei, leírják a modell állapotát, a paramétereket, valamint az ágensek viselkedését és struktúráját.

```
model SimpleModel {
    taxRate = 0.2;
    var pocketMoney;

    tax = pocketMoney * taxRate;

    class Agent {
        var name;
        var money;
    }
};
```

### 1. Példa –Egyszerű definíciók

A modell viselkedése függvények segítségével adható meg. Mivel a FABLES leginkább egy funkcionális nyelv, az ágensek interakcióit, a modell működését függvények irányítják: a relációkat, a szabályokat és a formulákat mind megadhatjuk velük. A funkcionális megközelítésben a függvények mindig igaz relációkat definiálnak. Segítségükkel mindig azt írjuk le, hogy mit tegyen egy függvény, és nem azt, hogy azt hogyan végezze. A nyelv szintakszisa úgy lett kialakítva, hogy hasonló legyen a publikációkban használt matematikai formalizmushoz (algoritmusok közlésére általában nincs hely, így a legtöbb modellt formulákkal és kvantorokkal írjuk le).

A nyelv fel van készítve kollekciónak hatékony kezelésére is. A kollekciónak (sorozatok, listák) a nyelvi alapelemihez tartoznak, és igen könnyen kezelhetőek.

```
foreach = for each i in [1..5] do println(i) ;
sequence = [ println(i) : i is [1..5] ] ;
odd_squares = { x*x : x is [1..10] when x*x mod 2 == 1 } ;
factorial(n) = (n == 0) => 1
              otherwise => n * factorial(n-1);
```

### 2. Példa – Kollekciónak kezelése

A szimuláció állapot-változásainak leírását az ütemező (*scheduler*) végzi. A FABLES ütemezője diszkrét idő és diszkrét esemény paradigmával dolgozik. Ez azt jelenti, hogy a szimuláció dinamikáját események definiálják, melyek diszkrét időlépésekhez rendelhetőek.

A FABLES hatékony és rugalmas ütemezési lehetőségekkel rendelkezik: a modellekhez korlátlan számú ütemezés adható, melyek dinamikusan megállíthatóak és újraindíthatóak, mi több, független események is bármikor beszúrhatóak és végrehajthatóak.

<pre>// Egyszerű, diszkrét // események által // meghatározott ütemezés schedule {   1 : println("1");       println("Begin");   3 : println("3");   10 : println("10");   12 : println("12");   15 : println("15");   44 : println("44");       println("End"); }</pre>	<table border="1"> <tr><td>1</td><td>Fables statement;</td></tr> <tr><td>3</td><td>Fables statement;</td></tr> <tr><td>10</td><td>Fables statement;</td></tr> <tr><td>12</td><td>Fables statement;</td></tr> <tr><td>15</td><td>Fables statement;</td></tr> <tr><td>44</td><td>Fables statement;</td></tr> </table>	1	Fables statement;	3	Fables statement;	10	Fables statement;	12	Fables statement;	15	Fables statement;	44	Fables statement;
1	Fables statement;												
3	Fables statement;												
10	Fables statement;												
12	Fables statement;												
15	Fables statement;												
44	Fables statement;												
<pre>// Opcionálisan: ciklikus // ütemezés schedule cyclic 1 {   1 : println("1");   3 : println("3");   5 : println("5"); }</pre>	<p>The diagram illustrates how a cyclic event is expanded into global events. On the left, a table labeled 'Global events' shows a sequence of events: 1 (println(1)), 2 (println(1)), 3 (println(1); println(3)), 4 (println(1); println(3)), 5 (println(1); println(3); println(5)), and so on. On the right, a smaller table labeled 'Cyclic 1' shows the repeating sequence: 1 (println(1)), 3 (println(3)), and 5 (println(5)). Three arrows point from the 'Cyclic 1' table to the corresponding rows in the 'Global events' table, showing how the cyclic sequence is interleaved with other events.</p>												

### 3. Példa – Ütemezés definiálása

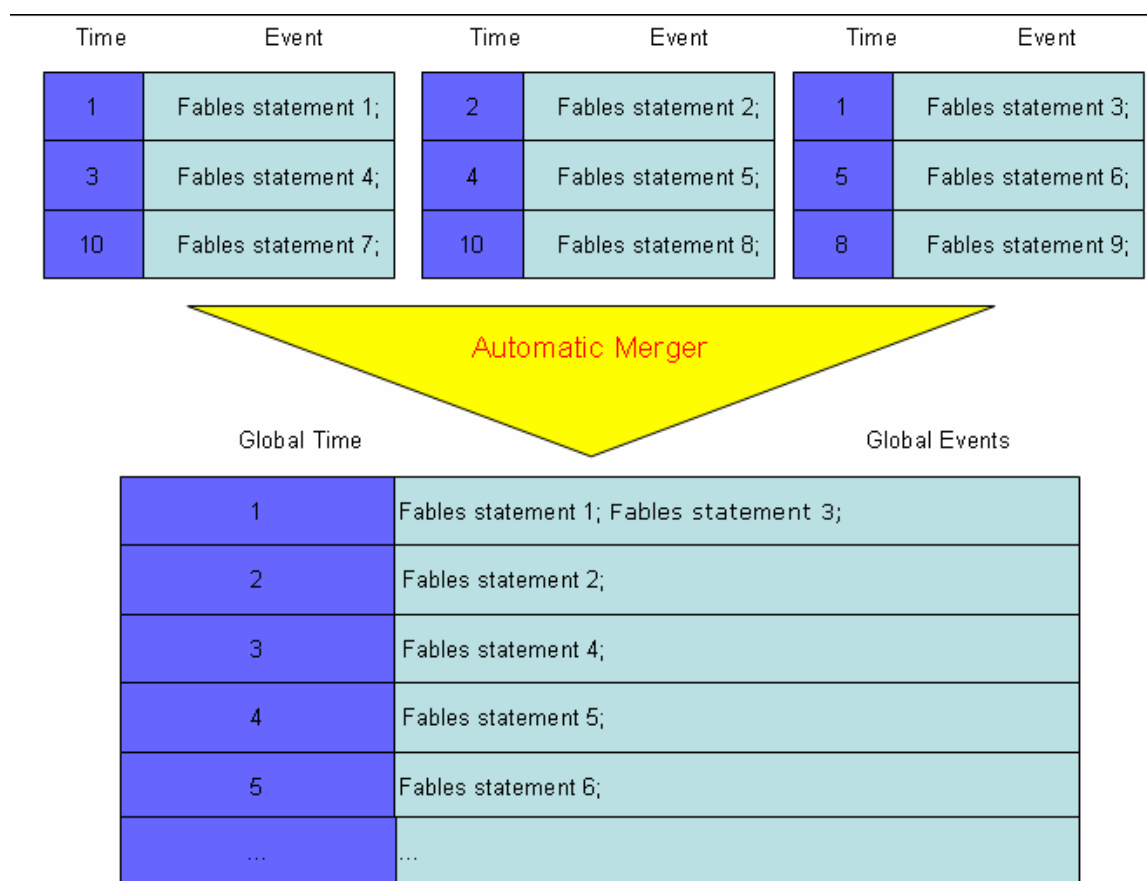
Az ágenseknek saját belső ütemezők lehetnek, melyek az ágens létrejöttkor indulnak el, így idejük is ekkortól számítódik.

```
class Agent {
  schedule {
    5 : println( "Time is now : " ++ time );
  }
}

schedule Main {
  10 : new Agent;
}

Time is now : 15
```

A különböző ütemezők eseményeit a futtatórendszer automatikusan összehangolja.



## 2. ábra – Ütemezők összefűzése

A korábban már említett struktúrák, változók és dinamikus elemek mellett a FABLES rendelkezik egy speciális struktúrával a szimuláció kezdeti feltételeinek megadásához. A `startUp` szekcióban foglaltak egymás után, a szimuláció 0. körében, az ütemező(k) eseményei előtt kerülnek végrehajtásra.

A FABLES modellek Repast J 3.1-re (Java) fordulnak. A fordító Java forráskódot generál, amelyet haladó felhasználók prototípusként is használhatnak a további fejlesztéseikhez.

A FABLES modellezési környezet ugyancsak lehetőséget nyújt arra, hogy haladó felhasználók elérjék Java függvényeiket FABLES programjaikból (Java2FABLES interfész). Így a FABLES modellek Java könyvtárakkal egészíthetők ki, illetve a bonyolultabb metódusok Java-ban is megírhatóak.

A FABLES emellett számos saját beépített függvénnyel is rendelkezik, amiket a Standard Library (StdLib) fog össze. Ezek megkönnyítik és gyorsabbá teszik a modellek elkészítését. Az StdLib függvények például kollektciók kezelésével, statisztikai műveletekkel, vagy pszeudó véletlen számsorozatok generálásával foglalkoznak.

## 3 Telepítés

A Fables IME telepítéséhez Windows környezetben futtassa a Fables\_Installer.exe fájlt. A telepítő szoftver végigvezeti a telepítés lépésein. Egyéb esetben az install.txt utasításai alapján végezze el Fables.zip fájl kitömörítését.

### 3.1 Rendszerkövetelmények

#### 3.1.1 Hardver

Intel x86/x64 és ezzel 100% kompatibilis processzorok teljes mértékben támogatottak. Minimális konfigurációnak egy Pentium III 700 MHz vagy gyorsabb processzor, legalább 256 MB RAM, és 200 MB szabad tárkapacitás szükséges.

#### 3.1.2 Operációs rendszer

Jelen pillanatban a következő operációs rendszerek támogatottak:

- Linux
- Macintosh / MacOS X (bármely, Carbon Application Programming Interface támogatással rendelkező változat)
- Windows 2000 (SP3+)
- Windows XP Home (SP1+)
- Windows XP Professional (SP1+)
- Windows Server 2003 R2
- Windows Vista\*

\* **Megjegyzés:** Habár rendelkezésre áll a Fables IME egy olyan változata is, amely képes teljes mértékben kihasználni a Windows Vista képességeit, ez még kísérleti állapotban van. Ajánlott a szoftver Windows XP operációs rendszerre készített változatát használni Vista alapú rendszereken is.

