



Alkalmazási példa

Készítette:

*Baranya Tibor, Iványi Márton Dávid,
Mészáros Róbert, Bocsi Rajmund,
Szabó Attila és Erdélyi Viktor
Az ELTE-IKKK részére*



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

2008 Március

Tartalom

1	Bevezető	3
1.1	Ágens-alapú modellezés	3
1.2	MASS	3
1.3	MEME Tutorial	4
1.4	Iterált fogolydilemma	4
1.5	A tutorial felépítése	5
2	Importálás	6
3	Adatok rendszerezése	10
3.1	Számítások	10
3.1.1	Összetett (aggregative) műveletek	11
3.1.2	Scriptelés	12
3.2	Rendezés	12
3.3	Nézet neve és leírása	13
4	Diagramok létrehozása	14
4.1	Oszlopdiagram készítése	15
4.2	Vonaldiagram készítése	15
4.3	Nagyítás	16
4.4	Ábrák exportálása és formázása	17
4.5	Beépített megjelenítések	17
4.6	A diagramok mentése	18
5	Exportálás	19
6	Megjegyzések	20
7	Referenciák	21
8	Függelék	22
8.1	PrisonersAgent.java	22
8.2	PrisonersModel.java	22
8.3	PrisonersModelBatch.java	24
8.4	PrisonersModelGUI.java	25

1 Bevezető

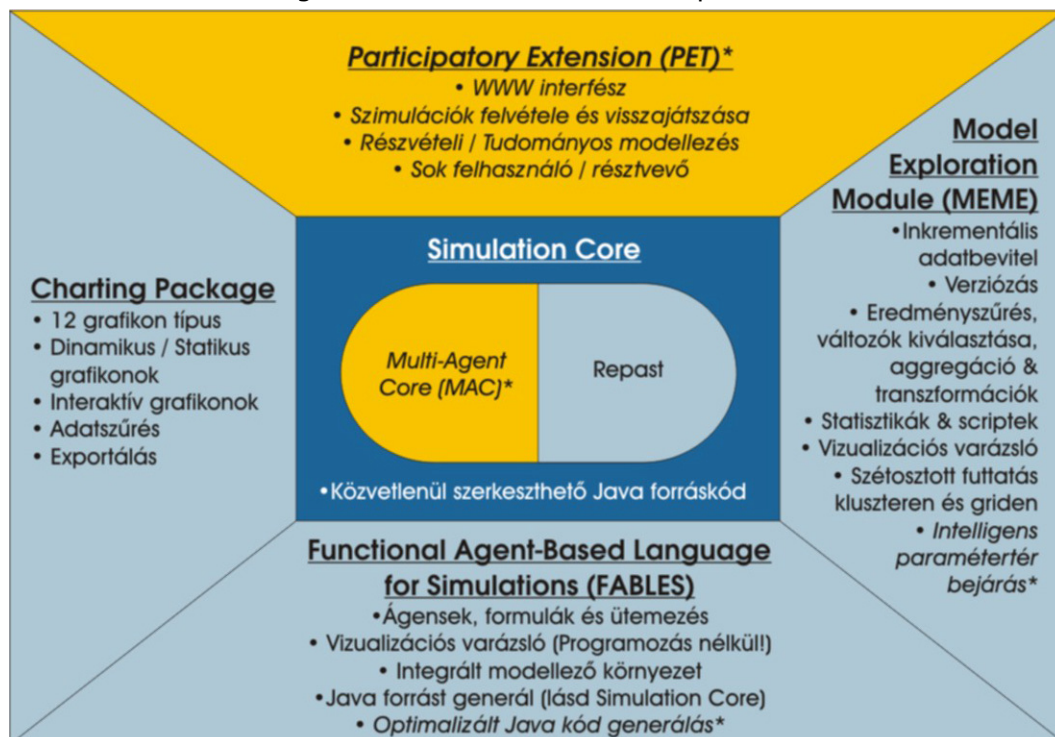
1.1 Ágens-alapú modellezés

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 saját maggal (ez a MAC), illetve képes futni Repast alapokon is. A több szimulációs magon való futtathatóság biztosítja, hogy a modellek magfüggetlenek 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 MEME Tutorial

Ez a dokumentum egy példán keresztül mutatja be a MEME program használatát, azt, hogy hogyan lehet a segítségével szimulációk futási eredményeiből adatokat kinyerni, azokat rendszerezni, majd diagramon megjeleníteni.

A MEME egy olyan eszköz, amely szimulációk sokszori, ismételt, többféleképpen paraméterezett futásainak, illetve az azokból származó adatoknak a kezelésére szolgál. Lehetővé teszi a felhasználó számára, hogy adatbázis(ok)ban tárolja és rendszerezze a nyers adatokat, majd megfelelő táblákat készítsen belőlük, és megjelenítse azokat diagramokon.

