



# Repast szimulációk Web alapú és részvételi változatának elkészítése a MASS/PET-el szoftverrel

*Repast Integrációs Manuál és tutorial*

*Készítette: Kozma Vilmos*

*az ELTE-IKKK*

*részére*



*A projekt az EU társfinanszírozásával, az Európa Terv keretében  
valósul meg.*

*2007 Október*

## Contents

|         |  |    |
|---------|--|----|
| 1       | Bevezetés  | 4  |
| 1.1     | Ágens-alapú modellezés                                     | 4  |
| 1.2     | MASS   | 4  |
| 1.3     | Cél  | 4  |
| 1.4     | Előnyök  | 5  |
| 1.5     | Előfeltételek  | 5  |
| 2       | Az integráció megvalósítása                                | 6  |
| 2.1     | Fontosabb funkcionális különbségek                         | 6  |
| 2.2     | A megvalósult rendszer                                     | 6  |
| 2.3     | Inkrementális integráció                                   | 6  |
| 2.4     | Az integráció szintjei                                     | 7  |
| 2.4.1   | Az alap szint  | 8  |
| 2.4.2   | Vizualizációk  | 8  |
| 2.4.2.1 | A Repast modell vizualizációi                              | 8  |
| 2.4.2.2 | Egyedi vizualizációk                                       | 8  |
| 2.4.3   | Az ágens láthatóság szint                                  | 8  |
| 2.4.4   | Az ágens konfigurálhatóság szint                           | 9  |
| 2.4.5   | A részvételi szint   | 10 |
| 2.4.6   | Törölhető ágensek  | 11 |
| 2.5     | Szabályok és korlátok                                      | 11 |
| 2.5.1   | Paraméter nélküli konstruktor a Repast modell-nek          | 11 |
| 2.5.2   | Security menedzser   | 12 |
| 2.5.3   | I/O műveletek  | 12 |
| 2.5.4   | Ütemezett metódusok közvetlen hívása                       | 12 |
| 3       | Tutorial: A Hőbogarak (Heatbugs) példa modell integrációja | 13 |
| 3.1     | A Hő-bogarak (Heatbugs) modell bemutatása                  | 13 |
| 3.2     | Szoftver feltételek  | 14 |
| 3.3     | Előkészületek  | 14 |
| 3.4     | Az alap szint elérése                                      | 14 |
| 3.5     | A modell installálása a PET-be                             | 15 |
| 3.5.1   | Források lefordítása és a Jar fájl készítése               | 15 |
| 3.5.2   | A Jar fájl bájtkód instrumentációja                        | 15 |
| 3.5.3   | A telepítés lépései  | 15 |
| 3.5.4   | A szimuláció futtatása                                     | 15 |
| 3.6     | Vizualizációk átvétele                                     | 18 |
| 3.7     | Ágensek kinyerése  | 19 |
| 3.7.1   | Ágensek definiálása a leíróban                             | 19 |
| 3.7.2   | Az AgentProvider interfész implementálása                  | 20 |
| 3.7.3   | Az ágens láthatóság kipróbálása                            | 21 |
| 3.8     | Ágensek konfigurálása                                      | 21 |
| 3.9     | Ágensek irányítása   | 23 |
| 3.10    | Making the bugs removable                                  | 27 |
| 4       | Befejezés  | 30 |

|         |                                      |    |
|---------|--------------------------------------|----|
| 5       | Melléklet: A modell család leíró xml | 31 |
| 5.1     | A leíró struktúrája                  | 31 |
| 5.1.1   | A model-visualization elem           | 31 |
| 5.1.2   | Az agent elem                        | 31 |
| 5.1.2.1 | A field elem                         | 32 |
| 5.1.2.2 | A field-defaults elem                | 33 |
| 5.1.2.3 | The user-interface-mappings element  | 33 |

# 1 Bevezetés

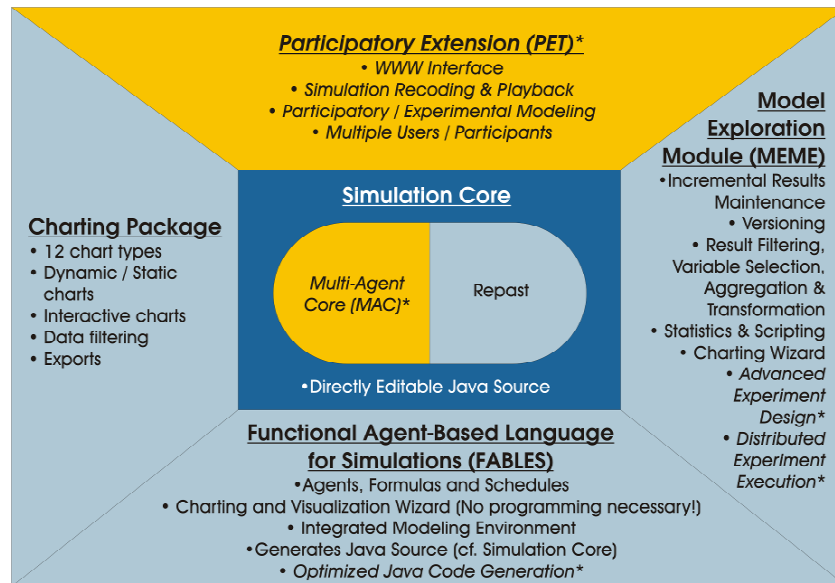
## 1.1 Ágens-alapú modellezés

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.

## 1.2 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

## 1.3 Cél

A dokumentum célja, hogy bemutassa, hogyan kell egy Repast modell-t a Participatory Extension (PET) szoftverbe integrálni. A PET részét képezi a Multi-Agent Simulation Suite (MASS) szoftvernek és alkalmas részvételi szimulációk futtatására. A szimulációkat

futtató motorja alapját a Repast és a saját fejlesztésű Multi-Agent Core (MAC) egyaránt képezik.

A dokumentum végig megy azokon a lépéseken, amelyeket egy Repast modell PET-be integrálásakor kell végrehajtani. Elmagyarázza a szabályokat, felmerülő fogalmakat, kiemeli a hibák lehetséges forrásait és kifejti az integrációt inkrementális megközelítésben.

## 1.4 Előnyök

Egy PET-be integrált Repast modell örökli a PET minden funkcionalitását, ami lehetővé teszi a modell írók számára, hogy létező modelljeiket új területeken használhassák. Az alábbi lista az integráció potenciális előnyeit foglalja össze:

- A Web-es környezetben futó szimulációt egyszerre több felhasználó is tudja nézni és irányítani.
- Az ágensek irányíthatóak a Web-en keresztül. Ilyenkor az irányító felhasználó döntése alapján „lép” az ágens
- Futás után a szimuláció visszajátszható. A visszajátszott szimuláció a felhasználók által irányított ágensek lépéseit is tartalmazza.
- A Repast modell vizualizációi elérhetőek a böngészőn keresztül.
- Az ágensek elő-konfigurálhatóak. Ez az ágensek modellhez való hozzáadását és az ágensek tulajdonságainak az egyenkénti beállítását jelenti még a szimuláció futása előtt.
- Mindezek mellett a modell továbbra is használható marad, mint normál Repast szimuláció.

## 1.5 Előfeltételek

A dokumentum olvasójától az alábbiakat várjuk el:

- Java kód értése és írása kezdő szinten
- Alapszintű jártasság Repast modellek készítésében
- A PET adminisztrátor szintű ismerete előny. (lásd: PET adminisztrátorok kézikönyve)

## 2 Az integráció megvalósítása

A MAC és a Repast szimulációs keretrendszerek interfészeik és osztályaik által támogatják az ágens alapú szimulációk írását. A két rendszerben előfordulnak közös, az ágens alapú szimulációhoz köthető fogalmak. A mód azonban, ahogy a két rendszer az egyes fogalmakat megvalósítja bizonyos helyeken eltér, ugyanis ezek egymástól függetlenül lettek kifejlesztve. Ez azt eredményezte, hogy az integrációs fejlesztés (a PET azon a fejlesztési folyamata, amely után az integráció lehetővé vált), jól kidolgozott legyen. Mindezek mellett a kialakult rendszer egy jól használható szimulációk futtatását támogató környezet.

### 2.1 Fontosabb funkcionális különbségek

- A PET JEE technológiára épül, és Web-es környezetben fut, míg a Repast egy swing-re épülő desktop alkalmazás.
- A PET-ben a felhasználó irányíthatja az ágenseket sőt, akár egyszerre több felhasználó irányíthat egy-egy ágenszt (részvételi szimuláció), míg a Repast esetén a szimuláció futásába menet közben nem lehet beavatkozni.
- A Repast-ban csak „globális” vizualizációk vannak, míg a PET-ben a vizualizáció alapvetően az ágensekhez kötött (a globális vizualizációkat a „világ” ágenshez kötjük) ezért alkalmas arra, hogy a „világot” az ágens szemszögéből mutassa meg.
- A Repast esetén az ágenseket a szimulációs modell adja hozzá és konfigurálja, míg PET-ben ezeket a feladatokat a felhasználó végezheti el. A PET modell konfigurációs folyamata az alábbi:
  - Szimuláció tulajdonságainak a beállítása.
  - Ágensek hozzáadása a modellhez.
  - Ágensek tulajdonságainak az állítása.

Fontos megemlíteni, hogy némi eltérés van a két rendszer között a modell család, modell és szimuláció fogalmakat érintően:

A PET-ben a modell család fogalmát a megírt szimuláció forrás kódja, konfigurációs állománya és egyéb fájljai jelentik. A modell család példányosítása által jön létre a modell, amelyből konfigurálás után hozhatunk létre szimulációt.

Repast-ban a modell-család fogalma nem létezik. Azonban az előbbi definícióból kiindulva mondhatjuk, hogy a Repast-ban a modell család fogalmát a forráskód és egyéb kísérő fájlok takarják, az ágensek és modell paraméterek értéke nélkül. Ezeket az értékeket tipikusan a `setup()` metódusban állítjuk be. A modell a teljes forráskód és kísérő fájlok, míg a szimuláció a modell egy példánya.

### 2.2 A megvalósult rendszer

A megvalósult rendszer mind a PET mind a Repast funkcionalitását örökölte, ám általában (a felhasználók szemszögéből), a Repast oldalt kellett erősíteni, hogy a megírt modell elérje a PET funkcionális szintjét. Mindazon által, a Repast alapú PET szimulációknak vannak bizonyos részei, amelyek nem szerepelnek egy natív PET szimulációban. Egy teljesen integrált Repast modell mindenre képes, amire egy PET modell, miközben használhatja a nagy mennyiségű Repast osztályt, hiszen az, az integrálás után is Repast modell marad. A legtöbb esetben ugyanaz a modell képes egyszerre futni PET és Repast környezetben is.

