



Útmutató MASS/PET modell család készítéséhez

PET modell írók kézikönyve és tutotál

Készítette Korompai Attila, Kozma Vilmos és Marton David Ivanyi

Az ELTE-IKKK részére



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

2007 Október

Tartalomjegyzék

1	Bevezető	3
1.1	Ágens-alapú modellezés	3
1.2	MASS	3
1.3	Célok	3
1.4	Előfeltételek	4
2	A PET modell család struktúrája	5
2.1	Ágensek	5
2.1.1	Ágensek tulajdonságai	5
2.1.2	Visitor osztályok	6
2.2	Vizualizációk	6
2.2.1	Applet típusú vizualizációk	6
2.2.1.1	Az objektum típusú vizualizációs applet megvalósítása	7
2.2.1.2	Kép típusú vizualizáció megvalósítása applet-en keresztül	7
2.2.2	Kép típusú vizualizáció	8
2.2.3	Link típusú vizualizáció	8
2.3	A modell család leíró xml fájl	8
3	Tutoriál: A vadász játék elkészítése	9
3.1	A vadász játék bemutatása	9
3.2	Szoftver feltételek	9
3.3	A modell család elkészítése	9
3.3.1	A leíró vázának elkészítése	9
3.3.2	A Space ágens létrehozása	9
3.3.3	A CreateSpaceVisitor osztály létrehozása	11
3.3.4	A tér ágens definiálása a leíróban	12
3.3.5	Creating the Runner agent	13
3.3.6	A Runner ágens deklarációja a leíróban	16
3.3.7	Creating a visualization applet	17
3.3.8	Detektor metódusok hozzáadása az ágensekhez	18
3.3.9	A detektorok beírása a leíró fájlba	19
3.3.10	Akciók létrehozása	21
3.4	A modell család telepítése	23
4	Befejezés	24
5	Melléklet: A modell család leíró xml	25
5.1	A leíró struktúrája	25
5.1.1	Az agent elem	25
5.1.1.1	A field elem	26
5.1.1.2	A visualization elem	26

1 Bevezető

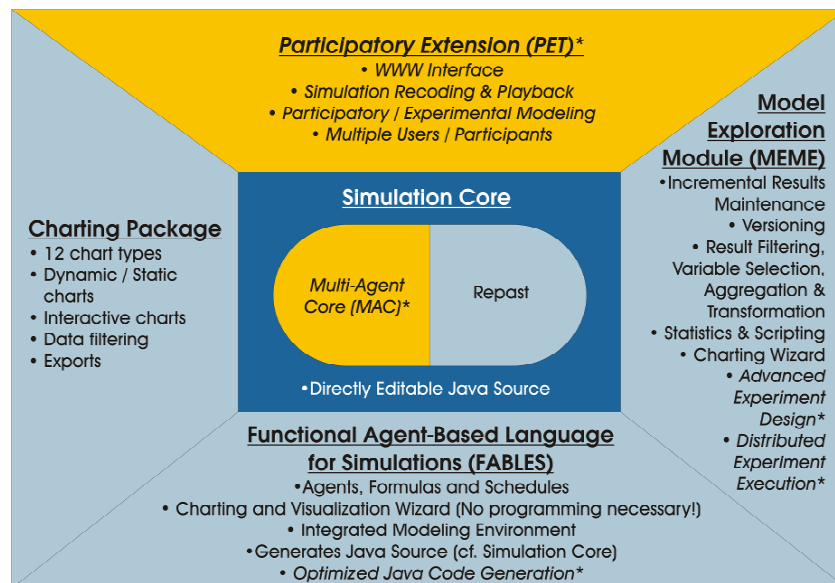
1.1 Ágens-alapú modellezés

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.



Ábra 1 - Multi-Agent Simulation Suite

1.3 Célok

A dokumentum célja, hogy útmutatást adjon a modell családok készítéséhez és telepítéséhez a (Participatory Extension) PET rendszerben. A PET részét képezi a Multi-Agent Simulation Suite (MASS) szoftvernek és alkalmas részvételi szimulációk

futtatására. A szimulációkat futtató motor alapját a Repast és a saját fejlesztésű Multi-Agent Core (MAC) egyaránt képezik. A dokumentum végigmegegy a PET modell családok készítését érintő kérdéseken, tárgyalja a felmerülő kérdéseket kiemelve a lehetőségeket.

1.4 Előfeltételek

A dokumentum olvasójától az alábbiakat várjuk el:

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

2 A PET modell család struktúrája

2.1 Ágensek

Egy PET szimuláció központi részét az ágensek képezik. Minden ágens osztálynak az `ai.aitia.mass.base.Agent` osztályból kell öröklődnie. Egy ágensnek három logikailag elkülöníthető része van:

- **Tulajdonságok (Property) és azok író / olvasó metódusai:** egy ágens aktuális állapotát annak tulajdonságai írják le. Megkülönböztetünk alap és saját tulajdonságokat.
- **Akciók:** Olyan metódusok, melyek visszatérési értéke `void`, és nem kapnak paramétert. Ezeket a metódusokat hívhatja az ütemező, vagy az ágens irányító felhasználók, a Web felületen keresztül.
- **Detektorok:** olyan metódusok, amik visszatérési értéke tetszőleges lehet, és nem kapnak paramétert. Ezen metódusok célja, hogy az ágens információt szolgáltatson magáról és környezetéről. Ezt az információt gyakran a vizualizációk használják fel.