#### 3.1.3 Szoftver

A Fables IME használatához a Java futtatókörnyezet (Java Runtime Environment) 1.6 vagy újabb verziószámú változata szükséges. A telepítőszoftver ezt a JRE változatot is tartalmazza. Egyelőre kizárólag 32-bites rendszerek támogatottak.

## 4 Bevezetés

Ez a kézikönyv végigvezeti Önt a keretrendszer használatának alapvető lépésein, és egy bepillantást nyújt a Fables programozási nyelv alapjaiba. Ennek a leírásnak a segítségével képes lesz egy egyszerű szimuláció létrehozására, és megismeri a Fables Modellezői Környezet (Integrated Modeling Environment, IME) alapvető funkcióit. John H. Conway Életjátékának egy modelljét fogjuk létrehozni.

A Fables (Functional Agent-Based Language for Simulations) egy egyszerű, könnyen használható programozási nyelv, amely ágens alapú szimulációk létrehozására koncentrál. A programozási nyelv nyelvi szinten támogatja szimulációk létrehozását, ütemezését valamint megfigyelését. Minimális programozói tapasztalattal is jól használható, a formalizmusa igen közel áll a témában publikált forrásokban felhasznált formalizmusokhoz.

A nyelv egy integrált fejlesztőkörnyezettel is rendelkezik, amely az Eclipse platformra épül. A futtatásához a telepített programot kell mindössze elindítani, másra nincs szükség. Az IME számtalan előnnyel rendelkezik, amely megkönnyíti Fables programok készítését, mint például a kódszínezés és kiegészítés, beépített sűgők és automatikus dokumentumgenerátorok.

Ha már rendelkezik egy kevés programozói tapasztalattal, észreveheti, hogy a Fables egy multiparadigmás programozási nyelv: vegyíti az objektum elvű, funkcionális és procedúrális nyelvek tulajdonságait.

## 4.1 John H. Conway Életjátéka

Az Életjáték a sejt-automaták egy leginkább ismert változata, amelyet egy angol matematikus, John Horton Conway talált ki az 1970-es években. A modell egy egyszemélyes játék, amelynek fejlődése egyedül a kezdeti állapottól függ. Mivel nincs szükség a szimuláció során emberi beavatkozásra, az ágens alapú megközelítés tökéletes architektúra hozzá.

### 4.1.1 Szabályok

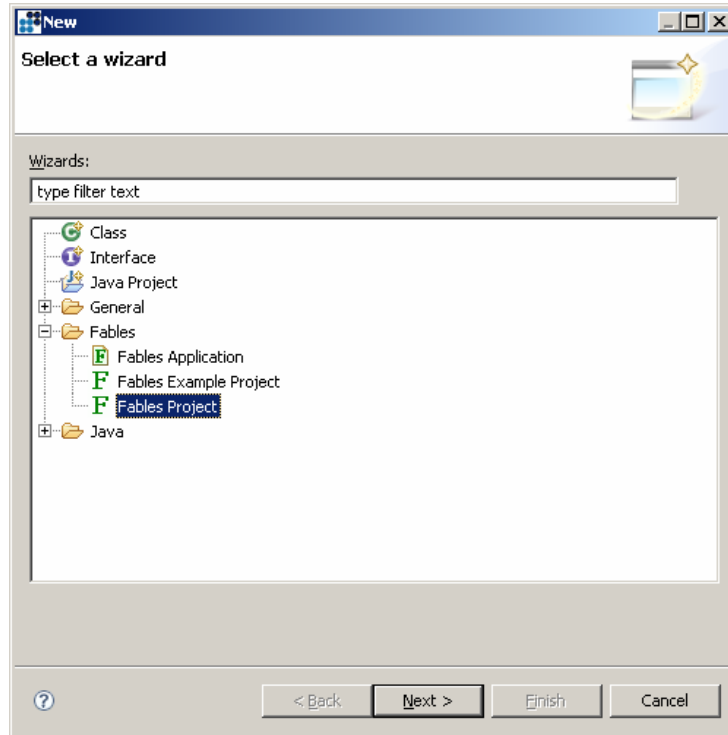
Az Életjátékban a világ egy véges, kétdimenziós négyzetrács, ahol minden egyes cellának (sejtnek) két különböző állapota lehet. A sejtek élők vagy halottak lehetnek, és környezetükben lévő cellákkal állnak kölcsönhatásban, ahol a környezet alatt a vele vízszintesen, függőlegesen és átlósan szomszédos nyolc cellát értjük. Egy sejtrel egy szimulációs körben a következő négy dolog történhet:

1. Egy élő sejt elpusztul, ha kettőnél kevesebb szomszédja van (elszigetelődés).
2. Egy élő sejt elpusztul, ha háromnál több szomszédja van (túlnépesedés).
3. Egy sejt túléli a kört, ha pontosan két vagy három szomszédja van.
4. Egy halott cella élővé változik, ha környezetében pontosan három sejt található.

A kezdeti állapot határozza meg a rendszer első generációját. A második generációt úgy kapjuk, ha a fenti szabályokat végrehajtjuk az első generáció teljes populációjára, így az egyes cellák hol életre kapnak, hol elpusztulnak. A további generációk előállításához a fenti szabályokat alkalmazzuk egymás után.

## 5 Új projekt létrehozása

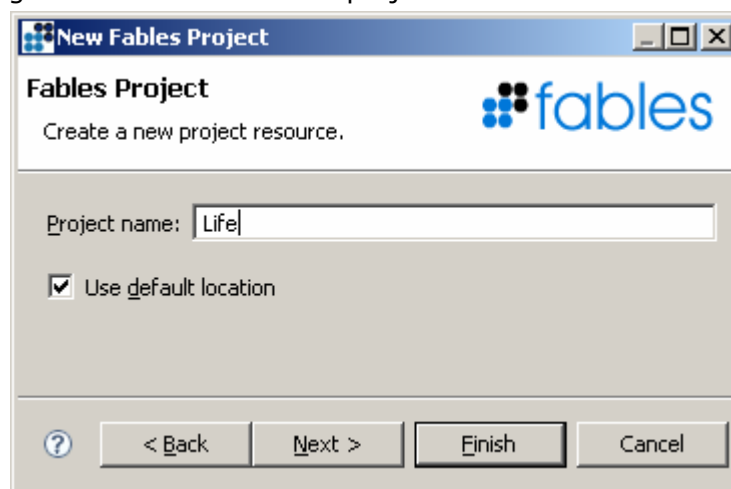
Új Fables Projekt létrehozásához nyomja meg a CTRL + N billentyűkombinációt, vagy kattintson a File -> New -> Project menüre. A következő képernyőt fogja látni, amely azt jelzi, hogy egy új projektet készül létrehozni:



3. ábra

A lenyíló menüből válassza a Fables Project elemet. Egy Fables projekt ugyanolyan, mint egy egyszerű Java projekt, de azon kívül, hogy Java forrásfájlok kezelésére is képes (így lehetővé téve Repast szimulációk futtatását), Fables állományokat is használhatunk benne.

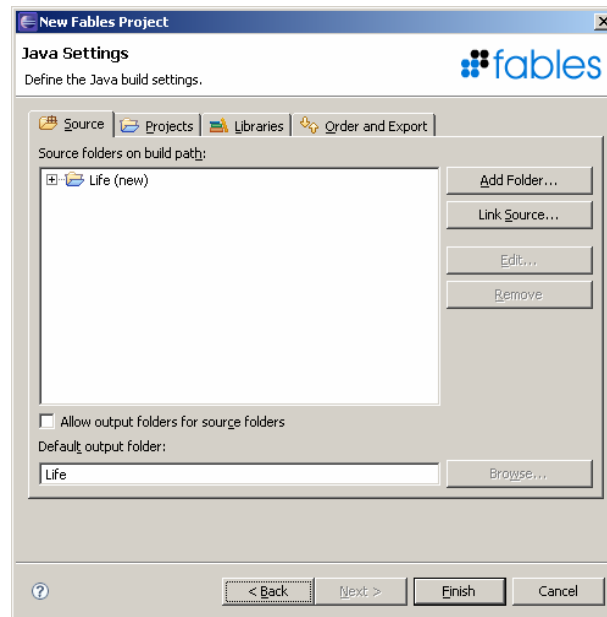
Miután kiválasztotta a Fables Project opciót, nyomja meg a Next gombot, és a következő dialógusablakon nevezze el a projektet:



4. ábra

A "Project name" mezőbe (lásd 4. ábra) írja be a "Life" nevet. A projekt neve egyedül arra jó, hogy a többi projekttől meg lehessen különböztetni, semmilyen hatással nincs magára a modellre.

Az új projekt létrehozásához nyomja meg az ENTER billentyűt vagy kattintson a Finish gombra, és máris elkezdheti a modell szerkesztését. Ha egyéb, Java specifikus beállításokat is szeretne a projekten végezni (mint például egyéb programozói könyvtárak hozzáadása, CLASSPATH beállítások, stb.), nyomja meg a Next gombot, és ezt megteheti a 5. ábraán feltüntetett dialógusablakon.



5. ábra

## 6 Az IME használata

Gratulálunk, ezzel létrehozta az első Fables modelljét. Amint észreveheti képernyő bal oldalán a Navigator ablakban, az új modell hozzáadódott az IME munkaasztalához (workspace), és már tartalmaz is néhány könyvtárat, mégpedig a következőket:

- **Charts:** ez a könyvtár fogja tartalmazni a projekthez tartozó modellek grafikonleíró állományait
- **Documents:** a projekthez tartozó modellek alapján generált dokumentumokat ide helyezi a fordítóprogram
- **Repast:** a projekthez tartozó modellek alapján generált futtatható Repast/Java kódot ide helyezi a fordítóprogram.