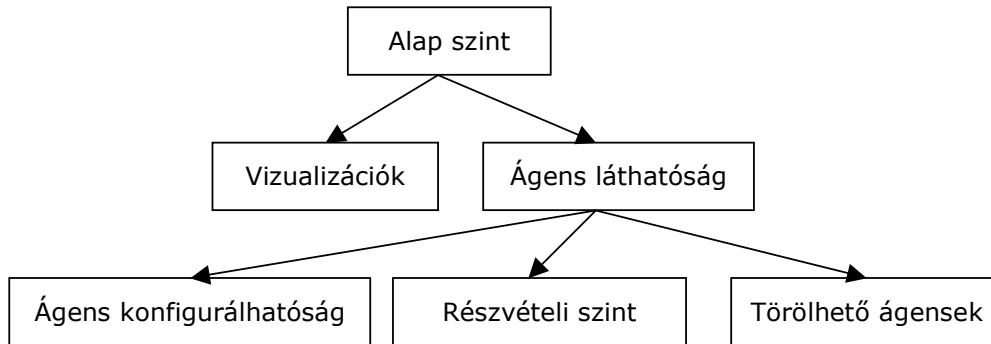
### 2.3 Inkrementális integráció

Az integráció folyamatát úgy terveztük meg, hogy a kivitelezés amennyire csak lehet egyszerű. A folyamatot fázisokra bontottuk. A fázisokat jelölik az integráció fokát, az integráció folyamata tehát inkrementális. Fontos azonban megjegyezni, hogy minimális

erőfeszítéssel (egy tíz sor hosszú egyszerűen strukturált xml fájl megírásával) már jó eredményt érhetünk el, amit az alap szintnek hívunk. Egy Repast modelltől nem várhatjuk el, hogy a PET-ben futtatva minden PET funkció a modell író segítségével tudjon működni, ezért helyenként a modellezőnek ki kell egészíteni a kódot. Azonban ez a munka nem nagy, és a jó tervezésnek köszönhetően nem is bonyolult.

## 2.4 Az integráció szintjei

Némely integrációs szint elérésének előfeltétele egy másik szint elérése, míg bizonyos szintek függetlenek a többitől. Megjegyezzük, hogy nem szükséges minden szintet megvalósítani. Az 1. ábra az integrációs szintek függőségi fáját ábrázolja.



1. ábra – Az integrációs szintek függőségei

Az alábbi lista az egyes szintek hozzáadott funkcionalitását részletezi:

- **Alap szint:** a felhasználó kezelheti a szimulációt azaz elindíthatja, leállíthatja és **visszajátszhatja**. Fontos megemlíteni, hogy a visszajátszás funkció már automatikusan működik alapszinten is. Ez lehetővé teszi a szimuláció elejétől végéig való lejátszását. A visszajátszás után a szimuláció lejátszását a végétől folytathatjuk. Alap szinten egy speciális ágens automatikusan bekerül a modellbe, amely a Repast modellt hivatott reprezentálni. A Repast modell paramétereit ennek a speciális ágensnek a tulajdonságai jelentik. Az ágens neve: „Repast Model Agent”
- **Vizualizációk:** A szimuláció vizualizációi (`DisplaySurface` és `OpenGraph` osztályok példányai) egy applet-en keresztül láthatóak a Web-en. A vizualizációk kinyerése bizonyos feltételek mellett automatikus. Erről részletesen később tárgyalunk.
- **Ágens láthatóság:** Ezen a szinten a szimuláció ágensei ismertek a PET számára, ezért az ágensek tulajdonságait a felhasználó interaktív módon állíthatja a Web-en keresztül.
- **Ágens konfigurálhatóság:** A konfiguráló felhasználó ágenseket adhat a Repast szimulációhoz, mielőtt az elindulna. A hozzáadott ágenseket egyenként lehet konfigurálni a tulajdonságaik és azok láthatóság és inicializáltság attribútumainak állítgatása által.
- **Részvételi szint:** Ezen a szinten az ágenseket a felhasználó irányíthatja a Web-en keresztül. Ez azt jelenti, hogy az ágens helyett, minden körben az irányító felhasználó a felelős „lépésért”.
- **Törölhető ágensek:** A PET (bizonyos beállítások mellett) törölhet egy ágens, ha az ágenset irányító felhasználó nem „lép” időben. Azonban a Repast modell segítségével a PET nem képes törölni az ágenseket a szimulációból. Ezért ennek a funkciónak a működéséhez a Repast ágensnek implementálnia kell a `Removable` interfészt.

## 2.4.1 Az alap szint

Az alapszint eléréséhez a fejlesztőnek írnia kell egy leíró fájlt, ami a PET számára szolgáltat szükséges információkat ahhoz, hogy a PET megfelelően értelmezni tudja a modellt. Ez egy szigorúan strukturált xml fájl amit a PET elvár ahhoz, hogy az installált Repast modellt felismerje. Az installáció után, a PET-ben megjelenik egy használatra kész, új modell család. Az installáció folyamatát a 3.5. fejezet tárgyalja részletesen. A leíró fájlról az 5. fejezetben olvashat bővebben.

## 2.4.2 Vizualizációk

### 2.4.2.1 A Repast modell vizualizációi

Amennyiben a leíró fájl hivatkozik a Repast modell vizualizációjára, az automatikusan elérhetővé válik a rendszerben. A PET a vizualizációkat úgy azonosítja, hogy hozzájuk rendel egy indexet. A leíróban erre az indexre hivatkozunk. Fontos megérteni a vizualizációk kinyerésének folyamatát. A PET a kinyerésre két módszert alkalmaz:

- Az `ai.aitia.mass.base.repast.abilities.DisplayProvider` interfészen keresztül:

```
public interface DisplayProvider
{
    public List getDisplays();
}
```

Az interfész egyetlen metódusa a `getDisplays()` visszaad egy listát, amely tartalmazza mindazokat a vizualizációkat, amelyeket a modell írója elérhetővé akar tenni a PET számára. A lista elemei `DisplaySurface` és `OpenGraph` példányok, vagy ezekből leszármazott osztályok példányai. A vizualizáció hivatkozási indexe vizualizáció listában betöltött indexe lesz.

- Ha az interfészt a modell nem implementálja a dolgok automatikussá válnak. A PET `DisplaySurface` és `OpenGraph` (és leszármazottak) példányait keresi a `SimModelImpl` osztály `simulation-event-listener and media-producer` listáiban. A hivatkozási index ebben az esetben a listákba való bejegyzés sorrendje lesz. A `SimEventListener` listában keres először. Ha egy vizualizáció mindkét listában szerepelne, akkor az első előfordulás indexe az érvényes index. Amennyiben a Repast modell nem a `SimModelImpl` osztályból származik (direkt vagy indirekt módon), akkor az egyetlen lehetőség az `DisplayProvider` interfész implementálása marad.

### 2.4.2.2 Egyedi vizualizációk

Lehetőség van egyedi vizualizáció készítésére minden ágens típushoz. Ez lehetőséget ad arra, hogy a modellíró olyan vizualizációt készítsen, amely a világot az ágens szemszögéből mutatja. Nézzük az alábbi példát: az ágensek foglyok, a világ egy labirintus a foglyok célja, hogy kijussanak a labirintusból. Ebben az esetben, ha az ágens (vagy az irányító felhasználó) a labirintus minden részét látná, akkor a feladat túl egyszerű lenne. Ezért a labirintus csak azon részeit mutatjuk meg, amit az irányított ágens éppen lát. Egy ilyen vizualizáció létrehozásához a modellírónak egy úgynevezett detektor metódussal kell kiegészítenie az ágens osztályt, írnia kell egy renderelő metódust, valamint ezeket hivatkozni a leíró fájlban. Itt most nem foglalkozunk ezzel a témakörrel, mert az ilyen típusú vizualizációk elkészítésének módja részletesen le van írva a PET modell írók kézikönyvében.

## 2.4.3 Az ágens láthatóság szint

A `SimpleModel` osztályból származó modellek kivételével a Repast modellek általában tetszőleges módon tartják nyilván az ágens objektumaikat. Ezért, ha a PET számára elérhetővé akarjuk tenni az ágenseinket, akkor a Repast modellünkkel implementálnunk kell a `ai.aitia.mass.base.repast.abilities.AgentProvider` interfészt. Ennek az interfésznek egy metódusa van, ami arra hivatott, hogy visszaadjon egy listát, amely

tartalmazza az összes olyan ágenszt, amit a modell író elérhetővé szeretne tenni a PET számára. Ezt a függvényt a PET a Repast modell `begin()` metódusának meghívása után bármikor hívhatja. Nem fontos, hogy mindig ugyanazt a lista objektumot adjuk vissza, de a hatékonyság miatt célszerű. A tárgyalat interfész az alábbi:

```
public interface AgentProvider
{
    public List getAgents();
}
```

#### 2.4.4 Az ágens konfigurálhatóság szint

Egy konfigurálható modell lehetővé teszi a felhasználó számára, hogy a Repast szimulációhoz, Web-es felületen keresztül adjon ágenseket hozzá, még mielőtt a szimuláció elindul. Ezután a hozzáadott ágensek tulajdonságait állíthatja be. Egy Repast modell az `ai.aitia.mass.base.repast.abilities.AgentConfigurator` interfész implementálása által válik „ágens konfigurálhatóvá”. Az interfésznek két metódusa van. Az első az alábbi:

```
public Object addAgent(String agentClass);
```

A metódusnak létre kell hoznia egy új példányt az `agentClass` paraméterben definiált típusból, és azt hozzá kell adnia a modellhez. A létrehozott ágenszt vissza is kell adnia, vagy hiba esetén a `ai.aitia.mass.base.repast.abilities.AddAgentError` osztály egy példányát. Az osztály forrása az alábbi:

```
public class AddAgentError
{
    public enum AddAgentErrorType {NO_MORE_OF_THIS_TYPE, A_DIFFERENT_TYPE_FIRST,
AGENT_TYPE_NOT_CONFIGURABLE, NOT_ADDED, UNKNOWN_ERROR}

    private String value = "";
    private AddAgentErrorType type;

    public AddAgentError(AddAgentErrorType type)
    {
        this.type = type;
    }

    public AddAgentError(AddAgentErrorType type, String value)
    {
        this.type = type;
        this.value = value;
    }

    public String getValue() {
        return value;
    }

    public void setValue(String value) {
        this.value = value;
    }
}
```

```

public AddAgentErrorType getType() {
    return type;
}

public void setType(AddAgentErrorType type) {
    this.type = type;
}
}

```

Előfordul néha, hogy az ágens hozzáadása valamilyen oknál fogva nem sikerül, vagy a szimuláció állapota miatt nem lehetséges. Ebben az esetben az `addAgent()` metódusnak egy `AddAgentError` objektummal kell visszatérnie, ami definiálja a hiba okát a `type` mező segítségével. A `value` mező részletesebb leírást adhat a hiba okáról. A `type` és `value` mezők tartalmát a PET megjeleníti a képernyőn. A `type` mező lehetséges értékei, az alábbiak a lehetnek:

- `NO_MORE_OF_THIS_TYPE`: Az ágensek száma az adott típusból elérte a maximális határt.
- `A_DIFFERENT_TYPE_FIRST`: Először más ágens típust kell hozzáadni a modellhez.
- `AGENT_TYPE_NOT_CONFIGURABLE`: Ennek a típusnak a konfigurálását a modell nem támogatja.
- `NOT_ADDED`: Az ágens nem lett hozzáadva tetszőleges oknál fogva.
- `UNKNOWN_ERROR`: Ismeretlen hiba történt.