2.1.1 Ágensek tulajdonságai

Egy ágensnek kétféle tulajdonsága lehet:

- **Alap tulajdonságok:** A PET-ben minden ágens az `ai.aitia.mass.base.Agent` osztályból öröklődik. Ez azt eredményezi, hogy minden ágens ugyanazzal az alap tulajdonság halmazzal rendelkezik, melyet az alábbi lista részletez. Megjegyezzük, hogy alap esetben, a modellírónak nem kell ezekkel a tulajdonságokkal foglalkoznia.
 - **id:** az ágens egyedi azonosítója.
 - **type:** az ágens típusa.
 - **controllable:** logikai változó, amely mutatja, hogy az ágens irányítható-e vagy sem.
 - **visible:** logikai változó, amely mutatja, hogy az ágens látható-e vagy sem.
 - **visualization:** hivatkozás a hozzárendelt vizualizációra.
 - **detectorsVisible:** logikai változó, amely mutatja, hogy az ágens detektorai láthatóak-e a Web felületen vagy sem.
 - **controlled:** logikai változó, amely mutatja, hogy az ágens éppen irányítás alatt áll-e vagy sem.
 - **defaultAction:** referencia az alapértelmezett meghívható akcióra.
 - **timeoutAction:** referencia az alapértelmezett meghívható timeout akcióra. A timeout metódus hívásokat az ágens timeout eseményei váltják ki. A PET-et be lehet úgy állítani, hogy ki-timeout-oljon egy olyan ágenset, amelynek irányítója egy bizonyos ideig nem mutat aktivitást. A timeout beállításokról bővebb információt a PET adminisztrátorok könyvéből szerezhet.
 - **actions:** Egy lista, amely tartalmazza az ágens összes akció metódusát. Az akció metódusok `void`-al térnek vissza és nem lehet paraméterük. Ezeket a metódusokat hívhatja az ütemező, vagy az ágens irányító felhasználók, a Web felületen keresztül.
 - **detectors:** Egy lista, amely tartalmazza az ágens összes detektor metódusát. visszatérési értéke tetszőleges lehet, és nem kapnak paramétert. A detektorok célja, hogy az ágens információt szolgáltatson magáról és környezetéről.

- o `simulation`: A szimulációt leíró objektum.
- **Saját tulajdonságok**: A PET-be minden tulajdonság, amelyet a modellíró ad egy ágenshez saját tulajdonságnak tekintünk. A saját tulajdonságok elérhetőségét a modell család leíró fájlban definiálhatjuk. (Lásd: 2.3. fejezet). A setter metódus elhagyásával egy tulajdonság automatikusan csak olvashatóvá válik.

Hogy különbséget tegyünk az alap tulajdonságok és a saját tulajdonságok között, minden alaptulajdonság, és azok getter/setter metódusai `_Pet`-re végződnek. Az ágens egy tulajdonsága akkor lesz ismert a PET számára, ha azt hivatkozunk a modell család leíróban. Megkülönböztetünk négy speciális alap tulajdonságot (`visible`, `controllable`, `visualization` és `detectorsVisible`), melyeket „virtuálisan” hivatkozunk a leíróban. A hivatkozott tulajdonságokat a rendszer tudja kezelni és meg is jeleníti a Web-es interfészen. Minden hivatkozott tulajdonság az alábbi attribútumokkal bír:

- **Elérhetőség**: a tulajdonság elérését lehet vele szabályozni. Lehetséges értékei:
 - o `no`: a tulajdonság nem érhető el a Web interfészen.
 - o `r`: a tulajdonság csak olvasható.
 - o `rw`: a tulajdonság írható / olvasható.
- **Alapértelmezett érték**: a tulajdonság alapértelmezett értéke.
- **Megjelenítési név**: a Web interfészen megjelenített név.
- **Class**: ha a tulajdonság típusa nem primitív típus vagy `string`, akkor ezzel megadhatjuk a típusát.
- **Rövid leírás**: a tulajdonság egy rövid leírása, ami tool-tip formájában jelenik meg a Web interfészen.
- **Konstans lista**: lehetőség van arra, hogy a tulajdonság számára definiáljunk, egy nevesített értékekből álló listát. Ha ezt megtesszük, akkor a Web interfészen a felsorolt értékek nevei jelennek meg egy lenyíló listában.
- **Inicializált**: Logikai érték, amely mutatja, hogy a tulajdonság értéke változtatható-e a szimuláció futása közben, amikor a szimuláció éppen felfüggesztett állapotban van.
- **Láthatóság**: Logikai érték, amely mutatja, hogy a tulajdonság látható-e a felhasználó számára.

2.1.2 A Visitor osztályok

Egy ágens az `Agent` osztály `IAgent[] getAgents_Pet()` függvényén keresztül érheti el a többi ágens, vagy az `IAgentVisitor` interfész `visit()` metódusával. Egyes fejlesztők szerint az utóbbi megoldás tisztább kódot eredményez.

```
public interface IAgentVisitor
{
    public void visit(IAgent agent);
}
```

Könnyen hajthatunk végre műveleteket (meglátogatjuk az ágens) az összes ágensen, ha meghívjuk az `Agent` osztály `visit_Pet(IAgentVisitor visitor)` metódusát. Paraméterül az `IAgentVisitor` interfész egy implementációját kell adnunk.

2.2 Vizualizációk

A PET támogatja a modellírók számára a vizualizációk készítését. A következő három fejezet a lehetőségeket tárgyalja.

2.2.1 Applet típusú vizualizációk

Az applet típusú vizualizáció egy applet-ben jelenik meg a böngészőben. A működési elve az alábbi: a szimuláció futása közben, egy a böngészőbe betöltött applet periodikusan hívogat egy detektor servlet-et, vagy úgynevezett image-producer servlet-et, ami egy

kiszerializált objektummal vagy egy kép bájttjaival tér vissza. A visszaadott objektum, vagy kép tartalmazza mindazt az információt, ami a vizualizáció kirajzolásához szükséges. Mint azt az előbbi leírás is sugallja, kétféle vizualizációs applet létezik:

- **Objektum típusú:** Ebben az esetben az applet egy detektor servlet-et hív meg, ami egy a vizualizáció megrajzolásához szükséges objektummal tér vissza. A servlet az objektumot úgy állítja elő, hogy meghívja az ágens valamelyik detektor metódusát. A meghívandó detektor függvény nevét a http kérés tartalmazza. A visszaadott objektumnak tartalmaznia kell mindazt az információt, ami szükséges ahhoz, hogy az applet ki tudja rajzolni a vizualizáció aktuális állapotát, az ágens szemszögéből. Fontos megjegyezni, hogy a visszaadott objektumnak implementálni kell a `java.io.Serializable` interfészt.
- **Kép típusú:** Ebben az esetben az applet az úgynevezett image-producer servlet-et hívja meg, ami egy képfájl bájttjaival tér vissza. Az applet ezt a képet rajzolja ki a felületére vizualizáció gyanánt.