Található itt még egy elem, mégpedig a `Life.fab` fájl. A `.fab` kiterjesztésű fájlok Fables forrásfájlok, amelyekben modelleket hozhat létre. Jelen pillanatban a tartalma majdnem üres, egy minimális modelldefiniációt tartalmaz csak, amely nem csinál semmit:

```
model Life {  
  
    startUp {  
        ;  
    }  
}
```

A képernyő alsó részén a Fables Console ablakot láthatja, ahol a fordítóprogram értesíti Önt a fordítások eredményeiről (ide kerülnek például a figyelmeztetések és hibaüzenetek).

A képernyő jobb részén az Outline ablakot láthatja, amely a modell fastruktúráját tartalmazza, a könnyebb navigálhatóság kedvéért.

Sokszor hasznos lehet, ha a szerkesztőablakot maximalizáljuk az IME ablakán belül. Ehhez kattintson a `Life.fab` állományt tartalmazó editor fölére kétszer, vagy nyomja meg a CTRL + M billentyűkombinációt.

### 6.1 A modell struktúrája

Elsőnek definiálunk két konstans értéket a modellhez, amelyek a világban lévő egyes cellák lehetséges állapotait fogják jelölni:

```
dead = 0 ;  
live = 1 ;
```

A modellben szereplő világ méretének megadásához definiáljuk a `worldSize` értéket a következők szerint:

```
worldSize = 50 ;
```

A következőkben egy olyan függvényt definiálunk, amellyel a diszkrét világunkat, amelyet egy kétdimenziós intervallum fog reprezentálni, átjárhatóvá tesszük. Ez egy normalizáló függvény lesz (`norm`).

Igen hasznos a forráskódhoz megjegyzéseket fűzni. Ezáltal a forráskód sokkal érthetőbb, átláthatóbb lesz mind a fejlesztő, mind bárki más számára, aki azt olvassa. Egyes speciális megjegyzéseket a dokumentum generáló eszközök is felhasználhatnak, ezeket be is képesek illeszteni a generált dokumentumba. Erről részletesen még a Dokumentumok generálása fejezetben szólnunk majd. Ilyen típusú megjegyzéseket a forráskódban a `/**` és `*/` jelek között lehet elhelyezni.

A normalizáláshoz használt függvény a hozzá tartozó megjegyzés definíciójával a következőképpen néz ki:

```
/** To normalize x depending on worldSize. */
```

```
norm (x) = x mod worldSize ;
```

A `mod` operátor, amelyet a fenti definícióban használtunk, egy beépített Fables operátor a matematikai maradék számolására. Az `x` értéket adja vissza, ha az értéke a `[0..worldSize]` intervallumon belül van (azaz például `norm(7)` eredménye 7 lesz). Ellenben minden, ezen az intervallumon kívül eső értéket ebbe az intervallumba képez le. Például, a `norm(55)` kifejezés 5 értéket ad, a `norm(-1)` kifejezés 49-et, stb. A Fables Kézikönyvben részletesen megtalálja az operátor működésének leírását.

A modellben szereplő világot egy mátrix típusú változóval fogjuk reprezentálni, ennek definícióját lásd alább. Azért szükséges változóként definiálnunk ezt a tagot, mert ez egy folyamatosan változó tagja lesz a modellnek, a szimuláció minden egyes körében dinamikusán változik, és így áll elő a világ új állapota.

```
var world;
```

Azzal, hogy definiáltuk a világot reprezentáló változónkat (`world`), a világ méretét (`worldSize`), valamint a cellák állapotait jelölő konstansokat (`dead`, `live`), valamint a normalizálást végző (`norm`) függvényt, megalkottuk a modellünk vázát.

Ahhoz, hogy inicializáljuk a modellt, implementálnunk kell a `startUp()` blokkot, ahol megadjuk a `world` változó kezdeti értékét, és beállítjuk a szimuláció futását befolyásoló véletlen szám generátor seed értékét nullára:

```
startUp(worldSize) {
  seed(0);
  world := [ [ discreteUniform( dead, live ) ]
            : x is [0..worldSize-1]
            : y is [0..worldSize-1] ] ;
}
```

A fenti kód alapján a `worldSize` konstans a szimuláció egy paramétere lesz (mivel a `startUp` kulcsszó utáni paraméter-felsorolási listán szerepel). Ez azt jelenti, hogy a szimulációs futások alatt az értéke bármikor megváltoztatható. A `worldSize` értékét arra használjuk, hogy beállítsuk a világ mátrixának méretét (`world`), amely alapértelmezés szerint egy 50x50 méretű mátrix lesz.

Az Életjáték szabályai szerint minden cellához meg kell tudnunk mondani pontosan hány élő szomszédja is van, hogy a következő generációbeli állapotát meg tudjuk határozni. Ehhez meg kell vizsgálnunk az összes (vízszintesen, függőlegesen valamint átlósan) szomszédos sejtet is. Egy adott  $(x, y)$  koordinátájú cella élő szomszédai számának meghatározására definiáljuk a következő függvényt:

```
numberOfNeighbours(x, y) = size [ 1 :
  dx is [-1..1], dy is [-1..1]
  when not (dx == 0 == dy)
  and ( world(norm(x+dx)) (norm(y+dy)) == live )
] ;
```

Elsőre talán bonyolultnak tűnhet a definíció, de ez a példa tökéletesen bemutatja, hogyan kell felsorolásokat iterációkkal, feltételekkel használni.

Alapvetően a fenti kód egy felsorolás méretét adja vissza. Ez a felsorolás pontosan annyi egyes értéket fog tartalmazni, ahány élő szomszédja van az  $(x, y)$  cellának. Ezt a következő mátrix alapján érthetjük meg legegyszerűbben:

$(x-1, y-1)$	$(x-1, y)$	$(x-1, y+1)$
$(x, y-1)$	$(x, y)$	$(x, y+1)$
$(x+1, y-1)$	$(x+1, y)$	$(x+1, y)$

Az  $x$  érték szerinti bejárást a `dx` értékkel, az  $y$  szerinti bejárást a `dy` értékekkel végezzük. A `not (dx == 0 == dy)` feltétel pedig csak arra szolgál, hogy a sejt önmagát ne tekintse szomszédnak.

Már csak egy függvényt kell definiálnunk, az evolúciós szabályt, amely a szokásos "23/3" szabály lesz, ahogy azt a 4.1.1 fejezetben már felvázoltuk:

```
step(x,y) = (n==3 or ( world(x)(y)==live and n==2) )
            => live
            otherwise
            => dead
            where (
                n = numberOfNeighbours(x,y)
            );
```

A fenti kódrészlet egy parciális függvény definíciója. Paraméterként megkapja egy cella **x** és **y** koordinátáját, és egy értéket ad vissza, amely szerint az adott cella a következő generációban élő (**live**, amelyet 1 értékkel reprezentáltunk) vagy halott (**dead**, amelyet a 0 értékkel reprezentáltunk) lesz. Egy helyi változót is bevezet a függvény a **where** kulcsszó után. Ez egy új lokális konstanst készít a függvényhez (**n**), amelyet ebben, és csakis ebben a függvénydefinícióban használhatunk, és az aktuális cella szomszédainak számát adja meg.

Szükség van még egy mátrix definiálására, amely az új generáció értékeit fogja tartalmazni, hiszen mindkét mátrixra szükségünk van a szomszédsági relációk vizsgálatához:

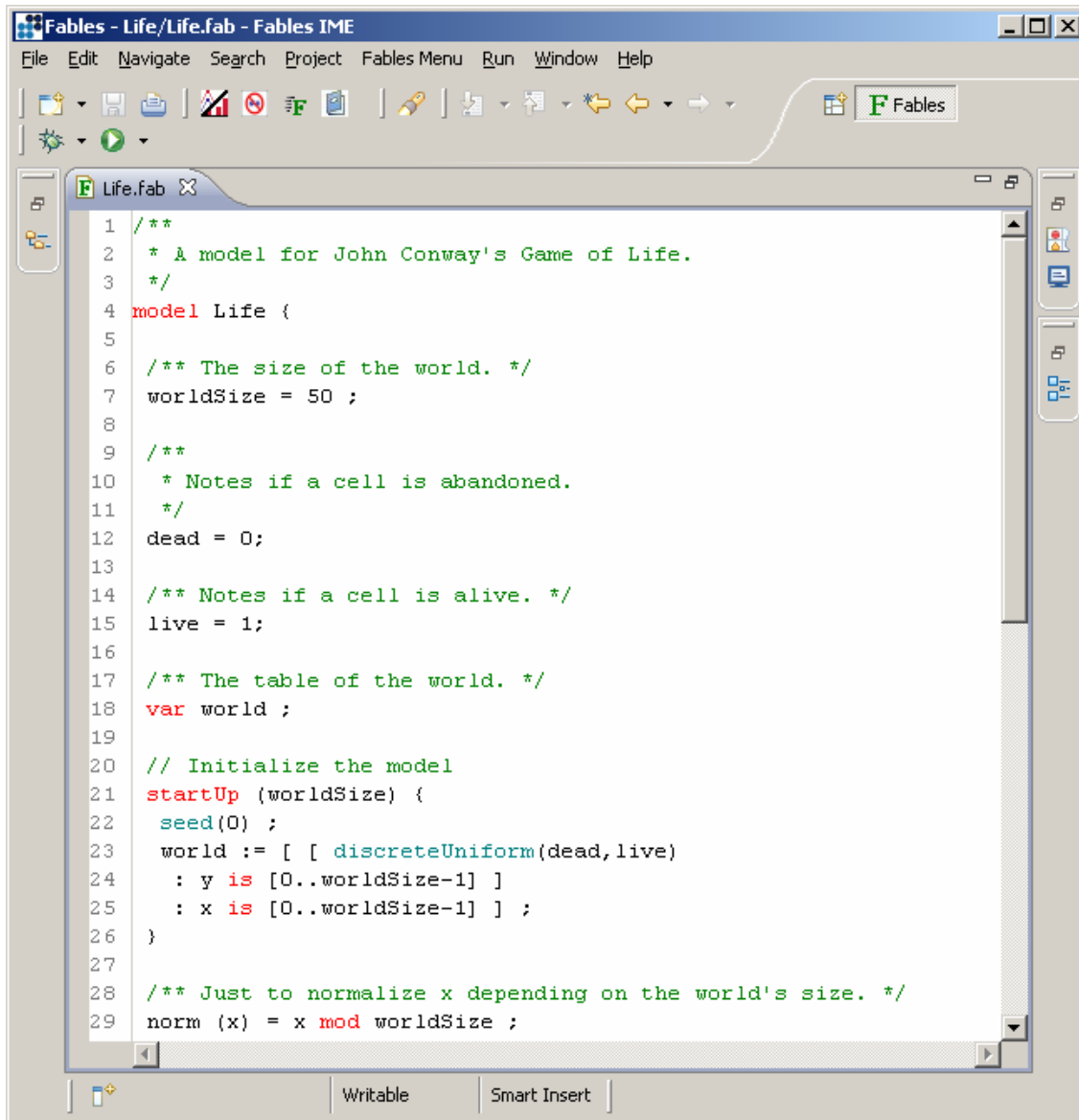
```
newWorld = [ [ step(x, y)
               : y is [0..worldSize-1] ]
             : x is [0..worldSize-1] ] ;
```