Az `AgentConfigurator` interfész második metódusa az alábbi:

```

public boolean createDummyConfiguration();

```

A metódus feladata, a szimuláció olyan állapotba helyezése, ahol a `begin()` metódus meghívása biztonságos. Ahogy a név is sugallja, ezt a függvényt csak próba modelleknél hívjuk meg, és az ilyen szimulációt soha nem indítjuk ténylegesen. Erre a metódusra azért van szükség, mert a PET úgynevezett próba szimuláció objektumokat hoz létre, hogy információt gyűjtsön a Repast modell integrációs fokáról, a regisztrált vizualizációkról, illetve az ágens tulajdonságainak alapértelmezett értékeiről.

## 2.4.5 A részvételi szint

Feltételezzük, hogy minden ágensnek van legalább egy metódusa, amit a Repast ütemező meghív (nevezzük ütemezett metódusnak) közvetett vagy közvetlen módon. Az ütemezett metódusok hajtják végre a lépéseket minden körben. Amennyiben egy ágens felhasználó által vezérelt, akkor az ütemezett metódusok helyett, azoknak a felhasználó által hívható alternatívái kerülnek meghívásra, az eredeti ütemezett metódus hívása pillanatában és így az ágens lépését illetően a döntés áttolódik a felhasználó oldalára. Amennyiben a PET számára ismerté tesszük az ütemezett metódusokat és azok alternatív metódusait ill. bájt kód instrumentálással (byte code instrumentation) az igényeink szerint módosítjuk az ágens, akkor a felhasználók által vezérelt ágensek koncepciója megvalósíthatóvá válik. A bájt kód instrumentálás a leíró fájl alapján egy segédalkalmazással történik, amit a PET tartalmaz. Az irányítás átvétele alatt azt értjük, hogy az ütemezett metódus helyett egy, a felhasználó által kiválasztott metódus fog meghívódni. További lehetőség az irányításra, ha a felhasználó közvetlenül állítgatja az ágens tulajdonságait, szintén a Web-felületen keresztül.

Nézzük az alábbi példát: a szimuláció során mikor egy ágensre kerül a sor, az ágensnek a négy irány (észak, kelet, dél, nyugat) valamelyikébe kell elmozdulni. Az ütemezett metódus `move()` és a négy alternatív metódus: `moveNorth()`, `moveEast()`, `moveSouth()` és `moveWest()`. Amikor az ágens nem felhasználó által vezérelt akkor, az ütemező által meghívott `move()` metódus hajtódik végre. A metódus meghoz egy döntést az elmozdulás irányát illetően és azt végre is hajtja. Ezzel ellentétben, amikor az ágens irányított (egy felhasználó által a Web felületen keresztül), akkor az, az irányító döntéseit hajtja végre

az elmozdulás irányát illetően. Fontos megjegyezni, hogy az irányító által kiadott utasítások nem azonnal hajtódnak végre, hanem egy pufferbe kerülnek, a végrehajtás akkor történik, amikor az ütemező az eredeti `move()` metódust meghívja. A puffer mindig csak a legutolsó utasítást tartalmazza, így az előzőleg kiadott utasítás (ha volt ilyen), mindig felülíródik. Ez a mechanizmus automatikusan történik bájt kód instrumentálás segítségével. A `move()` metódus mindig akkor hívódik, amikor az ütemező azt meghívja. Ha az irányító hosszabb ideig nem ad utasítást az ágensnek, akkor az egy alapértelmezett „lépést” fog végrehajtani, nem csinál semmit, vagy megállítja az egész szimulációt, várva a felhasználói inputra. Ezt a viselkedést a modell `Wait` policy és a `Timeout` policy beállításai határozzák meg. Erről bővebb információt a PET adminisztrátorok kézikönyvében olvashat.

Egy szimuláció akkor tekinthető részvételinek, ha legalább egy ágens irányítható. Egy ágens irányítható ha:

- Legalább egy ütemezett metódusa hivatkozva van a leíró fájlban, vagy van legalább egy módosítható tulajdonsága. Utóbbi információt szintén a leíró fájlból lehet megtudni.
- A `controllable` tulajdonság értéke igaz.

A leíró fájlról részletesen az 5. fejezetben lesz szó.

Amikor a felhasználó megfog, vagy elenged egy ágens, a PET mindkét akcióról egy eseményt generál. Amennyiben a Repast szimuláció tudni akar ezekről az eseményekről, az ágensnek implementálnia kell a `ai.aitia.mass.base.repast.abilities.Controllable` interfészt.

```
public interface Controllable
{
    public void onTake();
    public void onRelease();
}
```

Az interfésznek két metódusát (`onTake()`, `onRelease()`) a PET hívja azért, hogy értesítse a Repast modellt az események bekövetkeztéről. Ez egy opcionális lehetőség, de hasznos lehet, amikor a fogott ágenseket más módon akarjuk megjeleníteni a vizualizációban. (Például más színnel kirajzolni)

## 2.4.6 Törölhető ágensek

Egy ágens a PET számára akkor törölhető a szimuláció közben, ha implementálja az `ai.aitia.mass.base.repast.abilities.Removable` interfészt. Ennek az interfésznek egy metódusa van, amit a PET akkor hív, ha az ágens el akarja távolítani a szimulációból. Az interfész tehát lényegében informálja a Repast modellt, hogy az ágens el kell távolítani, és magát a törlést már a modellnek kell véghez vinnie. Az interfész az alábbi:

```
public interface Removable
{
    public void remove();
}
```

## 2.5 Szabályok és korlátok

Az elkészült rendszerben van néhány szabály és korlát, amiket figyelembe kell venni. Ezek az alábbiak:

### 2.5.1 Paraméter nélküli konstruktor a Repast modell-nek

A Repast modellnek lennie kell paraméter nélküli konstruktorának, különben a PET nem tudja a modellt példányosítani.

## 2.5.2 Security menedzser

A Repast szimulációt futtató JEE környezet rendelkezhet Security menedzserrel. Ebben az esetben bizonyos, modellezők által írt kódok hibát okozhatnak és a szimuláció ezáltal hibás futást produkálhat. Ebben az esetben vagy távolítsa el a Security menedzsert az alkalmazás szerveréből, vagy cserélje le / távolítsa el a nem megfelelő részeket a kódból.

## 2.5.3 I/O műveletek

Ez a probléma akkor keletkezhet, amikor a Repast szimuláció írni vagy olvasni akar egy fájlból. I/O funkciók nem mindig működnek, mert a szimuláció egy JEE környezetben fut a szerver oldalon. A PET direkt módon nem akadályozza ezeknek a funkcióknak a végrehajtását, ezek sikere teljes mértékben a futtató környezeten múlik. A tipikus hibaforrások az alábbiak:

- Installált security menedzser (az alapértelmezett telepítéssel ez nem fordul elő)
- Nem megfelelő fájl struktúra
- Operációs rendszer szintű írási / olvasási engedély hiánya

## 2.5.4 Ütemezett metódusok közvetlen hívása

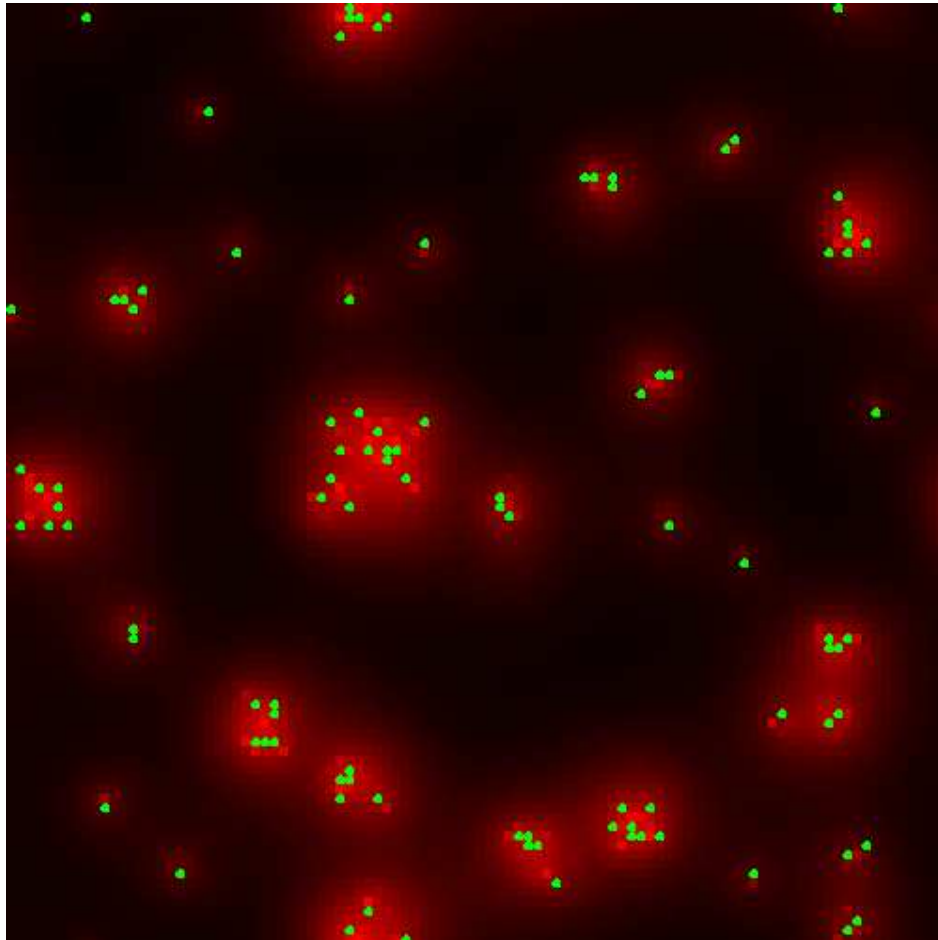
Egy Repast modell ágensei tipikusan rendelkeznek olyan metódusokkal, amiket az ütemező hív. Néha azonban a modell vagy más ágensek közvetlenül meghívják ezeket. Amennyiben egy ütemezett metódust definiáltunk a leíró fájlban és az ágens, irányítás alatt áll, ez nem várt működést eredményezhet, mert a pufferben tárolt akciók hajtódhatnak végre, ha a puffer nem üres.

## 3 Tutorial: A Hőbogarak (Heatbugs) példa modell integrációja

Ez a fejezet a gyakorlatban mutatja be egy példamodellen keresztül az integráció lépéseit. Példamodellünk a Heatbug modell, ami jól ismert modellező körökben.