2.2.1.1 Az objektum típusú vizualizációs applet megvalósítása

Ehhez a modellírónak létre kell hozni egy új Java osztályt, amit az `ai.aitia.mass.web.applet.DetectorApplet` osztályból kell örököltetni. Ez az osztály egy applet, ami minden megtesz, amit csak lehet, hogy a modellírónak minél kevesebbet kelljen kódolnia. Az egyetlen feladata a modellírónak tehát a `paintObject(Graphics g, Object obj)` metódus megírása, melyben az `obj` paraméter által tartalmazott rajzolási információkat kell rárajzolni a szintén paraméterben kapott `g` rajzfelületre. Az `init()` metódus felüldefiniálása hasznos lehet az inicializációs jellegű feladatok elvégzésére.

2.2.1.2 Kép típusú vizualizáció megvalósítása applet-en keresztül

A kép típusú vizualizáció megvalósításához az előzővel ellentétben nem applet-et, hanem egy úgynevezett image-producer osztályt kell írni. Az image-producer osztály egy képet generál, amit az applet megjelenít a képernyőn. A modell család leíróban kell megadni mindazon információkat, amik szükségesek az applet működéséhez. Egy image-producer osztályt nem nehéz megírni, mert csak annyit kell tennünk, hogy írunk egy osztályt, ami implementálja az `ai.aitia.mass.visu.ImageProducer` interfészt. Az image-producer objektum élettartama a szimuláció élettartamával egyezik meg. Az interfész definíciója az alábbi:

```
public interface ImageProducer {

    public void init(Simulation sim, IAgent agent);

    public void drawImage(Graphics2D g2d);

    public void stop();

    public int getImageWidth();

    public int getImageHeight();

}
```

Nézzük meg a függvényeket egyenként:

- `init(Simulation sim, IAgent agent)`: ez a függvény inicializálja az image-producer objektumot. Minden inicializáló jellegű objektumot ide kell tenni.
- `drawImage(Graphics2D g2d)`: ebben a metódusban kell megrajzolni a képet az `obj` paraméter alapján.
- `stop()`: Ez a metódus akkor hívódik meg, amikor a szimuláció megáll.

- `getImageWidth()`: visszaadja a legenerált kép szélességét.
- `getImageHeight()`: visszaadja a legenerált kép magasságát.

2.2.2 Kép típusú vizualizáció

Ezt a vizualizáció fajtát akkor használjuk, amikor a modellíró állókép jellegű vizualizációt akar készíteni. Ilyenkor a böngészőben csak egy állókép fog megjelenni, ami csak az oldal újratöltésekor frissül. Az implementáció az előző fejezetben tárgyalt módszerrel történik. Az egyetlen különbség az előző fejezethez képest az, hogy a modell család leíróban másképp van hivatkozva az image-producer osztály.

2.2.3 Link típusú vizualizáció

Amennyiben a modellíró ezt a vizualizáció típust válassza, szinte bármit képes lesz megvalósítani a Web technológiák (Flash, Ajax, Applet) nyújtotta lehetőségek korlátain belül. Egy link típusú vizualizáció a böngészőben linkként jelenik meg, és a tényleges vizualizáció a link hivatkozási helye. Erre kattintva bejöhethet egy Flash, egy applet vagy akár egy egész Ajax alkalmazás, lényegében véve bármi, ami képes meghívni az ágens detektor függvényeit a detektor servlet-en keresztül.

2.3 A modell család leíró xml fájl

Egy jól 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 fájl tartalmaz minden szükséges információt (pl.: ágensekről, azok tulajdonságairól, vizualizáciokról stb.) amire a PET-nek szüksége lehet a modell családból készített modellek konfigurálása és a szimulációk futtatása során.

3 Tutoriál: A vadász játék elkészítése

3.1 A vadász játék bemutatása

Ebben a szimulációban kétféle ágens szerepel. Néhány ágens vadász, a többi áldozat. Ezek egy kétdimenziós, térben mozognak. Ahogy a nevük is sugallja, a vadászok megpróbálják elkapni az áldozatokat, miközben azok megpróbálnak kitérni előlük. Amikor egy vadász elkapja áldozatát, az áldozat eltűnik, és a tér egy véletlenszerű pontján jelenik meg újra.

3.2 Szoftver feltételek

- Telepített JDK 5.0 vagy újabb
- Egy telepített és működő PET alkalmazás

3.3 A modell család elkészítése

3.3.1 A leíró vázának elkészítése

Első lépésként a modell család leíró vázát hozzuk létre. Hozzon létre egy új fájlt, melynek adja a `hunter.xml` nevet. Másolja bele az alábbi xml kódot:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE model-family SYSTEM "mac_model_family.dtd">

<model-family>

  <family-name>Hunter</family-name>
  <simulation-class>ai.aitia.mass.base.SimulatedRealTimeSimulation</simulation-class>
  <image>../images/hunter/tiger.jpg</image>