Az új világot a fent definiált **step** függvény segítségével építjük fel. Még egy utolsó definíciót kell hozzáadnunk a modellünkhöz, magát az ütemezést, amely a generációk előállítását minden egyes szimulációs körben elvégzi:

```
schedule Main cyclic 1 {
  1 : world := newWorld ;
}
```

A fenti ütemezőt **Main** névvel láttuk el. A feladata az, hogy az inicializálás után a szimuláció minden egyes körében elvégezze a **world := newWorld** értékadás műveletét (**cyclic 1**).

Ha mindent a leírásnak megfelelően tett, az IME a lenti képen látható módon fest. Ha bármilyen problémája volt a modell implementálásával, a 12. fejezetben megtalálja a teljes forráskódot.



```

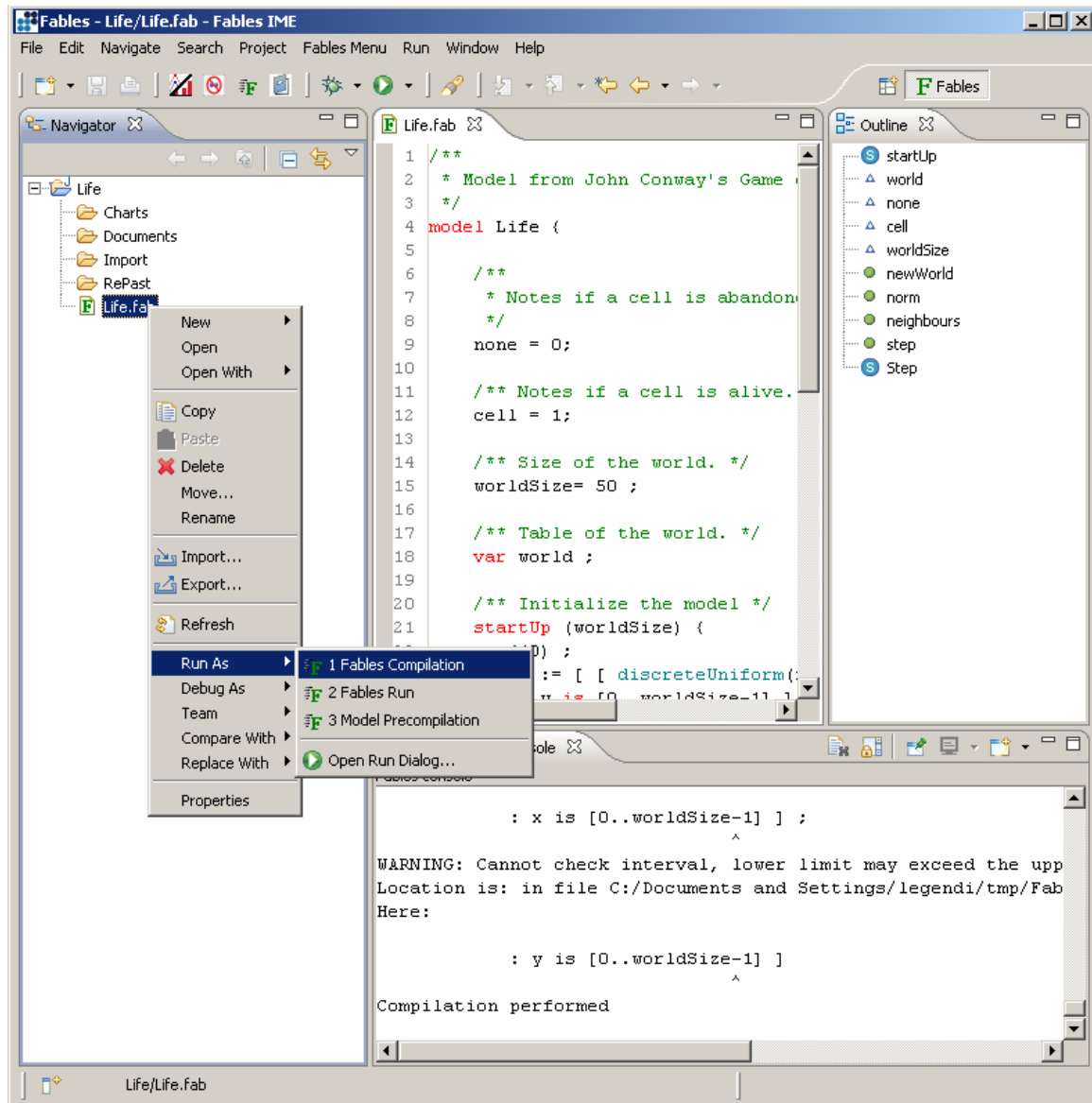
1  /**
2  * A model for John Conway's Game of Life.
3  */
4  model Life {
5
6  /** The size of the world. */
7  worldSize = 50 ;
8
9  /**
10 * Notes if a cell is abandoned.
11 */
12 dead = 0;
13
14 /** Notes if a cell is alive. */
15 live = 1;
16
17 /** The table of the world. */
18 var world ;
19
20 // Initialize the model
21 startup (worldSize) {
22   seed(0) ;
23   world := [ [ discreteUniform(dead, live)
24             : y is [0..worldSize-1] ]
25             : x is [0..worldSize-1] ] ;
26 }
27
28 /** Just to normalize x depending on the world's size. */
29 norm (x) = x mod worldSize ;

```

6. ábra

## 6.2 A Fables kód fordítása

A modell elkészült, most már lefordítható futtatható Reapast kódra. A Fables használatával a modell leírása lényegesen kevesebb erőfeszítésbe került, mintha tisztán Reapast segítségével készítettük volna. A Fables modellből történő Reapast modell generálás teljes mértékben automatikus, ez többféleképpen is elvégezhető. Kattintson az egér jobbgombjával a [Life.fab](#) állományra, és válassza a Run As → Fables application elemet a felbukkanó menüben (lásd 7. ábra).



7. ábra

A fordítás eredményeiről a Fables Console ablakban láthatunk információkat a képernyő alsó részében. A fordítás eredményeképp előálló Repast források az aktuális projekt RePast könyvtárába kerülnek. A munkaasztal minden fordítás elvégzése után automatikusan frissül, ekkor már láthatók is lesznek a legenerált Java állományok.

Amennyiben még nem készített a modellhez vizualizációt (ha mindent ennek a leírásnak a segítségével készített, akkor ebben a helyzetben van), a keretrendszer segítségével azt most megteheti. Erről szól a kézikönyv következő fejezete (7 A Charting Wizard).

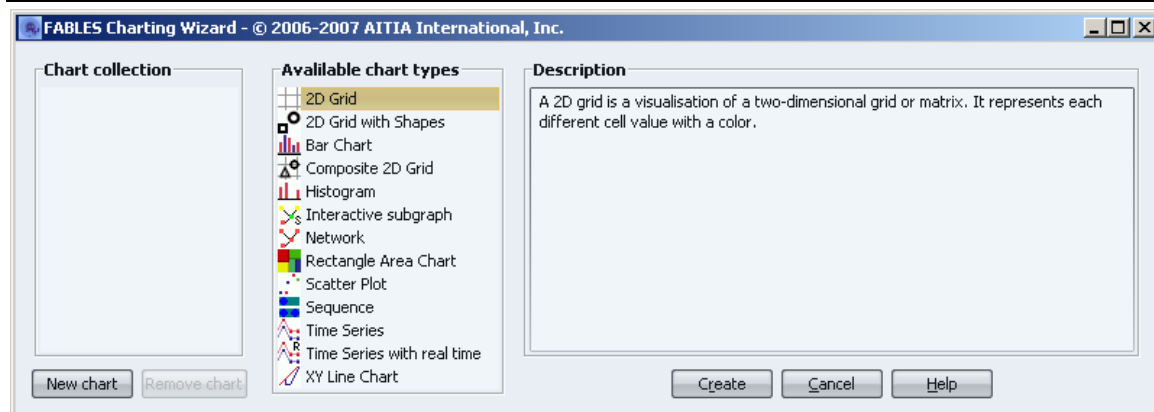
**Megjegyzés:** A fordítóprogram néhány figyelmeztető üzenetet is ír a képernyőre. Ez azért van, mert intervallumok használata esetén a fordítóprogram megpróbálja ellenőrizni, hogy az alsó és felső határok valóban helyesek-e, ugyanis ha az alsó határ meghaladja a felsőt, az futási idejű hibákhoz vezet (ilyen például a [3..1] intervallum). Ezt a vizsgálatot sok esetben képes elvégezni, azonban a [0..worldSize-1] felsorolás egy paramétert tartalmaz, amelynek értéke változhat a futtatás során. Ezért keletkezik tehát fordítási idejű figyelmeztetés is.