### 3.1 A Hő-bogarak (Heatbugs) modell bemutatása

Az eredeti hő-bogár szimuláció hő-bogarokból áll, melyek hőt vesznek fel és bocsátanak ki és egy hő-térből (heatspce) ami ezt a hőt szétteríti a bogár környezetében. A bogaraknak van egy jellemző ideális hőmérsékletük, és a térben úgy mozognak, hogy ehhez a hőmérséklethez lehetőleg minél közelebb kerüljenek. A szimulációnak van egy vizualizációja (lásd.: 2. ábra), amelyen látható a bogarak mozgása és a hőt kibocsátó tér.



2. ábra – A hő-bogár vizualizáció

Az integráció után a Repast modell az alábbi funkcionalitással egészül ki:

- A modell Web-alkalmazásként fut, ami egy több felhasználós környezetet biztosít.
- A hő-bogarak irányíthatóvá válnak. A felhasználók megfoghatják, irányíthatják, majd elengedhetik a bogarakat. Az irányított bogarak az irányító felhasználó által kívánt irányba mozognak és más színnel vannak jelölve.
- Hő-bogarakat egyesével, vagy csoportosan lehet a szimulációhoz hozzáadni és konfigurálni.

- A szimuláció visszajátszható, ami tartalmazza a részvevő felhasználók lépéseit is.

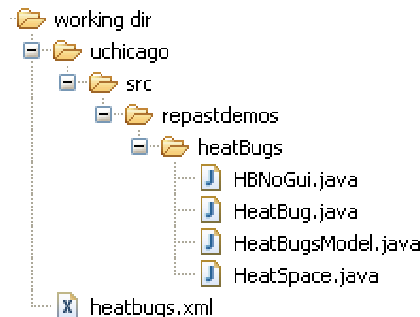
### 3.2 Szoftver feltételek

- Telepített JDK 5.0 vagy újabb
- Működő telepített PET alkalmazás

### 3.3 Előkészületek

- Válasszon egy mappát a merevlemezen ahová dolgozni fog. Ez lesz az ön munkakönyvtára ahol szerkeszteni tudja az állományokat a tutorial során.
- Másolja be a `<PET installation dir>/repast_integration/tutorial/sources/src` mappában található hő-bogár szimuláció fájljait ez előbb említett helyre.
- A tutorial során egy leíró fájlt fogunk készíteni. Hogy megkönnyítsük a fejlesztők munkáját a rendszerben egy sablon leíró fájlt tartalmaz, ami megtalálható a PET alkalmazás `<PET installation dir>/repast_integration/tutorial/sources/descriptor` mappájában. Másolja a `repast_descriptor.xml` fájlt a munkakönyvtárba és nevezze át `heatbugs.xml`-re. Ebben a könyvtárban egy `dtd` fájl is van, ami a modell család leíró xml fájl definíciója.
- A tutorial során hivatkozott interfészek a `<PET installation dir>/repast_integration/tutorial/build` könyvtárban vannak elhelyezve.

Az alábbi könyvtárszerkezet lett kialakítva:



3 ábra - A tutorial könyvtár struktúrája

### 3.4 Az alap szint elérése

Nyissa meg a leíró fájlt egy xml szerkesztővel (egy egyszerű text editor is jó) és módosítsa a fájl tartalmát az alábbira:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE model-family SYSTEM "repast_model_family.dtd">
<model-family>
  <family-name>Test Repast HeatBugs</family-name>
  <model-class>uchicago.src.repastdemos.heatBugs.HeatBugsModel</model-class>
  <image>../images/heatbugs.jpg</image>
  <description>Repast based heatBugs simulation</description>
  <model-visualizations/>
  <agents/>
</model-family>
  
```

Ezzel el is értük az alap szintet. Most installálhatja és futtathatja az új modell családot.

## 3.5 A modell installálása a PET-be

Az installálás három lépésből áll:

- Források lefordítása, és Jar fájl készítése
- A Jar fájl bájtkód instrumentációja
- A fájlok bemásolása a megfelelő helyre
- Szerver újraindítása

### 3.5.1 Források lefordítása és a Jar fájl készítése

A források lefordításával és a Jar fájl készítésével semmilyen speciális követelményt nem támasztunk, csak csinálja úgy ahogy rendesen tenné. Használhatja a `javac` és `jar` programokat, vagy valamilyen IDE-t, ami segít elvégezni ezeket a feladatokat.

### 3.5.2 A Jar fájl bájtkód instrumentációja

A létrehozott Jar fájlt instrumentálni kell. Ez a leíróban hivatkozott ágens osztályok bájtkód szintű módosítását jelenti, amely által azok alkalmassá válnak a PET számára, hogy az valóban ágensként tudja kezelni őket. A rendszer tartalmaz egy integráló programot ami a `<PET install dir>/repast_integration/lib/integrator.jar` mappában található.

- Ha Windows rendszeren dolgozik, akkor használhatja a `<PET install dir>/repast_integration/bin/integrate.bat` programot. Ebben az esetben az alábbi parancsot kell kiadni: `integrate.bat tutorial_descriptor.xml tutorial.jar`
- Egyéb (általános) esetben használja az alábbi parancsot: `java -cp ../lib/integrator.jar;../lib/asm-3.0.jar;../lib/commons-io-1.3.2.jar;../lib/commons-lang-2.0.jar;../lib/jdom.jar;../lib/log4j-1.2.11.jar;../lib/truexip-6.jar;../lib/xerces-2.6.2.jar;../lib/xercesImpl.jar;../lib/xml-apis.jar ai.aitia.mass.base.repast.asm.Repast2MACUtil tutorial_descriptor.xml tutorial.jar`. Ekkor az aktuális könyvtárnak a `bin` mappának kell lennie és a szükséges fájlokat (leíró fájl és Jar fájl), ebbe a mappába kell másolni.

### 3.5.3 A telepítés lépései

- Állítsa le a PET alkalmazást. Állítsa le az egész alkalmazás szerveret, vagy csak a PET alkalmazást. Amennyiben a rendszert a Windows-ra készített telepítő program használatával telepítette, akkor ezt az alábbi módon teheti meg. Nyissa meg a **Start Menu -> Control Panel -> Administrative Tools -> Services** panelt. Itt kattintson jobb egér gombbal az **Apache Tomcat** bejegyzésre, majd válassza ki a **Restart** funkciót.
- Másolja be az instrumentált Jar fájlt a PET Web-alkalmazás `WEB-INF/lib` mappába.
- Másolja be a leíró fájlt a PET Web-alkalmazás `WEB-INF/config/descriptors/repast` mappájába.
- Ezen a ponton végeztünk az installációval. Indítsa el az alkalmazás szerveret vagy PET-et (az első lépésben választottaktól függően)

### 3.5.4 A szimuláció futtatása

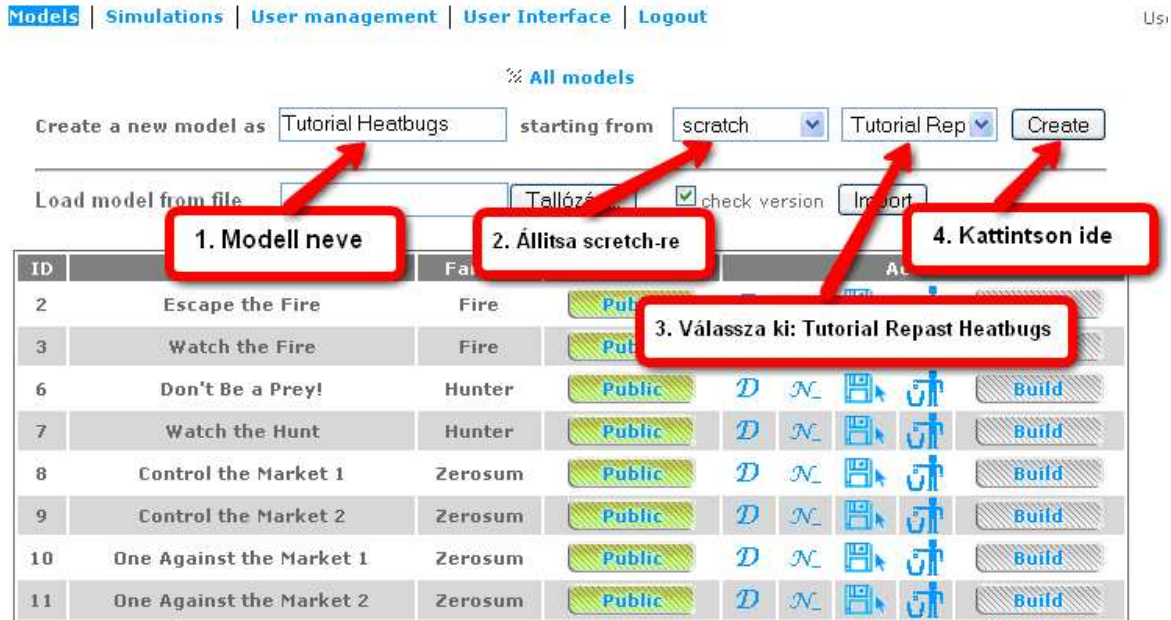
Nyisson egy új böngészőt és navigáljon a PET adminisztrátorok bejelentkező oldalára.

- Amennyiben a Windows-re készített telepítőből installálta a rendszert akkor kattintson a Start menüben található PET bejegyzés **Admin Login** menüjére
- Általános esetben írja be a PET alkalmazást futtató szerver IP címét a böngészőbe a `/pet/admin` szöveggel kiegészítve. (Például: `http://127.0.0.1/pet/admin`).

Lépjen be adminisztrátori jogosultsággal.

Belépés után a lap **Create new model** sávjában adjon egy nevet a leendő modellnek és állítsa a **Starting from** lenyíló listát **scratch**-re. Az újonnan installált modell családnak szerepelnie kell a következő lenyíló listában. Ha nem így van, akkor ellenőrizze a PET alkalmazás log-fájlját. Ezt a PET alkalmazás `WEB-INF/config/log4j.properties` fájljából

olvashatja ki a `log4j.appender.F.File` értéke alapján. Ha minden rendben van, akkor válassza ki a lenyíló listában az újonnan installált modell család nevét és kattintson a **Create** gombra. Lásd.: 4. ábra.



4. ábra – Modell létrehozása

A modell létrehozása után kattintson a **Private** feliratú gombra, hogy a modell publikus legyen, majd a konfiguráláshoz kattintson a nevére. A modell publikussá tétele ahhoz szükséges, hogy a normál felhasználók is láthassák a modellt. Lásd.: 5 ábra.



5 ábra – A modell publikussá tétele

Az alap integrációs szinten a modell konfigurálása az alábbi lépésekből áll:

- Repast modell paraméterek beállítása.
- A szimuláció tulajdonságainak a beállítása

Egy szimuláció létrehozásához modell beállítása után kattintson a **Build simulation** feliratú gombra. Lásd: 6. ábra.

[Models](#) | [Simulations](#) | [User management](#) | [User Interface](#) | [Logout](#)

↳ [Edit model \(Tutorial Heatbugs\)](#) | [List agents](#) | [Groups](#) | [Description](#) | [Build simulation](#)

⌘ [Edit model Tutorial Heatbugs \(18\) of type Tutorial Repast](#)

⌘ [Agents](#)

3. Szimuláció létrehozása

| Type         | Class                                      | Number | Actions |
|--------------|--|--------|---------|
| Repast model | ai.aitia.mass.base.repast.RepastModelAgent | 1      |         |

1. Repast modell paraméterek beállítása

⌘ [Simulation properties](#)

| Property                 | Class             | Value   |
|--------------------------|-------------------|---|
| Pause at                 | java.lang.Integer | -1  |
| Random seed              | java.lang.Long    | 1195478452750                                       |
| Release Policy           |                   | Continue <input checked="" type="checkbox"/>        |
| Timeout                  |                   | 300   |
| Timeout policy           | java.lang.Integer | Call on timeout <input checked="" type="checkbox"/> |
| Type of Event Logger DAO | java.lang.String  | Hibernate <input checked="" type="checkbox"/>       |
| Visualization            | java.lang.String  | None <input checked="" type="checkbox"/>            |
| Wait policy              | java.lang.Integer | Timeout <input checked="" type="checkbox"/>         |

2. Szimuláció paramétereinek konfigurálása. Kattintson az "Update" gombra.

6. ábra – A modell bekonfigurálása és egy szimuláció létrehozása

A modell paraméterek beállítását a 7. ábra mutatja.

[Models](#) | [Simulations](#) | [User management](#) | [User Interface](#) | [Logout](#)

↳ [Edit model \(Tutorial Heatbugs\)](#) | [List agents](#) | [Groups](#) | [Edit agent \(repastModelAgent\)](#)

⌘ [Edit agent #1302 of type repastModelAgent](#)

4. a modell további szerkesztéséhez kattintson ide

| Property      | Class             | Value | Visibility  | Initialized   |
|---------------|-------------------|-------|---|---|
| maxIdealTemp  | java.lang.Integer | 31000 | <input checked="" type="radio"/> true <input type="radio"/> false | <input type="radio"/> true <input checked="" type="radio"/> false |
| maxOutputHeat | java.lang.Integer | 10000 | <input checked="" type="radio"/> true <input type="radio"/> false | <input type="radio"/> true <input checked="" type="radio"/> false |
| minIdealTemp  | java.lang.Integer | 17000 | <input checked="" type="radio"/> true <input type="radio"/> false | <input type="radio"/> true <input checked="" type="radio"/> false |
| minOutputHeat | java.lang.Integer | 3000  | <input checked="" type="radio"/> true <input type="radio"/> false | <input type="radio"/> true <input checked="" type="radio"/> false |
| numBugs       | java.lang.Integer | 100   | <input checked="" type="radio"/> true <input type="radio"/> false | <input type="radio"/> true <input checked="" type="radio"/> false |
| worldXSize    | java.lang.Integer | 100   | <input checked="" type="radio"/> true <input type="radio"/> false | <input type="radio"/> true <input checked="" type="radio"/> false |
| worldYSize    | java.lang.Integer | 100   | <input checked="" type="radio"/> true <input type="radio"/> false | <input type="radio"/> true <input checked="" type="radio"/> false |

1. Property-k értékeinek állítása

2. Property-k átláthatóság és inicializáltság attribútumainak állítása

3. Beállítások mentése

7. ábra – A modell paraméterek beállítása

Új szimulációt a **Model edit** lap **Build simulation** linkjére kattintva hozhatunk létre. Kattintsunk rá, és lássuk ahogy a szimuláció fut. A vezérlő gombokkal tudjuk elindítani, leállítani és visszajátszani a szimulációt.

Mode: Simulation Simulation status: Ready

Name: Tutorial Repast HeatBugs 1 You are the owner of this simulation.

Type: Tutorial Repast HeatBugs

Created at: Mon Nov 19 13:30:57 CET 2007

Tick count: 0

**A kontroll gombok segítségével elindíthatja, leállíthatja, vagy újrajátszhatja a szimulációt.**

| Property                 | Class             | Value           |
|--------------------------|-------------------|-----------------|
| Current tick             | java.lang.Long    | 0               |
| Identifier               | java.lang.Long    | 13              |
| Last tick                | java.lang.Long    | 0               |
| Pause at                 | java.lang.Integer | -1              |
| Random seed              | java.lang.Long    | 1195475875640   |
| Release Policy           | java.lang.Integer | Continue        |
| Simulation status        | java.lang.Integer | Ready           |
| Timeout                  | java.lang.Long    | 300             |
| Timeout policy           | java.lang.Integer | Call on timeout |
| Type of Event Logger DAO | java.lang.String  | Hibernate       |
| Visualization            | java.lang.String  | None            |
| Wait policy              | java.lang.Integer | Timeout         |

8. ábra - Szimuláció indítása

Elindítás után látszani fog, ahogy a lépés számláló (Tick count) el kezd növekedni. Azonban most még nem látunk sokat, mert a Repast vizualizációkat nem lettek átvéve.

### 3.6 Vizualizációk átvétele

A leíró fájlban cserélje le az alábbi sort:

```
<visualizations/>
```

erre:

```
<model-visualizations>
  <model-visualization index="0" display-name="Default display"
    refresh-rate="100"/>
</model-visualizations>
```

Ezzel a részlettel azt deklaráljuk, hogy a modell nullával indexelt vizualizációját szeretnénk használni. További információért az indexelési mechanizmusról olvassa el a 2.4.2.1 fejezetet. Azzal, hogy hivatkozunk a leíróban a vizualizációra az indexén keresztül, a vizualizáció automatikusan kiválaszthatóvá válik, mint a modell vizualizációja.

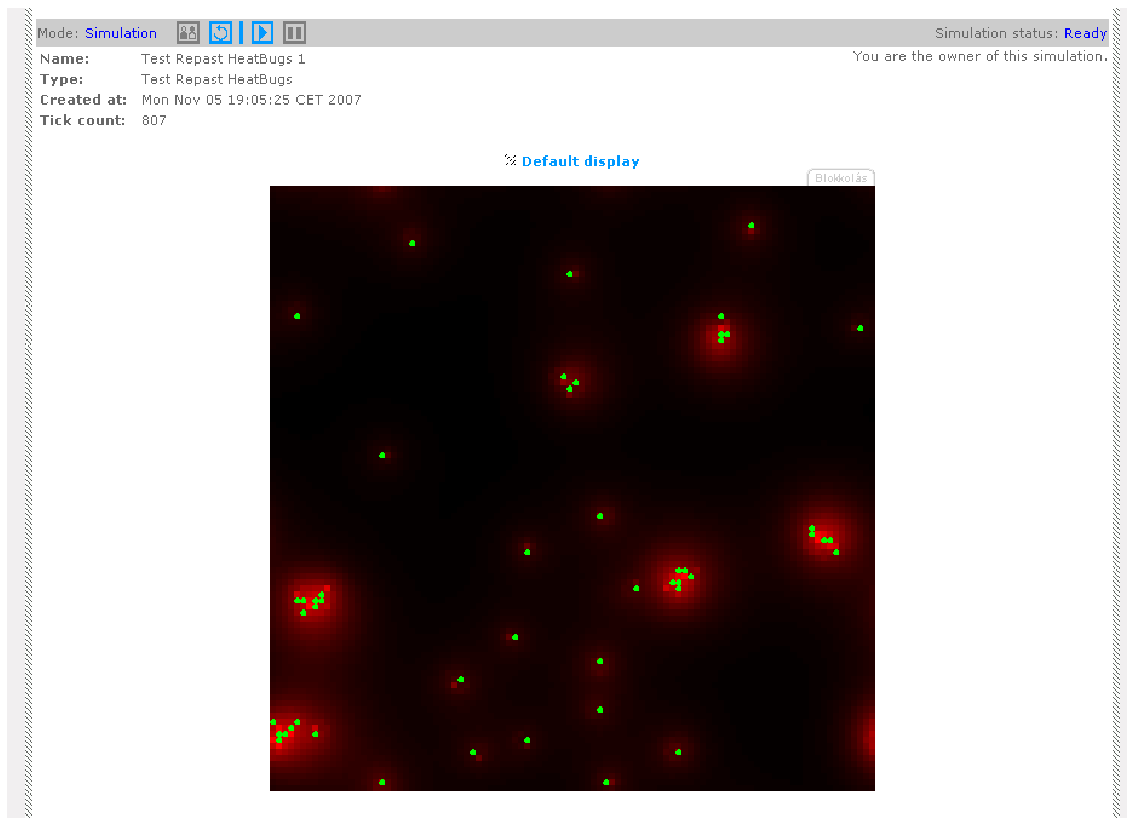
Most újra installálhatja a modell családot, a 3.5. fejezet utasításait követve. Vegye észre, hogy a Heatbug display megjelenik a szimuláció oldalán. Fontos tudni, hogy alpból nincs a modellhez vizualizáció kiválasztva, ezért azt külön meg kell tenni az Edit model oldalon. Lásd: 9. ábra és 10. ábra.

| Property                 | Class             | Value           |
|--------------------------|-------------------|-----------------|
| Pause at                 | java.lang.Integer | -1              |
| Random seed              | java.lang.Long    | 1195478452750   |
| Release Policy           | java.lang.Integer | Continue        |
| Timeout                  |                   | 300             |
| Timeout policy           |                   | Call on timeout |
| Type of Event Logger DAO | java.lang.String  | Hibernate       |
| Visualization            | java.lang.String  | Default display |
| Wait policy              | java.lang.Integer | Timeout         |

Update properties

Válassza ki a vizualizációt,  
majd kattintson az Update  
gombra.

9. ábra Vizualizáció kiválasztása



10. ábra – A Heatbugs vizualizáció

## 3.7 Ágensek kinyerése

Az ágensek kinyeréséhez definiálnunk kell azokat a leíróban, majd kiegészíteni a `HeatBugsModel` osztályt az `AgentProvider` interfésszel.

### 3.7.1 Ágensek definiálása a leíróban

A hő-bogár modellben kétféle ágens van: a hő-tér és a hő-bogár, azonban példánkban csak a hő-bogarakhoz adunk hozzáférést a PET számára. Egy ágens elérhetővé tételéhez definiálnunk kell azt a leíróban az `<agents>` elemen belül. Egészítse ki a leíróját az alábbi részlettel:

```
<agents>
```

```

<agent name="heatBug" class="uchicago.src.repastdemos.heatBugs.HeatBug">
  <fields>
    <field access="rw" name="unhappiness">
      <displayName>Unhappiness</displayName>
      <shortDescription>Unhappiness</shortDescription>
    </field>
    <field access="rw" name="x">
      <displayName>x coordinate</displayName>
      <shortDescription>x coordinate</shortDescription>
    </field>
    <field access="rw" name="y">
      <displayName>y coordinate</displayName>
      <shortDescription>y coordinate</shortDescription>
    </field>
    <field access="r" name="idealTemp">
      <displayName>Ideal temperature</displayName>
      <shortDescription>Ideal temperature</shortDescription>
    </field>
    <field access="r" name="randomMoveProb">
      <displayName>randomMoveProb</displayName>
      <shortDescription>randomMoveProb</shortDescription>
    </field>
  </fields>
  <field-defaults>
    <agent-field-default name="visualisation_Pet" visible="true"
      initialized="false" value="0" />
  </field-defaults>
  <user-interface-mappings />
  <detectors />
  <image>../images/bug.jpg</image>
  <visualizations>
    <model-visualization-for-agent index="0"
      display-name="Repast bugs" />
  </visualizations>
  <description>Heat bug</description>
</agent>
</agents>

```

A fenti részletben definiáljuk a hó-bogarat megvalósító osztályt, annak mezőit, ezen felül pedig a modell nullával indexelt vizualizációját is elérhetővé tesszük minden hó-bogár számára. Ezt a vizualizációt ki is válasszuk, mint a vizualizáció tulajdonság alapértelmezett értékét.

### 3.7.2 Az AgentProvider interfész implementálása

Adja hozzá a `HeatBugsModel` osztályhoz az `AgentProvider` interfészt és fejtse ki annak `getAgents()` metódusát az alábbiak szerint:

```

public class HeatBugsModel extends SimModelImpl implements AgentProvider
{
    ...

    public List getAgents() {
        return heatBugList;
    }
}

```

A fenti metódusban egyszerűen csak visszaadjuk a már létező `heatBugList` listát, amely tartalmazza az összes hő-bogár objektumot.

Most megint installálhatja a modell családot a 3.5. fejezet utasításait követve.

### 3.7.3 Az ágens láthatóság kipróbálása







Az Edit model oldalon láthatóak a modellben szereplő hivatkozott ágens típusok. Lásd: 11. ábra.

Models | Simulations | User management | User Interface | Logout U:

↳ Edit model (Tutorial Heatbugs) | List agents | Groups | Description | Build simulation

⌘ Edit model Tutorial Heatbugs (18) of type Tutorial Repast HeatBugs

⌘ Agents

| Type         | Class                                      | Number | Actions   |
|--------------|--|--------|---|
| Repast model | ai.aitia.mass.base.repast.RepastModelAgent | 1      |       |
| Heat bug     | uchicago.src.repastdemos.heatBugs.HeatBug  | ?      |    |





11. ábra – a hő-bogár ágens típus

Ezen felül, egy szimuláció létrehozása és az **Agents** almenüpont kiválasztása után a szimulációban szereplő ágensek is láthatóvá válnak, az így megjelenő listán. Lásd: 12. ábra.

Models | Simulations | User management | User Interface | Logout User: admin


↳ Simulation Control>Tutorial Repast HeatBugs 1 | Agents | Agent properties


⌘ Simulation Control (Tutorial Repast HeatBugs 1)

Mode: Simulation    

Simulation status: Ready  
You are the owner of this simulation.

Name: Tutorial Repast HeatBugs 1  
Type: Tutorial Repast HeatBugs  
Created at: Mon Nov 19 16:24:36 CET 2007  
Tick count: 0

⌘ Default display 



12. ábra Az **Agents** almenü kiválasztása

## 3.8 Ágensek konfigurálása

Az ágensek konfigurálása alatt azok egyenkénti vagy csoportos modellhez adását és tulajdonságainak állítását értjük. Ez a funkció úgy érhető el, ha a modell osztály implementálja az `AgentConfigurator` interfészt. Ez az osztály közvetlenül az `AgentProvider` interfészből öröklődik, és két metódussal rendelkezik: `addAgent(String agentClass)` valamint `createDummyConfiguration()`.

```

public Object addAgent(String agentClass) {
    if (HeatBug.class.getName().equals(agentClass)) {
        return createHeatBug();
    } else {
        return new AddAgentError(AddAgentErrorType.UNKNOWN_ERROR,
            "Unknown agent type.");
    }
}

public void createDummyConfiguration() {
}

```

Az alábbi kód implementálja a metódusokat a fentebb tárgyalt specifikáció szerint. Praktikus okokból a hő-bogarakat létrehozó kód részletet a `createHeatBug()` metódusba tettük.

```

private HeatBug createHeatBug() {
    int idealTemp = Uniform.staticNextIntFromTo(minIdealTemp, maxIdealTemp);
    int outputHeat = Uniform.staticNextIntFromTo(minOutputHeat, maxOutputHeat);
    int x, y;

    do {
        x = Uniform.staticNextIntFromTo(0, space.getSizeX() - 1);
        y = Uniform.staticNextIntFromTo(0, space.getSizeY() - 1);
    } while (world.getObjectAt(x, y) != null);
    HeatBug bug = new HeatBug(space, world, x, y, idealTemp, outputHeat,
        randomMoveProbability);
    heatBugList.add(bug);
    return bug;
}

```

A `buildModel()` metódus is megváltozott:

```

private void buildModel() {
    for (int i = 0; i < numBugs; i++) {
        createHeatBug();
    }
}

```

Amint az látszik, az egyszerűség kedvéért a hő-bogarak létrehozását az új metódusunkkal végezzük. A másik változtatás az, hogy a `space` és `world` objektumok létrehozását a `setup()` metódus végére tettük. Ez azért volt szükséges, mert az `addAgent()` metódust a PET a `buildModel()` előtt fogja meghívni, és egy ágens létrehozásához mindkét objektumra szükség van. Másolja az alábbi két sort a `setup()` metódus végére:

```

space = new HeatSpace(diffusionConstant, evapRate, worldXSize, worldYSize);
world = new Object2DTorus(space.getSizeX(), space.getSizeY());

```

Most installálja újra a modell családot a 3.5. fejezet utasításait követve és vegye észre az alábbi változásokat a Web-es felületen.

Az **Edit model** oldalon az **Add heatbugs** ikon szürkéről kékre változott jelezve, hogy a funkció elérhető. Lásd: 13. ábra.









**Models** | **Simulations** | **User management** | **User Interface** | **Logout**

↳ **Edit model (Tutorial Heatbugs)** | List agents | Groups | Description | Build simulation

⌘ **Edit model Tutorial Heatbugs (18) of type Tutorial Repast**

⌘ **Agents**

Heat bug-ok modellhez való hozzáadásához kattintson az ikonra.

| Type         | Class                                      | Number | Actions   |
|--------------|--|--------|---|
| Repast model | ai.aitia.mass.base.repast.RepastModelAgent | 1      |     |
| Heat bug     | uchicago.src.repastdemos.heatBugs.HeatBug  | 0 + ?  |     |

13. ábra – Új hő-bogarak hozzáadása a modellhez

Az ágensek modellhez adásának folyamatát a 14. ábra mutatja. Mint látható 10 hő-bogarat adunk a modellhez. A **controllable** tulajdonság értéke igaz, így a létrehozott szimulációban a 10 hő-bogár irányítható lesz. A hő-bogarak koordinátái 50, 50-re lettek állítva, ezért az így hozzáadott bogarak a vizualizáció közepén fognak megjelenni az induláskor. A ideális hőmérsékletet 20000-re állítottuk.

**Models** | **Simulations** | **User management** | **User Interface** | **Logout**

↳ **Edit model (Tutorial Heatbugs)** | List agents | Groups | **New agent (Type Tutorial Repast)** | **new agent - heatBug**

1. Property-k értékeinek beállítása

| Property          | Class             | Value   | Visibility  | Initialized   |
|-------------------|-------------------|---|---|---|
| Controllable      | java.lang.Boolean | <input checked="" type="radio"/> true <input type="radio"/> false | <input type="radio"/> true <input checked="" type="radio"/> false | <input checked="" type="radio"/> true <input type="radio"/> false |
| Detectors visible | java.lang.Boolean | <input type="radio"/> true <input checked="" type="radio"/> false | <input type="radio"/> true <input checked="" type="radio"/> false | <input checked="" type="radio"/> true <input type="radio"/> false |
| Ideal temperature | java.lang.Integer | 20000   | <input checked="" type="radio"/> true <input type="radio"/> false | <input checked="" type="radio"/> true <input type="radio"/> false |
| randomMoveProb    | java.lang.Float   | 0.0   | <input checked="" type="radio"/> true <input type="radio"/> false | <input checked="" type="radio"/> true <input type="radio"/> false |
| Unhappiness       | java.lang.Double  | 0.0   | <input checked="" type="radio"/> true <input type="radio"/> false | <input checked="" type="radio"/> true <input type="radio"/> false |
| User step size    | java.lang.Integer | Small   | <input checked="" type="radio"/> true <input type="radio"/> false | <input checked="" type="radio"/> true <input type="radio"/> false |
| Visible           | java.lang.Boolean | <input checked="" type="radio"/> true <input type="radio"/> false | <input type="radio"/> true <input checked="" type="radio"/> false | <input checked="" type="radio"/> true <input type="radio"/> false |
| Visualization     | java.lang.String  | Repast b  | <input type="radio"/> true <input checked="" type="radio"/> false | <input checked="" type="radio"/> true <input type="radio"/> false |
| x coordinate      | java.lang.Integer | 50  | <input checked="" type="radio"/> true <input type="radio"/> false | <input checked="" type="radio"/> true <input type="radio"/> false |
| y coordinate      | java.lang.Integer | 50  | <input checked="" type="radio"/> true <input type="radio"/> false | <input checked="" type="radio"/> true <input type="radio"/> false |

2. Állítsa be a tulajdonságok láthatóság és a inicializáltság attribútumait

3. Hozzáadni kívánt ágensek száma

Add  piece(s)...

4. kattintson az Add gombra

14. ábra Ágensek hozzáadása a modellhez

### 3.9 Ágensek irányítása

Ahhoz, hogy a PET-be az ágensek irányíthatóak legyenek, valamelyest ki kell egészítenünk az ágens osztályt és a leíró. Először adjuk hozzá az alábbi `<field>` elemet a leíróhoz a `<fields>` rész végére:

```
<fields>
...
...
<field access="rw" name="userStepSize">
  <displayName>User step size</displayName>
```

```

<shortDescription>Describes how large a step is
    that a user can take with one action
</shortDescription>
<constants>
    <constant name="Small" value="1"/>
    <constant name="Medium" value="2"/>
    <constant name="Larger" value="3"/>
    <constant name="Even more large" value="4"/>
    <constant name="Largest" value="5"/>
</constants>
</field>
</fields>

```

A fenti xml részlet egy új tulajdonságot definiál a hő-bogár osztálynak konstans értékekkel. Az új tulajdonság segítségével tudjuk beállítani, hogy a bogár „mekkorát lépjen” a különböző irányokba. Ez csak egy opcionális kiterjesztése a modellnek ugyan, viszont jó arra, hogy demonstráljuk a konstans értékek működését. Egészítse ki `HeatBug` osztályt az alábbi tulajdonsággal és annak getter / setter metódusával:

```

private int userStepSize;

public int getUserStepSize() {
    return userStepSize;
}

public void setUserStepSize(int userIncrement) {
    this.userStepSize = userIncrement;
}

```

Adjunk `HeatBug` osztályhoz felhasználók által meghívható alternatív függvényeket és egészítsük ki a leírot is a megfelelő hivatkozásokkal. Az ütemezett metódusunk neve `step()`. Az alábbi négy alternatív metódust adtunk hozzá az ágenshez: `left()`, `right()`, `up()`, `down()`.