  <description>Hunter simulation</description>

  <agents>
  </agents>
</model-family>
```

A fenti xml részben megadtuk, hogy a modell család neve **Hunter** és a modell családhoz rendelt kép neve `tiger.jpg`, amely a PET alkalmazás `WEB-INF/./images/hunter` könyvtárában található. Másoljuk be a `tiger.jpg` fájlt a helyére. A modell család rövid leírása **Hunter simulation**. Ez a szöveg egy tool-tip-ként jelenik meg a Web-interfészen. Az `agents` elem még üres, mert még nem hoztunk létre ágens.

Kétféle ágens típusunk van:

- A tér, amit a `space` osztály definiál a szimulációban. Ebből a típusból csak egy példányra lesz szükségünk. Ez tartalmazza a másik ágens típus a **Runner** összes előfordulásának koordinátáit.
- A `Runner` osztály írja le a vadászokat és az áldozatokat.

3.3.2 A Space ágens létrehozása

Hasznos, ha van egy külön ágens, ami a teret képviseli. Habár a vadászok és áldozatok tartalmazznak minden, a lépéseik véghezviteléhez szükséges adatot, a térágens egy

megfelelő hely az olyan globális változók tárolására, mint amilyenek a tér dimenziói, vagy az ágensek maximális száma.

Jó hely arra is, hogy minden ágensről információt gyűjtsünk (például átfogó nézet létrehozásához).

A másik előnye, hogy a tér ágens, mint a Hunter és Prey ágensek tagváltozóját lehet definiálni. A PET ösztönzi ezt a fajta hierarchikus függőséget (Hunter & Prey nem példányosítható, ha nincs hozzájuk társítható tér), ami jó módszer annak biztosítására, hogy egy bizonyos ágens létezen bármiféle kód megírása nélkül.

```
public class Space extends Agent
{
    public static final int EMPTY = 0;
    public static final int HUNTER = 1;
    public static final int PREY = 2;

    private int height;
    private int width;
    private int[][] space;

    @Override
    public void init_Pet() {
        space = new int[width][height];
    }

    @Override
    public void onStep_Pet() throws StepFailedException {
        space = new int[width][height];
        CreateSpaceVisitor visitor = new CreateSpaceVisitor(space);
        visit_Pet(visitor);
    }

    @Override
    public void onTimeout_Pet() {
    }

    @Override
    public void onTake_Pet() {
    }

    @Override
    public void onRelease_Pet() {
    }

    @Override
    public void onRemove_Pet() {
    }
}
```

```

}

public int getWidth() {
    return width;
}

public void setWidth(int width) {
    this.width = width;
}

public int getHeight() {
    return height;
}

public void setHeight(int height) {
    this.height = height;
}
}

```

Az ágens osztályban az absztrakt metódusok a következők: `init_Pet()`, `onStep_Pet()`, `onTimeout_Pet()`, `onTake_Pet()`, `onRelease_Pet()`, `onRemove_Pet()`. Egy érvényes ágens osztálynak mindet implementálnia kell. Jelen esetben csak az `init_Pet()` és `onStep_Pet()` függvényekbe fogunk kódot írni. Amint az látható, az `onStep_Pet()` metódus használja a `CreateSpaceVisitor` osztály egy példányát. Ezt az osztályt a következő fejezet mutatja be.

3.3.3 A CreateSpaceVisitor osztály létrehozása

A `CreateSpaceVisitor` osztály implementálja az `IAgentVisitor` interfészt. Ezt a tér ágens használja arra, hogy koordináta információkat nyerjen ki a **Runner** ágensekből. Emlékeztetőül elolvashatja a 2.1.2. fejezetet.

```

public class CreateSpaceVisitor implements IAgentVisitor
{
    int[][] space;

    public CreateSpaceVisitor(int[][] space)
    {
        this.space = space;
    }

    public void visit(IAgent agent) {
        if (agent instanceof Runner) {
            Runner runner = (Runner)agent;

            boolean hunter = runner.isType();
            int x = runner.getX();
            int y = runner.getY();

```


☞ Add new agent - Space agent

Property	Class	Value	Visibility	Initialized
Controllable		<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
Detectors visible		<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
Space height	java.lang.Integer	300	<input type="radio"/> true <input checked="" type="radio"/> false	<input checked="" type="radio"/> true <input type="radio"/> false
Space width	java.lang.Integer	300	<input type="radio"/> true <input checked="" type="radio"/> false	<input checked="" type="radio"/> true <input type="radio"/> false
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
Visualization	java.lang.String	None	<input type="radio"/> true <input checked="" type="radio"/> false	<input checked="" type="radio"/> true <input type="radio"/> false

Alapértelmezett értékek

Deklarált tulajdonságok Add 1 piece(s)... Add

2. ábra A **Space** ágens deklarált tulajdonságai

3.3.5 A Runner ágens létrehozása

Mivel a vadász és az áldozat ágenseknek alapvetően ugyanaz a feladatuk (meghatározzák, melyik a legközelebbi ellenség és elmozdulnak), csak egy osztály létrehozása szükséges. A `type` nevű logikai változó dönti el, hogy egy adott ágens vadász-e vagy áldozat.

```
public class Runner extends Agent
{
    private boolean type; // true - hunter ; false - predator
    private int x;
    private int y;
    private int skip;
    private Space space;

    @Override
    public void init_Pet() {
        newRandomPlace();
    }

    @Override
    public void onStep_Pet() throws StepFailedException {
        if (getTick_Pet() % (skip + 1) != 0) return;

        Runner runner = getClosest();

        if (runner != null) {
            int diff = 1;
            if (!type) diff = -1;

            int oldX = x;
            int oldY = y;

            int otherX = runner.getX();
```

```
        int otherY = runner.getY();

        if (type && !runner.type && x == otherX && y == otherY)
            runner.newRandomPlace();

        if (oldX > otherX)
            x = oldX - diff;
        else if (oldX < otherX) x = oldX + diff;

        if (oldY > otherY)
            y = oldY - diff;
        else if (oldY < otherY) y = oldY + diff;

        if (y < 0) y = 0;
        if (y >= space.getHeight()) y = space.getHeight() - 1;

        if (x < 0) x = 0;
        if (x >= space.getWidth()) x = space.getWidth() - 1;

        if (type && !runner.type && x == otherX && y == otherY)
            runner.newRandomPlace();
    }
}

private void newRandomPlace() {
    x = getRandom_Pet().nextInt(space.getWidth());
    y = getRandom_Pet().nextInt(space.getHeight());
}