## 7 A Charting Wizard

A Fables minden információt kigyűjt a modellből, és meghatározza, milyen grafikonok készíthetők a modellben definiált változók, konstansok és függvények segítségével. Ha kiválaszt egy grafikontípust (lásd 7.1 bekezdés, 8. ábra), akkor csak az annak megfelelő modell tagokat fogja látni (lásd 9. ábra). Ha egy olyan grafikon típust választ ki, amelyhez nincs megfelelő adatforrás a modellben, az IME nem kínál választási lehetőséget.

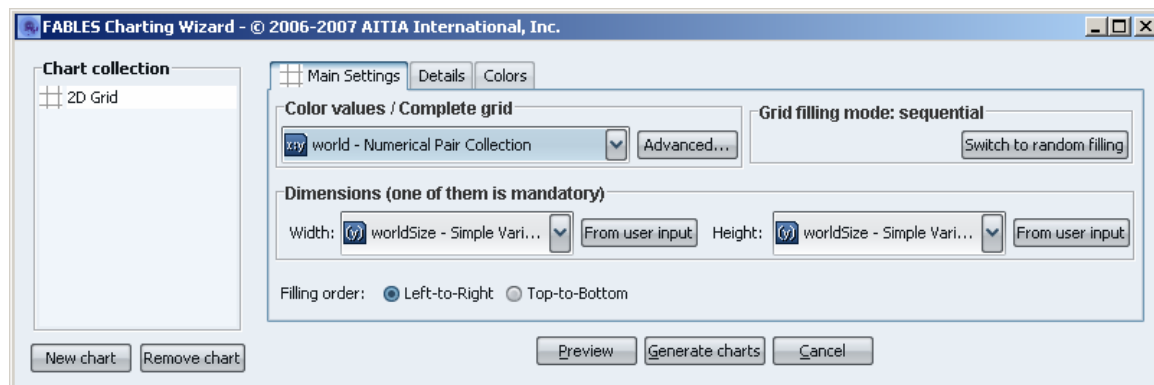
Jelen esetben egy kétdimenziós mátrix reprezentálja a világot, amely tökéletesen megfelel egy 2D Grid (négyzetrács alapú) grafikonnak.

### 7.1 Egy Grid2D grafikon létrehozása



8. ábra

Egy 2D Grid típusú grafikon létrehozásához kattintson rá kétszer az elérhető grafikontípusok listáján, vagy válassza ki, és kattintson a Create gombra (8. ábra). Ezután a varázsló továbbvezeti a következő képernyőre, amelyet a lenti képen (9. ábra) láthat. Ezen az ablakon az aktuális grafikon beállításait adhatja meg.



9. ábra

A *Main Settings* fülön a grafikont adatokkal feltöltő modell tagokat (ún. adatforrásokat) választhatja ki. Válasszuk a `world` változót, valamint állítsuk be a szélesség és magasság értékeket. Habár beállítható konstans, 50 értékre is, azonban ez mégsem javallott – hiszen a modell futása során maga a `worldSize` paramétert bármikor megváltoztathatjuk, de a vizualizáció nem fogja követni ezt a változást, és minden módosításánál szükség lesz a grafikon újbóli szerkesztésére. Ezen esetekre a Charting Wizard képes ezeket az értékeket szintén változókból meríteni. A `worldSize` egyszerű változó értékét választva mind a szélesség, mind a magasság értékének tökéletes megoldás erre a problémára.

A *Details* fülön a grafikonhoz tartozó számos feliratok, címkék szerkesztése végezhető el, ahogy azt a lenti 10. ábra mutatja.



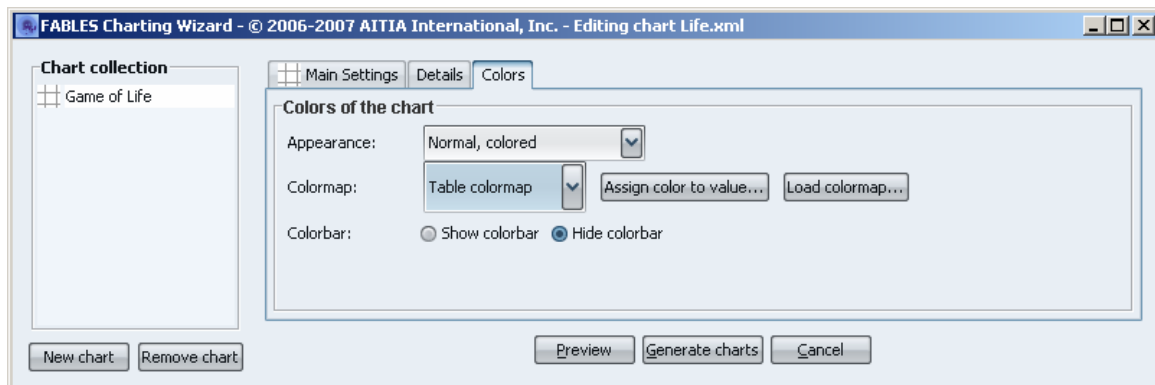
10. ábra

Állítsa be a következő értékeket:

- **Title:** Game of Life
- **Subtitle:** An example application for John H. Conway's Game of Life.

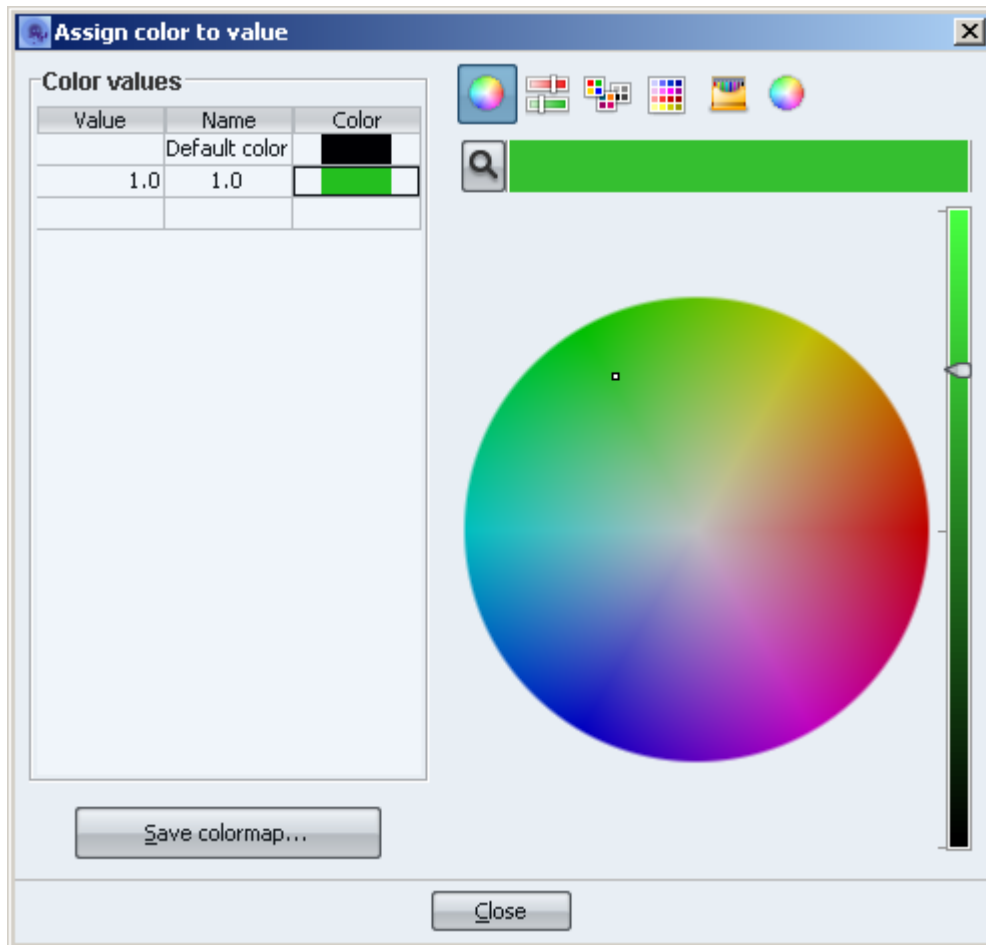
Ezek a feliratok a grafikonon magyarázó információként jelennek meg.

**A harmadik fül a Colors fül** (11. ábra), ami igen fontos beállításokat tartalmaz a 2D Grid grafikonok esetében, itt lehet ugyanis megadni a reprezentált értékek milyen színnel jelenjenek meg a grafikonon. Számos lehetőség áll rendelkezésre, ebből most a Table colormap opciót fogjuk használni. Ezzel a megadással érték-szín párok adhatóak meg a táblázat egyes értékeihez. Ha már definiált egy színtáblázatot, azt később a Load colormap... gombbal ismét felhasználhatja, de jelen esetben létre kell hozni egy új táblázatot az Assign color to value... gomb segítségével (lásd 12. ábra).



11. ábra

A `world` változó két értéket tartalmaz: a 0 érték jelöli a halott, és 1 az élő sejteket. Az alapértelmezett értéket állítsa feketére (kattintson a *Define default color...* feliratra, majd válassza ki a megfelelő színt), és adja hozzá a táblázathoz az 1-es értéket, majd válasszon egy tetszőleges színértéket hozzá (írja be a táblázatba az 1-es értéket, ahogy azt az alábbi 12. ábra mutatja). A Close gomb megnyomásával visszatérhet a varázsló főképernyőjére.