```

public void left() {
    x = x-userStepSize < 0 ? world.getSizeX() -
        Math.abs(x-userStepSize) : x-userStepSize;
}

public void right() {
    x = x+userStepSize > world.getSizeX() ?
        x+userStepSize-world.getSizeX() : x+userStepSize;
}

public void up() {
    y = y-userStepSize < 0 ? world.getSizeY() -
        Math.abs(y-userStepSize) : y-userStepSize;
}

```

```
public void down() {
    y = y+userStepSize > world.getSizeY() ? y+
        userStepSize-world.getSizeY() : y+userStepSize;
}
```

Cserélje le a leíróban az alábbi sort:

```
<user-interface-mappings/>
```

Az alábbi részlettel:

```
<user-interface-mappings>
  <scheduledMethod name="step" visible="true" displayName="Move"
    shortDescription="Move the heatbug">
    <timeoutMethod name="step"/>
    <humanCalledMethod name="left" displayName="Left"
      shortDescription="Move left"/>
    <humanCalledMethod name="right" displayName="Right"
      shortDescription="Move right"/>
    <humanCalledMethod name="up" displayName="Up"
      shortDescription="Move up"/>
    <humanCalledMethod name="down" displayName="Down"
      shortDescription="Move down"/>
  </scheduledMethod>
</user-interface-mappings>
```

Installálja újra a modell családot a 3.5. fejezetben leírtak alapján.

Mint opcionális lehetőség az ágens osztályok implementálhatják a [Controllable](#) interfészt, ami által lehetővé válik a Repast modell számára, hogy értesítést kapjon az ágensek megfogásáról és elengedéséről. Példánkban ezt mi arra használjuk, hogy más színnel jelenítsük meg az irányított ágenseket. Adjuk hozzá a [HeatBug](#) osztály forrásához az alábbi:

```
private Color color = Color.GREEN;

public void onTake() {
    color = Color.WHITE;
}

public void onRelease() {
    color = Color.GREEN;
}
```
















Hogy alkalmazzuk a kiválasztott színt a kirajzolásnál, írjuk át a [HeatBug](#) osztály [draw\(SimGraphics g\)](#) függvényét az alábbira:

```
public void draw(SimGraphics g) {
    g.drawFastRoundRect(color);
}
```

Eredményképpen az irányított ágens fehér színnel jelenik meg a képernyőn. Most teszteljük a kódot a modell család újra installálása után. Kövessük a 3.5. fejezetben leírtakat.

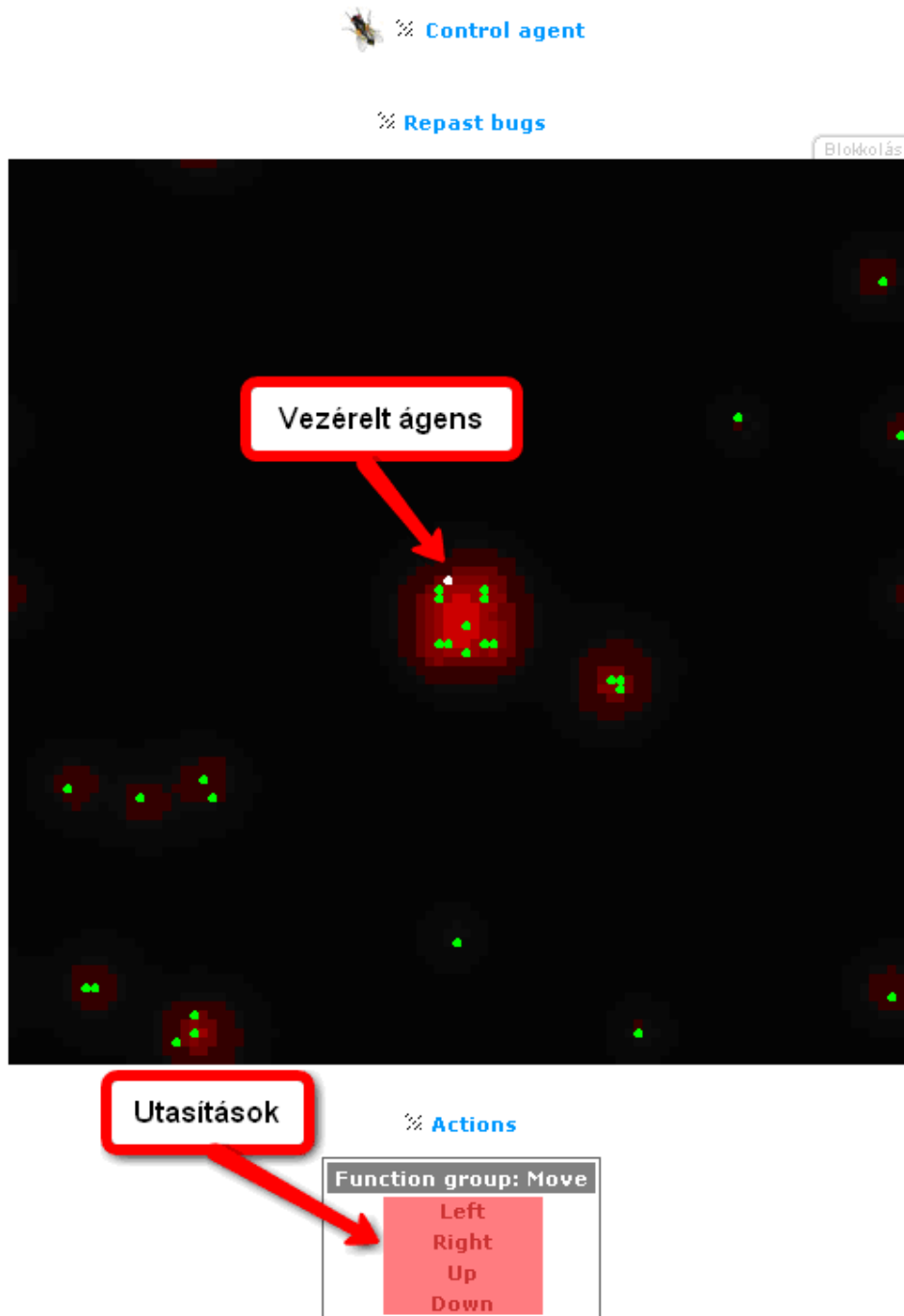
**Simulation Control** oldal **Agents** linkjére kattintva a szimulációban résztvevő ágensek listáját láthatjuk. Az ágenseket (egyszerre csak egyet) a Controll agent ikonra kattintva tudjuk megfogni és irányítani. Lásd: 15. ábra.

🔗 **Agent list**

| Id   | Type   | Added by | Controlled | Actions   |
|------|--|----------|------------|---|
| 1302 |  Repast model | User     | false      |   |
| 1353 |  Heat bug     |          |            |   |
| 1354 |  Heat bug     |          |            |   |
| 1355 |  Heat bug     | User     | false      |   |
| 1356 |  Heat bug     | User     | false      |   |

15. ábra – ágens megfogása

Az ágenseket a megfogás után irányítani a **Function group** rész alatt található parancsokkal tudjuk. Az irányított hő-bogarak jól megkülönböztethetőek a többitől, mert azok fehérrel vannak megjelenítve. Próbálja meg kattintgatni a parancsokra a szimuláció futása közben. Lásd: 16. ábra.



16. ábra – Ágens irányítása

### 3.10 Making the bugs removable

A PET-ben (bizonyos beállítások mellett) az ágensek törölhetőek, ha a felhasználó nem ad időben utasítást az irányítást illetően. A PET ezt a modellíró segítségével nem tudja megvalósítani, hiszen az ágens objektum tárolása a Repast modellben történik. Ezt a funkcionalitást úgy tudjuk elérni, hogy kiterjesszük a `HeatBug` osztályt a `Removable` interfésszel az alábbi módon:

```
public void remove() {
    bugList.remove(this);
}
```

```
}

```

Az interfésszel a `HeatBug` osztályt egészítettük ki, noha az ágensek listája a modell osztályban van tárolva. Ezért szükséges, hogy minden ágensben elérhetővé tegyük ezt a listát, hogy adott esetben törölhessék önmagukat a szimulációból. Tekintse meg az alábbi kódot:

```
private ArrayList bugList;

public void setBugList(ArrayList bugList) {
    this.bugList = bugList;
}

```

Minden hő-bogár létrehozása után meg kell hívunk a `setBugList()` metódust, ezért módosítsuk a modell osztály `createHeatBug()` metódusát az alábbira:

```
private HeatBug createHeatBug() {
    int idealTemp = Uniform.staticNextIntFromTo(minIdealTemp, maxIdealTemp);
    int outputHeat = Uniform.staticNextIntFromTo(minOutputHeat, maxOutputHeat);
    int x, y;