@Override
public void onTimeout_Pet() {
}

@Override
public void onTake_Pet() {
}

@Override
public void onRelease_Pet() {
}

@Override
```

```
public void onRemove_Pet() {
}

private Runner getClosest() {
    ClosestEnemyVisitor visitor = new ClosestEnemyVisitor(this);
    visit_Pet(visitor);
    return visitor.getClosestEnemy();
}

public boolean isType() {
    return type;
}

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

public int getX() {
    return x;
}

public void setX(int x) {
    this.x = x;
}

public int getY() {
    return y;
}

public void setY(int y) {
    this.y = y;
}

public int getSkip() {
    return skip;
}

public void setSkip(int skip) {
    this.skip = skip;
}

public Space getSpace() {
    return space;
}
```

```

    }

    public void setSpace(Space space) {
        this.space = space;
    }
}

```

Az `Agent` osztályban (amiből mi is örököltetünk) az `init_Pet()`, `onStep_Pet()`, `onTimeout_Pet()`, `onTake_Pet()`, `onRelease_Pet()` és `onRemove_Pet()` metódusok absztraktként vannak deklarálva ezért ezeket meg kell írunk. A rendszer minden körben meghívja az `onStep_Pet()` absztrakt függvényt, amikor senki sem irányítja az ágenszt.

3.3.6 A Runner ágens deklarálása a leíróban

Deklaráljuk a *Runner* ágenszt a leíróban:

```

<agent name="Runner" class="ai.aitia.mass.demos.hunter.Runner">
  <fields>
    <field name="type" access="r">
      <constants>
        <constant name="Hunter" value="true" />
        <constant name="Prey" value="false" />
      </constants>
    </field>
    <field name="x" access="r" />
    <field name="y" access="r" />
    <field name="skip" access="r" />
    <field name="space" access="no" />
  </fields>
  <actions/>
  <detectors/>
  <image>../images/hunter/paw.gif</image>
  <visualizations/>
  <description>Hunter and prey agents</description>
</agent>

```

A fenti xml részletben az ágensnek öt tulajdonságát definiáljuk, és hozzárendelünk egy képet is. Ahogy tettük a tér ágens esetén is, most is másoljuk be a leíróban hivatkozott `paw.gif` fájlt a `WEB-INF/./images/hunter` könyvtárba. Az ágens megjelenítési neve "Hunter and prey agents". A `type` tulajdonsághoz konstans nevesített értékeket rendelünk, hogy a PET ezt a tulajdonságot a Web-en az átlag felhasználó számára olvasható módon jelenítse meg.

Az eredmény az "Edit model" és "Add new agent" lapokon látható, ahogy azt a 3. ábra és 4. ábra is mutatja.

Type	Class	Number	Actions
Space agent	ai.aitia.mass.demos.hunter.Space	0	[Icons]
Hunter and prey agents	ai.aitia.mass.demos.hunter.Runner	0	[Icons]

3. ábra a Runner ágens típus

☞ Add new agent - Hunter and prey agents

Property	Class	Value	Visibility	Initialized
Controllable	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
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
skip	java.lang.Integer	0	<input checked="" type="radio"/> true <input type="radio"/> false	<input checked="" type="radio"/> true <input type="radio"/> false
space	ai.aitia.mass.demos.hunter.Space	279	<input type="radio"/> true <input checked="" type="radio"/> false	<input checked="" type="radio"/> true <input type="radio"/> false
type	java.lang.Boolean	Prey	<input checked="" type="radio"/> true <input type="radio"/> false	<input checked="" type="radio"/> true <input type="radio"/> false
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
Visualization	java.lang.Boolean	None	<input type="radio"/> true <input checked="" type="radio"/> false	<input checked="" type="radio"/> true <input type="radio"/> false
x	java.lang.Integer	0	<input checked="" type="radio"/> true <input type="radio"/> false	<input checked="" type="radio"/> true <input type="radio"/> false
y	java.lang.Integer	0	<input checked="" type="radio"/> true <input type="radio"/> false	<input checked="" type="radio"/> true <input type="radio"/> false

Add piece(s)...

4. ábra A Runner ágens tulajdonságai

3.3.7 A vizualizációs applet létrehozása

Ebben a fejezetben egy objektum típusú vizualizációs applet-et fogunk készíteni. Technikai információkért olvassa el a 2.2.1. fejezetet.

Hozza létre az alábbi `SpaceApplet` nevű osztályt:

```
public class SpaceApplet extends DetectorApplet
{
    private Graphics bufferGraphics;
    private Image offscreen;
    private int width;
    private int height;

    @Override
    public void init() {
        super.init();
        width = getWidth();
        height = getHeight();
        offscreen = createImage(width + 2, height + 2);
        bufferGraphics = offscreen.getGraphics();
    }

    @Override
    public void paintObject(Graphics g, Object obj) {
```

```

Object[] data = (Object[])obj;
int runnerX = ((int[])data[0])[0];
int runnerY = ((int[])data[0])[1];

int[][] map = (int[][])data[1];

bufferGraphics.setColor(Color.BLACK);
bufferGraphics.fillRect(0, 0, width + 2, height + 2);

for (int x = 0; x < map.length; x++) {
    int[] column = map[x];

    for (int y = 0; y < column.length; y++) {
        switch (map[x][y]) {
            case Space.HUNTER:
                bufferGraphics.setColor(Color.RED);
                bufferGraphics.fillRect(x, y, 3, 3);
                break;
            case Space.PREY:
                bufferGraphics.setColor(Color.WHITE);
                bufferGraphics.fillRect(x, y, 3, 3);
                break;
        }
    }
}

if (runnerX > 0 && runnerY > 0) {
    bufferGraphics.setColor(Color.YELLOW);
    bufferGraphics.fillRect(runnerX, runnerY, 3, 3);
}

g.drawImage(offscreen, 0, 0, this);
}
}

```

Ez az osztály a `DetectorApplet` osztályból öröklődik, ezért nekünk csak a rajzolást végző részre kell koncentrálnunk. A rajolás a `paintObject(Graphics g, Object obj)` metódusban történik és az `obj` paraméterben átadott információ alapszik. Az `obj` objektumot az ágensünk detektor metódusa meghívása által kapjuk. A hívást az applet végzi automatikusan, egy detektor servlet-en keresztül. Adjuk hozzá a detektor metódusokat az ágens osztályainkhoz.

3.3.8 Detektor metódusok hozzáadása az ágensekhez

Adja hozzá az alábbi a kód részletet a `space` ágenshez:

```

public Object[] getSpaceAndID() {
    Object[] data = new Object[2];
    data[0] = new int[] { -1, -1 };
    data[1] = space;
}

```

```

return data;
}

```

Adja hozzá az alábbi a kód részletet a `Runner` ágenshez:

```

public Object getSpaceAndID() {
    Object[] data = space.getSpaceAndID();
    data[0] = new int[] { x, y };
    return data;
}

```

Ezek a detektor függvényeink. Ahogy azt korábbi fejezetekben már tárgyaltuk, a detektor függvényeket a vizualizációs applet-ek hívják a hálózaton keresztül, és a visszaadott objektum alapján rajzolják ki a vizualizációt. Figyelje meg, hogy csak egy vizualizációs applet-et írtunk és mindkét ágensnél ezt fogjuk használni. Ez azt eredményezi, hogy a detektor metódusok által visszaadott objektumot úgy kell megtervezni, hogy annak a struktúrája azonos legyen a `Space` és `Runner` ágenseknél. A fenti kód részletekben a visszatérési érték egy kételemű, kétdimenziós tömb. Az első elem a vizualizált ágens koordinátáit tartalmazza. A `Space` ágens esetében ez `{-1,-1}`, ezzel jelezve, hogy a futó vizualizációban nem kell kijelölni egy `Runner` típusú ágens sem. `Runner` objektumok vizualizációja esetén ez az érték a vizualizált ágens koordinátáit jelöli. Ilyenkor az applet kiemeli az éppen vizualizált ágens. A visszatérési érték második eleme egy `Runner` térkép (a `space` objektum). Ez egy kétdimenziós, a tér méreteire beállított nagyságú tömb, amelyben a `Runner` ágensek aktuális koordinátái vannak letárolva.

3.3.9 A detektorok beírása a leíró fájlba

Cserélje ki a sort:

```
<detectors/>
```

Az alábbi xml kód részletre, a `hunter.xml`-nek a `Space` ágens és `Runner` ágens szekcióiban egyaránt:

```

<detectors>
  <detector name="getSpaceAndID" />
</detectors>

```

Ez a változtatás teszi lehetővé a PET számára, hogy az ágenseink detektor függvényeit kezelni tudja.

Cserélje ki a sort:

```
<visualizations/>
```

Az alábbi xml kód részletre a `Space` ágens szekcióban:

```

<visualizations>
  <visualization codebase="../applet" archive="sim.jar"
    code="ai.aitia.mass.demos.hunter.applet.SpaceApplet">
    <type>applet</type>
    <name>HunterSpace</name>
    <display-name>Hunter Space</display-name>
    <detector-name>getSpaceAndID</detector-name>
    <action-frame>false</action-frame>
    <width type="property">width</width>
    <height type="property">length</height>
    <paint-mode>paint_object</paint-mode>
  </visualization>
</visualizations>

```

```

    <refresh-rate>200</refresh-rate>
  </visualization>
</visualizations>

```

A **Runner** ágens szekcióban cserélje ki a következő sort:

```
<visualizations/>
```

Erre:

```

<visualizations>
  <visualization codebase="../../applet" archive="sim.jar"
    code="ai.aitia.mass.demos.hunter.applet.SpaceApplet">
    <type>applet</type>
    <name>RunnerVisualization</name>
    <display-name>Hunter Space for Runner</display-name>
    <detector-name>getSpaceAndID</detector-name>
    <action-frame>true</action-frame>
    <width type="property">width</width>
    <height type="property">length</height>
    <paint-mode>paint_object</paint-mode>
    <refresh-rate>200</refresh-rate>
  </visualization>
</visualizations>

```

A fenti résszel vizualizációt rendelünk az ágenseinkhez. Így, az adminisztrátori oldalon hozzá tudjuk rendelni a vizualizációkat az ágenseinkhez. Az 5. ábra mutatja a vizualizáció kiválasztását az „Add new agent” oldalon.

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

↳ [Edit model \(Don't Be a Prey!\)](#) | [List agents](#) | [Groups](#) | [New agent \(Type: Hunter and prey agents\)](#)

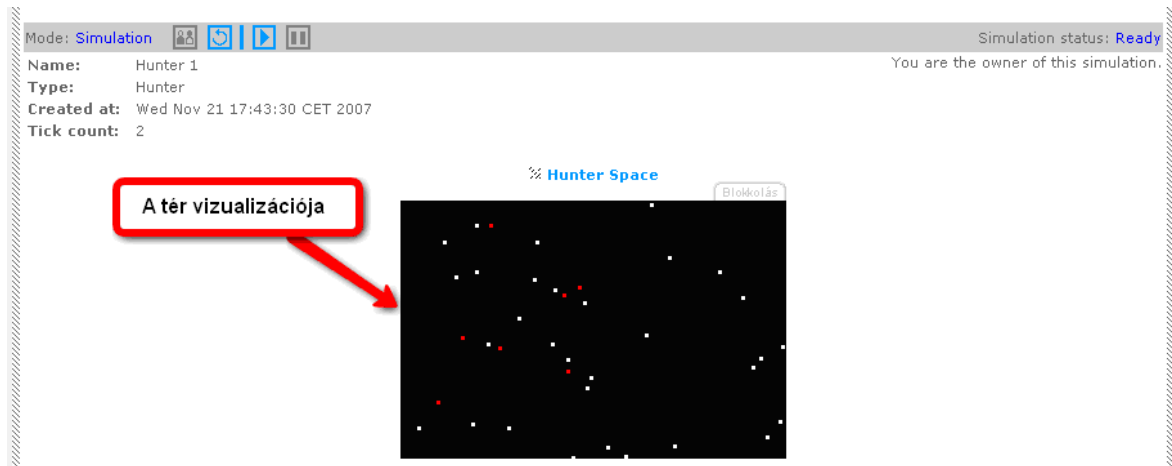
⚡ [Add new agent - Hunter and prey agents](#)

Property	Class	Value	Visibility	Initialized
Controllable	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
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
skip	java.lang.Integer	<input type="text" value="0"/>	<input checked="" type="radio"/> true <input type="radio"/> false	<input checked="" type="radio"/> true <input type="radio"/> false
space	ai.aitia.mass.demos.hunter.applet.SpaceApplet	<input type="text" value="279"/>	<input type="radio"/> true <input checked="" type="radio"/> false	<input checked="" type="radio"/> true <input type="radio"/> false
type		<input type="text" value="Prey"/>	<input checked="" type="radio"/> true <input type="radio"/> false	<input checked="" type="radio"/> true <input type="radio"/> false
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
Visualization	java.lang.String	<input type="text" value="Hunter S..."/>	<input type="radio"/> true <input checked="" type="radio"/> false	<input checked="" type="radio"/> true <input type="radio"/> false
x	java.lang.Integer	<input type="text" value="0"/>	<input type="radio"/> true <input checked="" type="radio"/> false	<input checked="" type="radio"/> true <input type="radio"/> false
y	java.lang.Integer	<input type="text" value="0"/>	<input checked="" type="radio"/> true <input type="radio"/> false	<input checked="" type="radio"/> true <input type="radio"/> false

Add piece(s)...

5. ábra Vizualizáció kiválasztása

A vizualizáció beállítása után hozzunk létre egy új szimulációt és futtassuk azt. Lásd: 6. ábra.



6. ábra A tér ágens vizualizációja

3.3.10 Akciók létrehozása

A PET rendszer lehetővé teszi a felhasználók számára az ágensek „megfogását”, feltéve, hogy azok irányíthatónak lettek beállítva. Ezt a beállítást a rendszer automatikusan lehetővé teszi. Azonban fontos megjegyezni, hogy ha valóban irányítani akarunk egy ágenszt a Web interfészen keresztül, akkor akció metódusokkal kell az ágens osztályunkat kiegészíteni és ezeket a metódusokat hivatkozni is kell a leíróban.

Egészítse ki a `Runner` osztályt az alábbi metódusokkal:

```
private void move(int diffX, int diffY) {
    int newX = x + diffX;
    int newY = y + diffY;

    if (newX < 0) newX = 0;
    if (newX > space.getWidth() - 1) newX = space.getWidth() - 1;
    if (newY < 0) newY = 0;
    if (newY > space.getHeight() - 1) newY = space.getHeight() - 1;

    x = newX;
    y = newY;
}

public void moveLeft() {
    move(-1, 0);
}

public void moveLeftUp() {
    move(-1, -1);
}

public void moveUp() {
    move(0, -1);
}
```

```
public void moveUpRight() {
    move(1, -1);
}

public void moveRight() {
    move(1, 0);
}

public void moveRightDown() {
    move(1, 1);
}

public void moveDown() {
    move(0, 1);
}

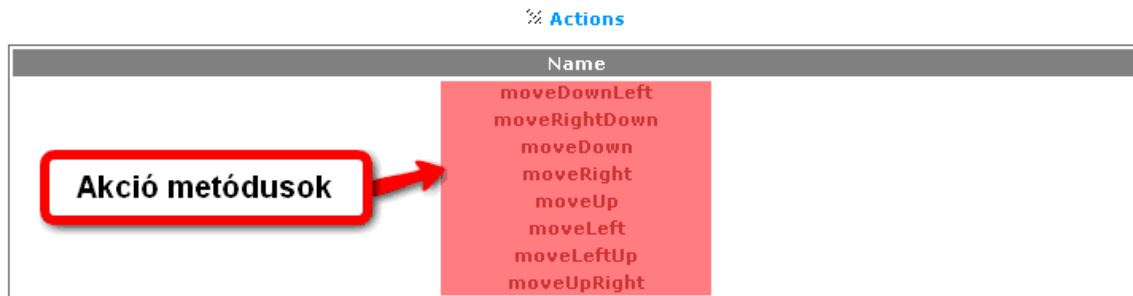
public void moveDownLeft() {
    move(-1, 1);
}
```

Az ágens a `move()` metódus meghívása által mozog. Ezt a metódust a `moveXYZ()` metódusok hívják, ezek az akció metódusok. Minden akció metódus egy egységnyit lépteti az adott ágens valamelyik irányba.

Hivatkozzuk be az akció metódusokat a leíróban:

```
<agent name="Runner" class="ai.aitia.mass.demos.hunter.Runner">
  ...
  <actions>
    <action name="moveLeft" />
    <action name="moveLeftUp" />
    <action name="moveUp" />
    <action name="moveUpRight" />
    <action name="moveRight" />
    <action name="moveRightDown" />
    <action name="moveDown" />
    <action name="moveDownLeft" />
  </actions>
  ...
</agent>
```

Eredményül, az ágens irányítása lapon egy dobozkát láthatunk, amiben fel vannak sorolva az akció metódusok (Lásd: 7. ábra). A kirakott linkekre kattintgatva az ágens lépegetni kezd az adott irányba.

7. ábra A `Runner` ágens irányítása

3.4 A modell család telepítése

- Helyezze el a modell család leíró fájlt (`hunter.xml`) a `WEB-INF/config/descriptors/mac` könyvtárba.
- Fordítsa le az összes most létrehozott Java kódot, és készítse belőle egy Jar fájlt. Ezt helyezze bele a `WEB-INF/lib` könyvtárba.
- Készítsen egy Jar fájlt, amit a böngésző fog letölteni a vizualizációs applet betöltésekor. Ebbe csak a `SpaceApplet`, `DetectorApplet` és `Space` osztályokat tegye bele. Az így elkészült Jar fájlt helyezze bele a PET alkalmazás `WEB-INF/./applet` mappájába.

Végezetül újra kell indítani 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.

4 Befejezés

A PET egy ágens alapú modellező keretrendszer, amely támogatja a részvételi szimulációk fejlesztését és lehetőséget ad, hogy azokat Web-es környezetben futtassuk. Jelen dokumentum bemutatja, hogyan kell natív modell családot írni és telepíteni a PET rendszerbe. Megemlíti a fontosabb fogalmakat, és elmagyarázza a fontosabb működési mechanizmusokat, ezzel is támogatva modellírók munkáját. A dokumentum egy tutorial típusú fejezetet is tartalmaz, amelyben lépésről lépésre bemutatja egy konkrét modell család implementálásának a menetét.

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/mac/](#) könyvtárában kell elhelyezni.
- A fájl tartalmának meg kell felelnie a [WEB-INF/descriptors/mac/mac_model_family.dtd](#) xml definíciós fájlban leírtaknak.

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

5.1 A leíró struktúrája

A gyökér elem neve: [model-family](#). Gyerekei az alábbiak:

- [family-name](#): a modell család neve
- [simulation-class](#): a használt szimuláció típust megvalósító osztály neve. Jelenleg a rendszer három ilyen támogat:
 - [ai.aitia.mass.base.SimulatedRealTimeSimulation](#)
 - [ai.aitia.mass.base.GameSimulation](#)
 - [ai.aitia.mass.base.RealTimeSimulation](#)
- [image](#): a modellhez rendelt kép fájl elérési útja. Az útvonal az alkalmazás [WEB-INF](#) könyvtárához képest relatív.
- [description](#): a modell család leírása
- [agents](#): [agent](#) elemek listáját tartalmazza. Az [agent](#) elemet az 5.1.1 fejezet tárgyalja.

5.1.1 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.
- [actions](#): [action](#) elemek listáját tartalmazza. Egy [action](#) elem az ágens egy metódusára hivatkozik. Az [action](#) elemnek üres a törzse és csak egy [name](#) attribútuma van, ami a metódus nevét tartalmazza. A hivatkozott metódusnak [void](#)-al kell visszatérnie, és nem lehet paramétere. Ezeket a függvényeket a felhasználók hívják a Web-en keresztül.
- [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 és annak környezetéről nyerjenek információt. Ezeknek tipikusan a vizualizációkban van szerepe.
- [image](#): az ágens típushoz rendelt kép elérési útja. Az útvonal relatív az alkalmazás [WEB-INF](#) könyvtárához.

- **visualizations:** `visualization` elemek listáját tartalmazza. Erről bővebben az 5.1.1.2 fejezetben olvashatunk.
- **description:** Az ágens típus neve.

5.1.1.1 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 írunk. 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.

5.1.1.2 A visualization elem

A `visualization` elemnek négy attribútuma van:

- **codebase:** A vizualizációt megvalósító applet Jar fájljának elérési útját tartalmazza a fájlnev nélkül. Csak az applet típusú vizualizációkra vonatkozik. (Lásd. később)
- **archive:** A vizualizációt megvalósító applet Jar fájljának nevét tartalmazza az elérési út nélkül. Csak az applet típusú vizualizációkra vonatkozik. (Lásd. később)
- **code:** Az applet osztály neve. Csak az applet típusú vizualizációkra vonatkozik. (Lásd. később)
- **href:** Az image-producer servlet hivatkozás címe. Csak link típusú vizualizációknál használjuk. Lásd. később.

Az alábbi gyerek elemei vannak:

- **type:** a vizualizáció típusát definiálja. Háromféle vizualizáció típus van:
 - **applet:** A vizualizációt egy applet valósítja meg, amit a modellíró ír meg. Az `applet` osztálynak az `ai.aitia.mass.web.applet.DetectorApplet` osztályból kell öröklődnie.
 - **link:** Link típus esetén a vizualizáció helyett egy link jelenik meg a képernyőn, melyre kattintva bejön a valódi vizualizáció.
 - **image:** Ebben az esetben a vizualizáció egy állókép, amely csak az oldal újra töltésével frissül.
- **name:** a vizualizáció hivatkozási neve
- **display-name:** A vizualizáció képernyőn megjelenített neve
- **detector-name:** A detektor metódus neve. Ez az elem csak akkor szükséges, ha a vizualizáció típusa `applet` és a `paint-mode` elem értéke `paint_object`. (Lásd lejjebb)

- **image-name**: Az image-producer által legenerált képhez rendelt egyedi név. Ez az elem csak akkor szükséges, ha a vizualizáció típusa **applet** és a **paint-mode** elem értéke **paint_image**. (Lásd lejjebb).
- **width**: Az applet szélessége pixelben. Csak akkor szükséges, ha a vizualizáció típusa **applet**.
- **height**: Az applet magassága pixelben. Csak akkor szükséges, ha a vizualizáció típusa **applet**.
- **paint-mode**: Az applet működési módját definiálja. Csak akkor szükséges, ha a vizualizáció típusa **applet**. Két lehetséges értéke van:
 - **paint_object**: ebben az esetben az applet meghívja a hivatkozott detektor metódust, és a képet a detektor metódus által visszaadott objektum alapján frissíti.
 - **paint_image**: Ebben az esetben az applet egy image-producer servlet-et hív meg. Az image-producer servlet egy képet ad vissza melyet a hivatkozott image-producer osztály meghívásával állít elő. Az image-producer osztályokat a modellíró készíti. Ezeknek az osztályoknak implementálnia kell a **ai.aitia.mass.visu.ImageProducer** interfészt. (Lásd. lentebb)
- **refresh-rate**: A vizualizációs applet frissítési intervalluma milliszekundumban. Csak akkor szükséges, ha a vizualizáció típusa **applet**.

imageproducer-class-name: Egy osztálynak a neve, ami implementálja az **ai.aitia.mass.visu.ImageProducer** interfészt.