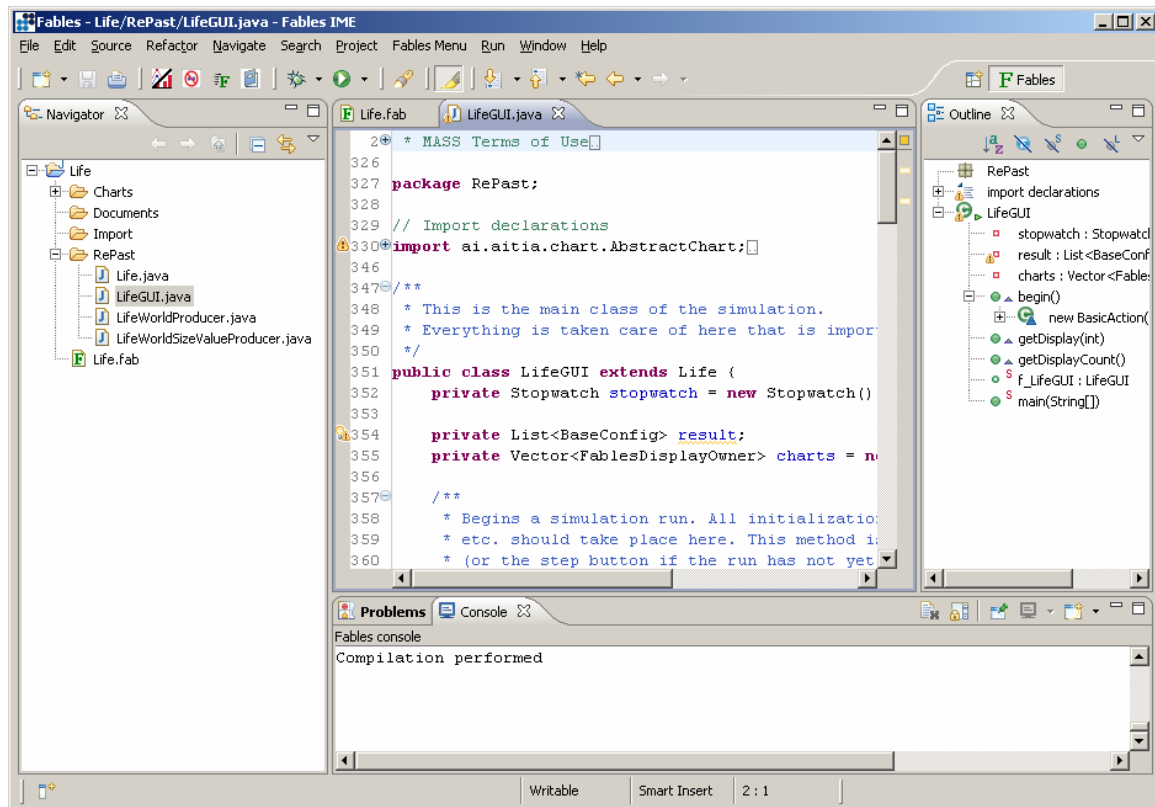
**12. ábra**

A főképernyőre való visszatérés után leellenőrizheti a beállításait a Preview gomb megnyomásával. Ha elégedett a beállításokkal, nyomja meg a Generate Charts gombot, és az új grafikon állomány létrejön a projekt Charts könyvtárában.

Ha az IME a fordítás során új grafikont készít, vagy egy már létezőt szerkeszt, akkor a fordító megvárja, míg a vizualizáció elkészül. Ebben az esetben rögtön folytatódik a fordítás, amint a grafikon elkészült.

## 8 A modell futtatása

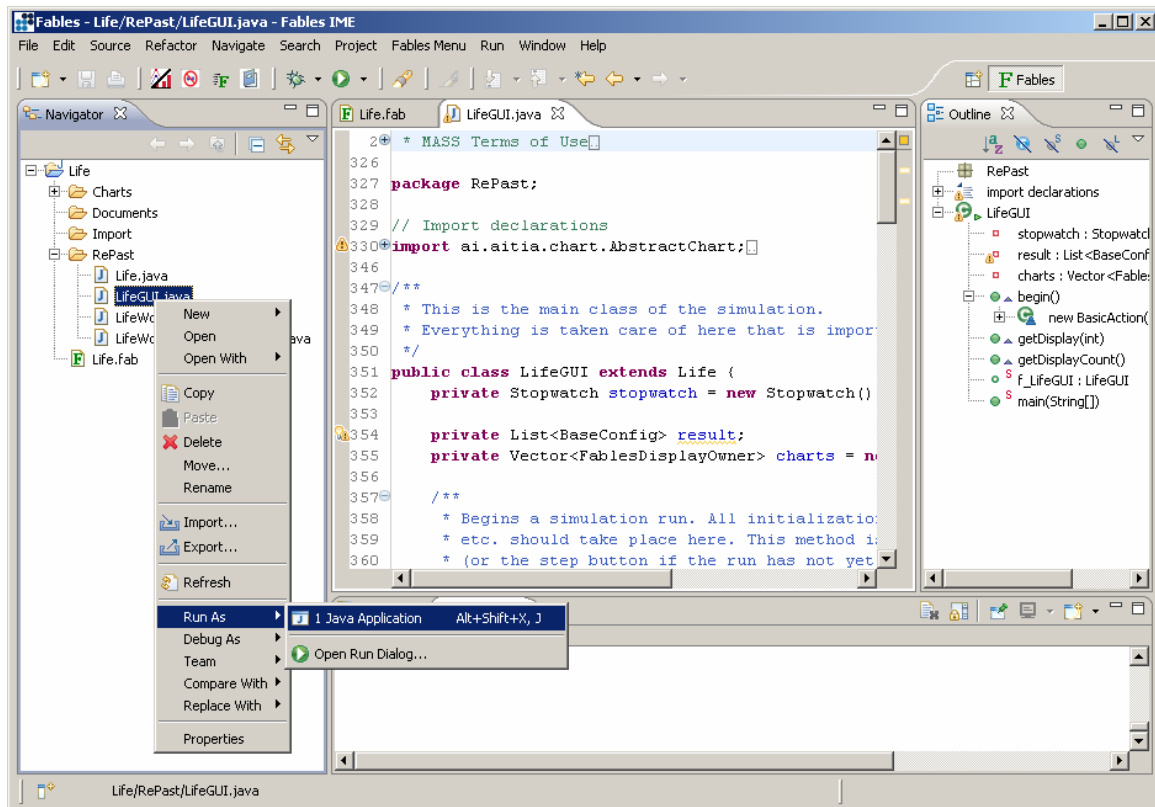
Ahhoz, hogy futtassa a lefordított Fables alkalmazásokat, valamely generált Java forráskódot kell futtatnia. Egyszerűbb modelleknél néhány generált forráskód van csak, jelen esetben négy darab. Az első maga a Repast modell ([Life.java](#), 13. ábra), a második a grafikus felülettel ellátott változat ([LifeGUI.java](#)), a másik két forrásfájl pedig az adatforrásokot reprezentáló forráskódok, amelyek a vizualizációhoz kellenek.



13. ábra

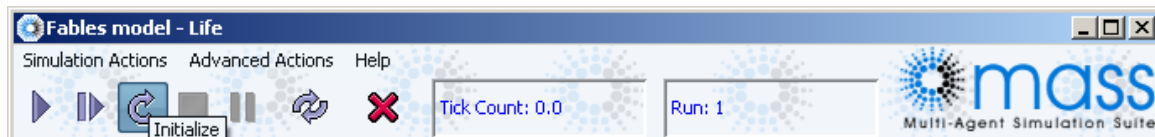
Amennyiben a sima modell futtatja, a futás során nem jelennek meg grafikonok, és csak a modell futása során végrehajtott `print` és `println` utasítások eredménye kerül a konzolra.

A grafikus modell futtatásához az egér jobbgombjával kattintson a [LifeGUI.java](#) fájlra, és válassza a `Run As` → `Java Application` elemet, amint azt a 14. ábra mutatja.