    do {
        x = Uniform.staticNextIntFromTo(0, space.getSizeX() - 1);
        y = Uniform.staticNextIntFromTo(0, space.getSizeY() - 1);
    } while (world.getObjectAt(x, y) != null);
    HeatBug bug = new HeatBug(space, world, x, y, idealTemp, outputHeat,
        randomMoveProbability);
    bug.setBugList(heatBugList);
    heatBugList.add(bug);
    return bug;
}

```

Ezen a ponton a Repast modellünket a lehető legnagyobb mértékben integráltuk a PET-be. Teszteljük a modell családot újrainstallálás után. Az újrainstalláláshoz kövessük a 3.5. fejezet utasításait. Az ágensek törlésének funkcióját a modell **Wait policy**, **Timeout policy**, és **Release policy** tulajdonságának 17. ábra szerinti beállításával tudjuk tesztelni.

#### ☞ Simulation properties

| Property                 | Class             | Value           |
|--------------------------|-------------------|-----------------|
| Pause at                 |                   | -1              |
| Random seed              |                   | 1195654856423   |
| Release Policy           | java.lang.String  | Remove agent    |
| Timeout                  | java.lang.Integer | 300             |
| Timeout policy           | java.lang.Integer | Release agent   |
| Type of Event Logger DAO | java.lang.String  | Hibernate       |
| Visualization            | java.lang.String  | Default display |
| Wait policy              | java.lang.Integer | Timeout         |

Update properties

A szimuláció policy tulajdonságainak állítása

### 17. ábra – A policy tulajdonságok állítása

Amennyiben az ábra szerint állítjuk be a modell tulajdonságait, akkor az irányított ágens törölődni fog a szimulációból, ha az irányító felhasználó nem „lép” egy bizonyos, a **Timeout** tulajdonságban beállított időtartamnyi ideig.

## 4 Befejezés

A PET egy ágens alapú modellező keretrendszer, amely támogatja a részvételi szimulációk fejlesztését és lehetőséget ad, hogy azokat Web-es környezetben futtassuk. A szimulációk futtatását két különböző motor végzi. Az egyik a natív Multi-Agent Core (MAC) nevű, amely saját fejlesztésű, a másik pedig a Repast-ra épülve lehetővé teszi létező Repast-ra megírt modellek egyszerű integrálását és futtatását. A PET felruhazza a Repast szimulációkat mindazokkal a képességekkel, amelyek magában megvannak, ami által a Repast szimulációk Web-en futtathatóak és részvételieliek lesznek.

## 5 Melléklet: A modell család leíró xml

Egy megfelelően installált leíró fájl segítségével a rendszer képes felismerni és megfelelő mennyiségű információt szerezni a rendszerbe telepített modell családról. Minden modell családhoz saját leíró fájl tartozik. A leíró fájjal szemben támasztott általános követelmények az alábbiak:

- A fájl neve tetszőleges, de a kiterjesztés xml kell hogy legyen.
- A fájlt a PET alkalmazás `WEB-INF/descriptors/repast/` könyvtárában kell elhelyezni.
- A fájl tartalmának meg kell felelnie a `WEB-INF/descriptors/repast/repast_model_family.dtd` xml definíciós fájlban leírtaknak.

Most vizsgáljuk meg a fájl struktúráját, ami a fent említett XML Document Type Definition (dtd) fájlra felel meg.

### 5.1 A leíró struktúrája

A gyökér elem neve: `model-family`. Gyerekei az alábbiak:

- `family-name`: a modell család neve
- `model-class`: a Repast modell osztálynak a neve
- `image`: a modellhez rendelt kép fájl elérési útja. Az útvonal a `WEB-INF` könyvtárhoz képest relatív.
- `description`: a modell család leírása
- `model-visualizations`: `model-visualization` elemek listáját tartalmazza. A `model-visualization` elemet a következő alfejezet tárgyalja.
- `agents`: `agent` elemek listáját tartalmazza. Az `agent` elemet az 5.1.2 fejezet tárgyalja.

#### 5.1.1 A model-visualization elem

Minden `model-visualization` elem a használt Repast modell egy display-ére hivatkozik. Az elem törzse üres és attribútumai az alábbiak:

- `index`: A hivatkozott display indexe. Egy display indexét az alábbi két módon határozhatjuk meg:
  - Ha a modell implementálja a `DisplayProvider` interfészt, akkor az index az ebben az interfészben deklarált `getDisplay()` függvény által visszaadott lista objektumainak indexe alapján értelmezendő.
  - Ellenkező esetben az index a `SimModelImpl` osztály event-listener és media producer listáiba való bejegyzési sorrend alapján kerül megállapításra.
- `display-name`: A vizualizáció képernyőn megjelenített neve. Ha üres, akkor az érték a display objektumból kerül kinyerésre. Ha az is üres, akkor az index lesz a megjelenítendő név.
- `width`: A vizualizációs applet szélessége. Elhagyás esetén az applet szélessége a display eredeti szélessége lesz.
- `height`: A vizualizációs applet magassága. Elhagyás esetén az applet magassága a display eredeti magassága lesz.
- `refresh-rate`: A vizualizációs applet frissítési intervalluma milliszekundumban.

Ha egy applet-et egyszer már hivatkoztunk, akkor a felhasználó által az automatikusan elérhetővé válik a modell beállításai oldalon.

#### 5.1.2 Az agent elem

Minden `agent` elem egy ágens típust definiál. Attribútumai az alábbiak:

- **class**: az ágens típus osztályának a neve
- **name**: az ágens típus hivatkozási neve. (a modell családon belül egyedinek kell lennie)

Az **agent** elem gyerek elemei:

- **fields**: **field** elemek listáját tartalmazza. Minden **field** elem egy mezőjét írja le ez adott ágens típusnak. A **field** elemet a következő alfejezet tárgyalja.
- **field-defaults**: Lehetővé teszi alapértelmezett értékek hozzárendelését az ágens objektumokhoz. Bővebben a 0. fejezetben van róla szó.
- **user-interface-mappings**: Leírja, hogy az ágens osztály mely metódusai hívódnak a Repast scheduler objektum által és hogy ezeket milyen alternatív (felhasználók által hívható) metódusokkal lehet kiváltani.
- **detectors**: **detector** elemek listáját tartalmazza. A **detector** elemnek üres a törzse és csak egy **name** attribútuma van, ami az ágens egy metódusának nevét tartalmazza. A hivatkozott metódusnak **void**-al kell visszatérnie, és nem lehet paramétere. A detektor metódusokat ágensek hívják, hogy más ágensekről nyerjenek információt. Ezeknek tipikusan a vizualizációkban van szerepe.
- **image**: az ágens típushoz rendelt kép elérési útja. Az útvonal relatív a **WEB-INF** könyvtárhoz.
- **visualizations**: **visualization** elemek listáját tartalmazza. Két gyerek eleme van:
  - **model-visualization-for-agent**: Ennek az elemnek a segítségével lehetséges, hogy egy modellhez rendelt Repast display-t hozzárendeljünk egy ágens típushoz. Az elem üres, és három attribútuma van:
    - § **index**: a modell display hivatkozási indexe
    - § **display-name**: A vizualizáció képernyőn megjelenített neve
    - § **refresh-rate**: az applet frissítési rátája milliszekundumban
  - **visualization**: A modellhez külön írt vizualizációkat lehet itt definiálni.
- **description**: Az ágens típus megjelenítési neve.

### 5.1.2.1 A field elem

Minden **field** elem az ágens típus egy mezőjét írja le. Két attribútuma van:

- **name**: A mező (property) neve
- **access**: A mezőhöz rendelt engedélyeket definiálja. A lehetséges értékek és jelentésük:
  - **r**: csak olvasás
  - **rw**: írás / olvasás
  - **no**: a mező nem látható
- **class**: Ez az attribútum opcionális. A mező osztályát definiálja.

A **field** elemnek az alábbi gyermek elemei lehetségesek:

- **displayName**: A mező megjelenített neve
- **shortDescription**: Egy rövid leírása a mezőnek
- **constants**: **constant** elemek listáját tartalmazza. Egy **constant** elem egy nevesített értéket ír le. Ez olyankor lehet hasznos, amikor egy értéket könnyen értelmezhető módon akarunk leírni ahelyett, hogy számokat íránk. Jó példa erre, amikor egy mezőnek 3 értéke van melyek különböző állapotokat jelölnek. Az állapotokat számokkal jelöljük. Például: 0=hideg, 1=langyos, 2=meleg. Az elemnek két attribútuma van:
  - **name**: A megjelenített név
  - **value**: A névhez rendelt érték.

### 5.1.2.2 A field-defaults elem

Ezt az elemet arra használjuk, hogy alapértelmezett értéket valamint láthatósági és inicializációs értékeket definiáljunk egy adott ágens mezőinek. Ezen felül, a felhasználó ágens csoportokat is definiálhat, amelyek azonos mező beállításokkal jönnek létre. Az elemnek két gyerek eleme van:

- **basic-agent-field-defaults**: Ezzel az elemmel a **controllable** és **visible** ágens mezőknek definiálhatunk alapértelmezett értékeket. Csak egy előfordulása lehet, a törzse üres és az alábbi attribútumai vannak:
  - **controllable**: Logikai érték, ami azt mutatja, hogy az ágens kontrollálható-e
  - **visible**: Logikai érték, ami azt mutatja, hogy az ágens látható-e
- **agent-field-default**: Ezzel az elemmel adhatunk alapértelmezett értéket az ágens egy tetszőleges mezőjének, valamint a mező **initialized** és **visible** tulajdonságainak. Az elemnek az alábbi attribútumai vannak:
  - **name**: a mező neve
  - **visible**: Logikai változó, ami mutatja, hogy a mező látható-e alpból.
  - **initialized**: Logikai változó, ami mutatja, hogy a mező inicializált-e alpból. Ha az értéke **false** akkor lehetőség van az ennek a mezőnek az értékét állítani az ágenseken az „Agent Properties” lapon.
  - **value**: a mező alapértelmezett értéke
- **set**: Az előző két elem (**basic-agent-field-defaults** és **agent-field-default**) minden ágensre vonatkozott az adott típusban. Ez az elem ugyanazt csinálja kivéve, hogy az érintett ágensek számát most a **count** attribútum definiálja. Két gyermek eleme van:
  - **basic-agent-fields**: Ugyanaz mint **basic-agent-field-defaults** de az érvényességét a szülő elem **count** attribútuma határozza meg.
  - **agent-field**: Ugyanaz mint **agent-field-defaults** de az érvényességét a szülő elem **count** attribútuma határozza meg.

### 5.1.2.3 The user-interface-mappings element

Az elem **scheduledMethod** elemek listáját tartalmazza. Az ütemezett metódus fogalmának jobb megértése érdekében olvassa el a fejezetet. A **scheduledMethod** elem a **name** attribútuma segítségével hivatkozik egy ütemezett metódusra. Minden **scheduledMethod** elem egy funkció csoportot definiál, amelynek legalább egy de lehet több alternatív metódusa. A **scheduledMethod** elemnek az alábbi attribútumai vannak:

- **name**: A hivatkozott ütemezett metódus neve
- **visible**: Logikai érték, amely meghatározza, hogy a definiált funkció csoport megjelenik-e a képernyőn.
- **displayName**: A funkció csoport képernyőn megjelenített neve
- **shortDescription**: A funkció csoport rövid leírása

Mint az előzőleg már említve lett, minden funkció csoporthoz definiálhatunk „timeout” metódusokat, ill. alternatív metódusokat. Ezt a **scheduledMethod** elem gyerek elemeivel tehetjük meg az alábbiak szerint:

- **timeoutMethod**: Ennek az elemnek üres a törzse, egy **name** attribútuma van amelyet „timeout” esemény bekövetkeztekor hívunk meg.
- **humanCalledMethod**: Ez az elem egy olyan függvényre hivatkozik, amelyet a felhasználó hívhat meg a Web interfészen keresztül és az őt definiáló ütemezett metódus helyett hívódik meg. Meghívása csak akkor lehetséges, ha az ágens egy felhasználó irányítja. Attribútumai az alábbiak:
  - **name**: a hivatkozott metódus neve

- `displayName`: A képernyőn megjelenítendő név
- `shortDescription`: egy rövid leírás, amely a képernyőn tooltip-ként jelenik meg.