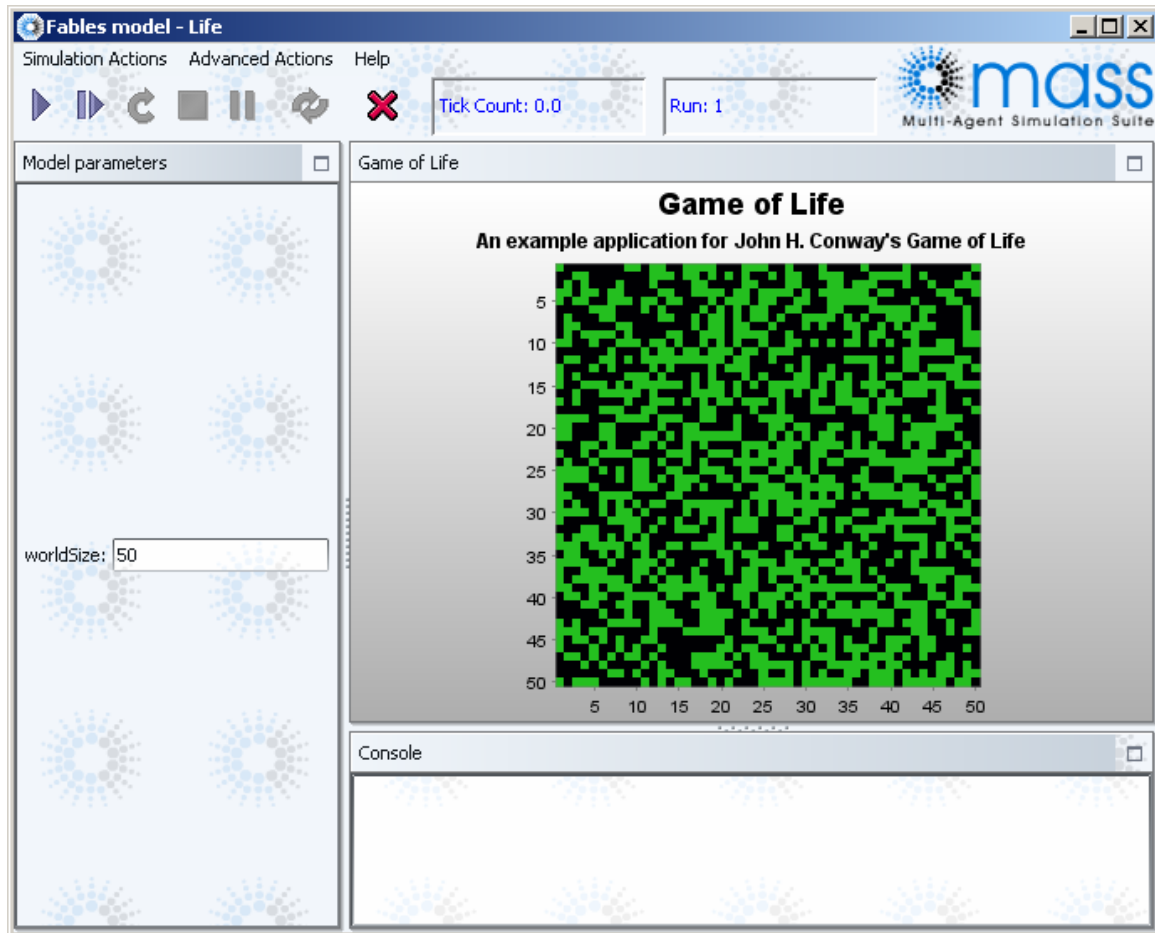
14. ábra

A fenti műveletet végrehajtva megjelenik a Repast konzol a képernyőn. Itt az Initialize gombbal elvégezheti az inicializálás folyamatát (lásd 15. ábra), ezután a modell készen áll a futtatásra.

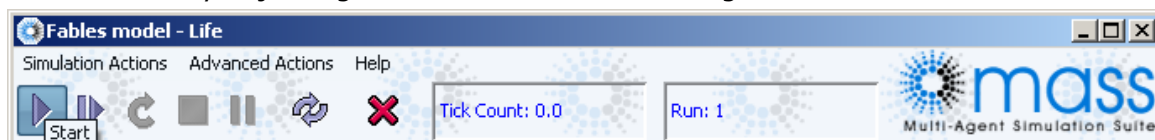


15. ábra

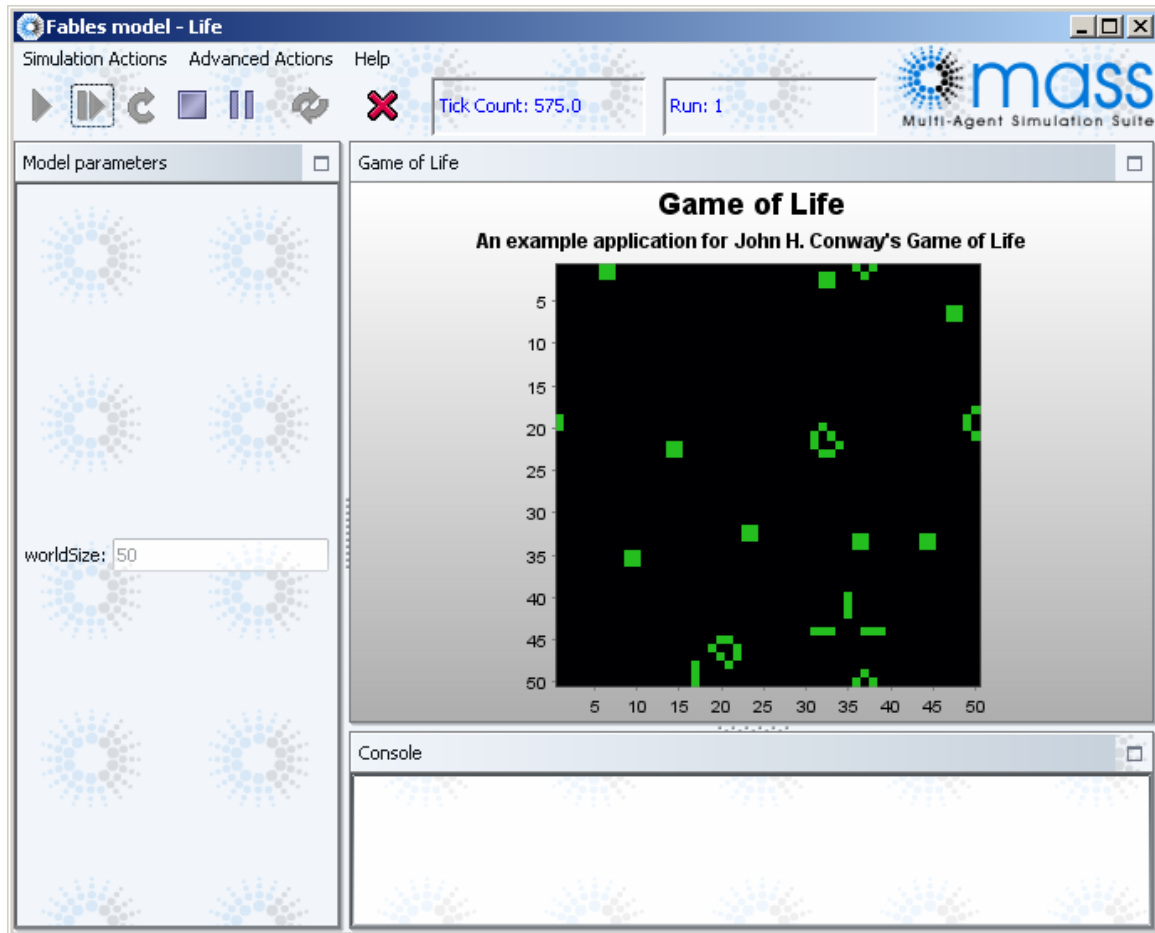
Ha a fenti lépéseket rendben végrehajtotta, a 16. ábraán szereplő ablakhoz hasonló grafikont láthat.

**16. ábra**

Ez a modell egy tipikus véletlenszerű kezdőállapota. A grafikon tartalmazza még azokat a feliratokat, amelyet a Charting Wizard beállításainál adott meg. A modell futtatásához nyomja meg a 17. ábraán látható Start gombot!

**17. ábra**

Jónéhány szimulációs kör múlva a modell nagy valószínűséggel elér egy nagyjából stabil állapotot, ahogy azt a lenti 18. ábra mutatja.

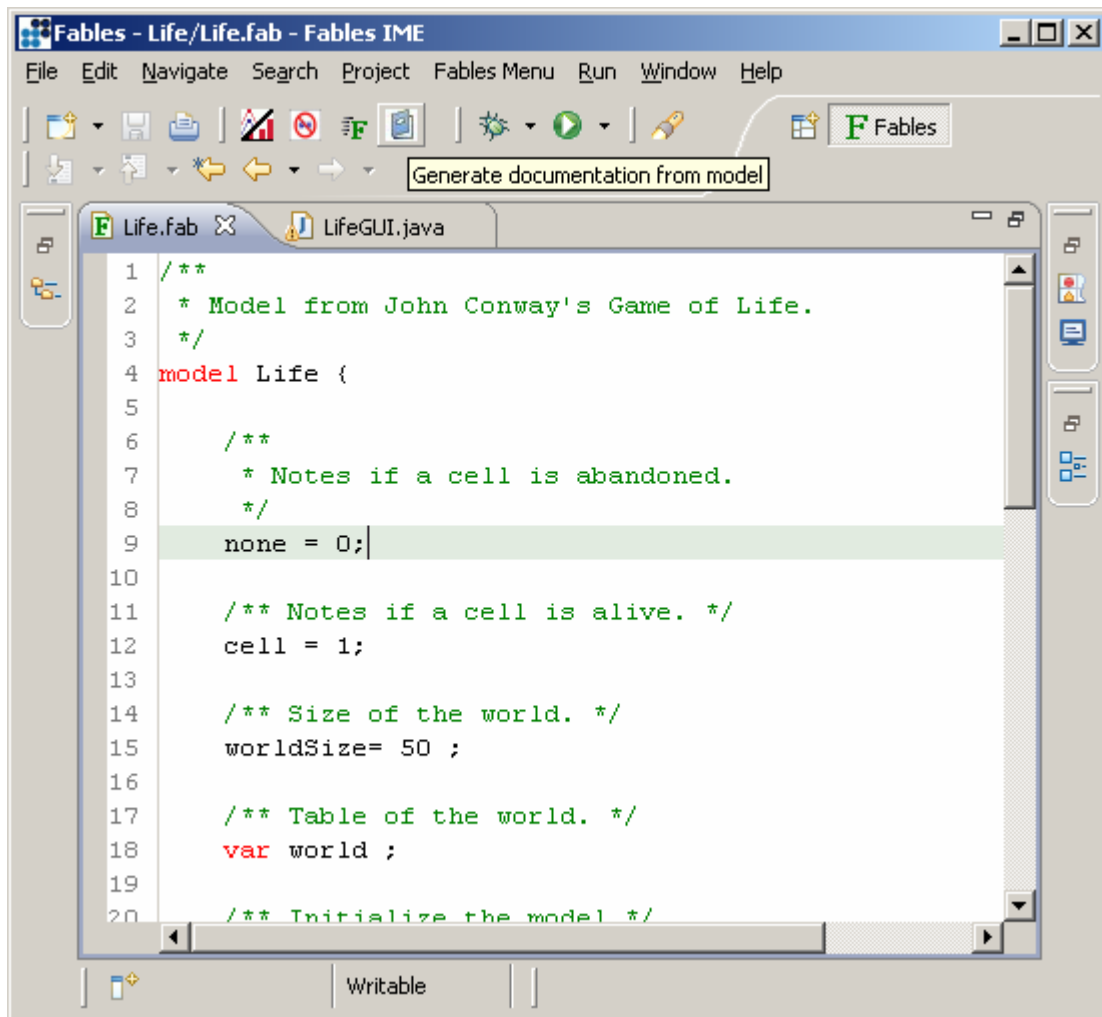
**18. ábra**

A legtöbb sejtcsoport elér vagy egy változatlan (ún. tengődő alakzatot) vagy egy ciklikus (ún. pulzáló alakzatot) állapotot. A változatlan állapotok stabil állapotok, a sejtek se nem fejlődnek, se nem mozognak tovább. A ciklikus állapotok reprodukálják önmagukat a generációváltások során.

## 9 Dokumentumok generálása

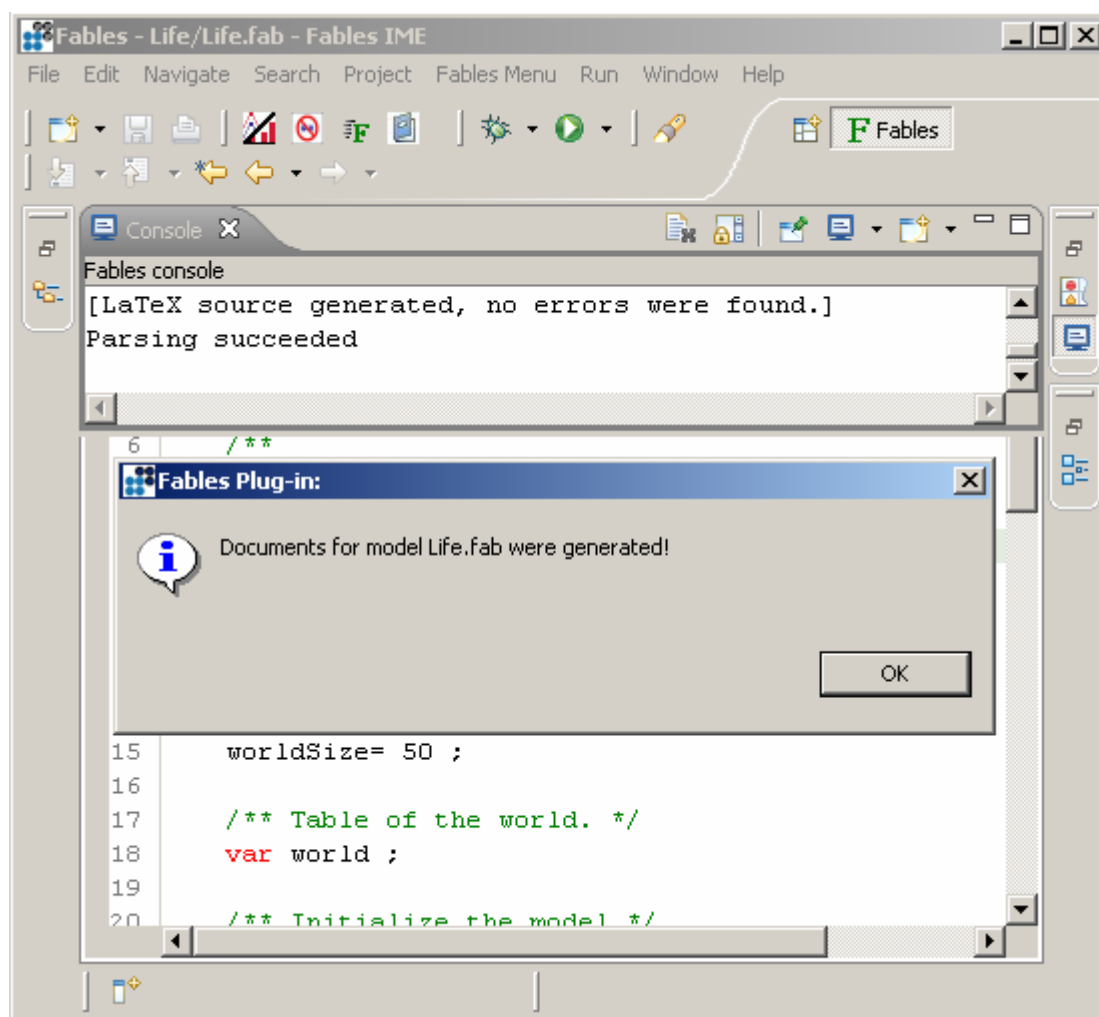
A Fables IME képes a leírt modellekből dokumentumokat generálni jónéhány ismert formátumban. A lehetséges formátumokról, és a dokumentációk generálásának részleteiről bővebben a Fables Kézikönyvben írunk.

Egy kiválasztott Fables modellből történő dokumentáció generáláshoz kattintson kétszer a modellre (a `Life.fab` fájl tökéletesen megfelel a célnak), majd nyomja meg a Generate Document gombot, amelyet vagy az eszköztáron, vagy a Fables menüben találhat meg (19. ábra)!



19. ábra

Kérjük tartsa szem előtt, hogy dokumentációk generálásához a modellnek szintaktikailag helyesnek kell lennie. Amint a forrásfájl értelmezése sikeresen befejeződött, egy értesítő üzenet jelenik meg a képernyőn, hogy a dokumentumok létrejöttek a projekt Documents könyvtárába (lásd 20. ábra).

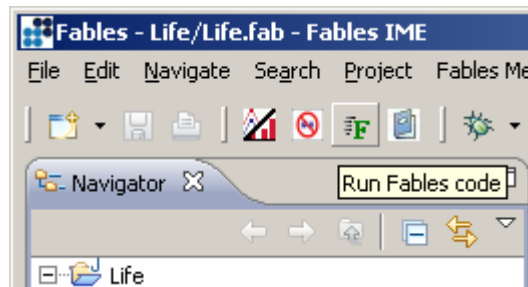
**20. ábra**

Nézze meg a modell HTML formátumú leírását, amelyet az IME generált a Life példaprogramhoz!

## 10 Többszöri fordítás és futtatás

Egy modell egymás után történő futtatását nevezzük többszöri fordításnak és futtatásnak. A Fables IME képes megtanulni egy modellhez az indítási konfigurációkat, ha első esetben kerülnek futtatásra, majd ezután mindig ezt a konfigurációt fogja a keretrendszer alkalmazni.

Számos kapcsoló létezik, amelyek befolyásolhatják a Repast kód tulajdonságait, ezek miatt fontosak a futtatási konfigurációk (ilyen például, hogy milyen grafikus felület készüljön a modelltől, erről részletesen a Fables Kézikönyvben írunk). Erre a feladatra létrehoztunk egy külön gombot az eszköztáron, a "Run Fables code" címkéjűt, amely minden egyes aktiválásnál ugyan azzal a konfigurációval egyszerre fordít és futtat egy Fables forrásfájlt. Ennek használatához az aktuális szerkesztőben egy Fables fájlnak kell lennie, ez lesz értelmezve a fordítás során.



21. ábra

## 11 Zárszó

Ha végére ért a kézikönyvnek, létre is hozta az első Fables szimulációját. Nehezen állítható, hogy a használata nehéz volt, vagy hogy túl sok programozói tudásra lett volna szükség a használatához. Természetesen összetettebb példák létrehozása nem ennyire egyszerű, de az leginkább a problémák matematikai komplexitása miatt lehetséges. A Fables-t arra terveztük, hogy segítsen egy matematikailag leírt modell megalkotásának minden aspektusában, amely az ágens alapú gondolkodásmód kereteibe beleillik.

A Fables telepítése után feltűnhet, hogy a szoftver számos beépített példaalkalmazással rendelkezik. Ezen alkalmazások tanulmányozása kiváló kiindulópont lehet a Fables nyelv elsajátításához, és a keretrendszer megismeréséhez.

## 12 Melléklet: A teljes forráskód

Ha bármi problémája akadt a kézikönyvben leírt modell implementálásával kapcsolatban, itt megtalálja annak teljes forráskódját, megszakítások és megjegyzések nélkül:

```
/**
 * A model for John Conway's Game of Life.
 */
model Life {

  /** The size of the world. */
  worldSize= 50 ; // PARAMETER

  /**
   * Notes if a cell is abandoned.
   */
  dead = 0;

  /** Notes if a cell is alive. */
  live = 1;

  /** The table of the world. */
  var world ;

  // Initialize the model
  startUp (worldSize) {
    seed(0) ;
    world := [ [ discreteUniform(dead, live)
                : y is [0..worldSize-1] ]
              : x is [0..worldSize-1] ] ;
  }

  /** Just to normalize x depending on the world's size. */
  norm (x) = x mod worldSize ;

  /** Determines the neighbourhood relation. */
  numberOfNeighbours(x, y) = size [ 1 :
    dx is [-1..1], dy is [-1..1]
    when not (dx==dy==0)
    and world(norm(x+dx))(norm(y+dy)) == live ] ;

  /** Determines the rule, now it is just the simple "23/3". */
  step(x, y) =
    (n==3 or ( world(x)(y)==live and n==2) )
    => live
    otherwise
    => dead
    where (
      n = numberOfNeighbours(x, y)
    );

  /**
   * Creation of the next generation.
   */
  newWorld = [ [ step(x, y)
                : y is [0..worldSize-1] ]
              : x is [0..worldSize-1] ] ;

  /** Will perform the generation change. */
  schedule Generator cyclic 1 {
    1 : world := newWorld ;
  }
}; // model{ Life }
```