A program telepítéséről, illetve az általános tudnivalókról a MEME Felhasználói Kézikönyvben (*MEME_Manual.pdf*) olvashat részletesen.

Ez a tutorial egy Repast modell batch módú futtatása eredményeinek importálásán, az adatok rendszerezésén, majd pedig a rendszerezett adatok vizualizálásán és elemzésén vezeti Önt végig. A program működésének és előnyeinek bemutatására egy példamodellt fogunk használni, az úgynevezett iterált fogolydilemmát.

1.4 Iterált fogolydilemma

A fogolydilemma a nem zérőösszegű játékok egy fajtája. A lényege, hogy két, súlyos bünténnel gyanúsított fogoly közül vallomást tesz-e (defektál) az egyik a másik ellen. Akárcsak a többi játékelméleti problémában, itt is feltételezzük, hogy az egyes játékosok saját „kifizetésüket” tartják szem előtt, tekintet nélkül a másik résztvevő kifizetésére. A játék klasszikus formájában a kooperációval szemben az árulás a domináns, így az egyetlen lehetséges egyensúly az összes fél számára az árulás. Egyszerűbben fogalmazva: attól függetlenül, hogy mit csinál a másik, az egyik játékos mindig nagyobb (nem kisebb) kifizetéshez jut, ha árulást követ el. Mivel az árulás minden helyzetben kifizetődőbb a kooperációnál, a racionális játékosok defektálni fognak.

A fogolydilemmánál a Nash-egyensúly nem vezet mindkét fél számára optimális megoldáshoz, mert ez ebben az esetben azt jelenti, hogy mindkét fogoly vall a másik ellen, még akkor is, ha a kooperációval nagyobb lenne a nyereményük. Bár mindkét fogoly jobban járna, ha kooperálnának, és egyikük sem vallana a másik ellen, mégis mindkettejüknek személyes érdekében áll vallani, akkor is, ha korábban kooperációt ígérenek. Ebben áll a fogolydilemma lényege.

Az iterált fogolydilemma során a játékot ismétlik, minden játékos számára adott a lehetőség, hogy „büntesse” a korábbi árulást, így a kooperáció egyensúlyi megoldássá válhat. A defektálás dominanciáját megtöri a büntetés fenyegetése, ezáltal megteremtve a lehetőséget a kooperációra. Ha játékot végtelen sokszor ismétljük, a kooperáció lehet Nash-egyensúly, noha mindkét játékos árulása még mindig egyensúlyi megoldás marad.

Robert Axelrod *The Evolution of Cooperation* (1984) című könyvében létrehozta a klasszikus fogolydilemma kibővítését, amelyet iterált fogolydilemmának nevezett. Ebben a résztvevőknek újra és újra választaniuk kell, hogy kooperálnak társukkal, vagy elárulják azt, miközben emlékeznek a korábbi eredményekre. Axelrod a világ minden részéről invitálta kollégáit, hogy vegyenek részt számítógépes stratégiáikkal egy iterált fogolydilemma versenyben. A benevezett programok nagyban különböztek többek között algoritmusaik összetettségében, ellenségességükben, megbocsátásra való hajlandóságukban.

Axelrod felfedezte, hogy amikor a játékot hosszú időn keresztül, több, eltérő stratégiával rendelkező ellenfél ellen játsszák, az „önző” stratégiák meglehetősen gyengén szerepelnek, míg az „önzetlen” algoritmusok jobban teljesítenek. Ezt azon természetes szelekciós folyamat szemléltetésére használta, amely során a tisztán önzésen alapuló folyamatokból emberbarát folyamatok alakulhatnak ki.

A legjobb determinista stratégiának az Anatol Rapoport által fejlesztett és benevezett „Tit for Tat” stratégia bizonyult. Ez volt a nevezett programok legegyszerűbbike, összesen egy négy soros BASIC kódból állt, és megnyerte a versenyt. A stratégiája az, hogy az első körben kooperál, és onnantól kezdve azt lépi, amit ellenfele lépett az előző körben

(„ha te úgy, én is úgy”). Valamivel jobb stratégia a „Tit for Tat” kiegészítése „megbocsátással”. Ha az ellenfél defektál, következő lépés valamekkora eséllyel (nagyjából 1-5% valószínűséggel) akkor is kooperálni fog. Ez lehetővé teszi a váltást folyamatos defektálás esetén. A megbocsátás pontos valószínűsége az ellenfelek felállításától függ. Utóbbi stratégia a legjobb abban az esetben, ha bevezetésre kerül a játék során a megbízhatatlan kommunikáció – előfordulhat az, hogy nem az ellenfél valós lépéséről informálódunk.

(Forrás: <http://www.wikipedia.org/>)

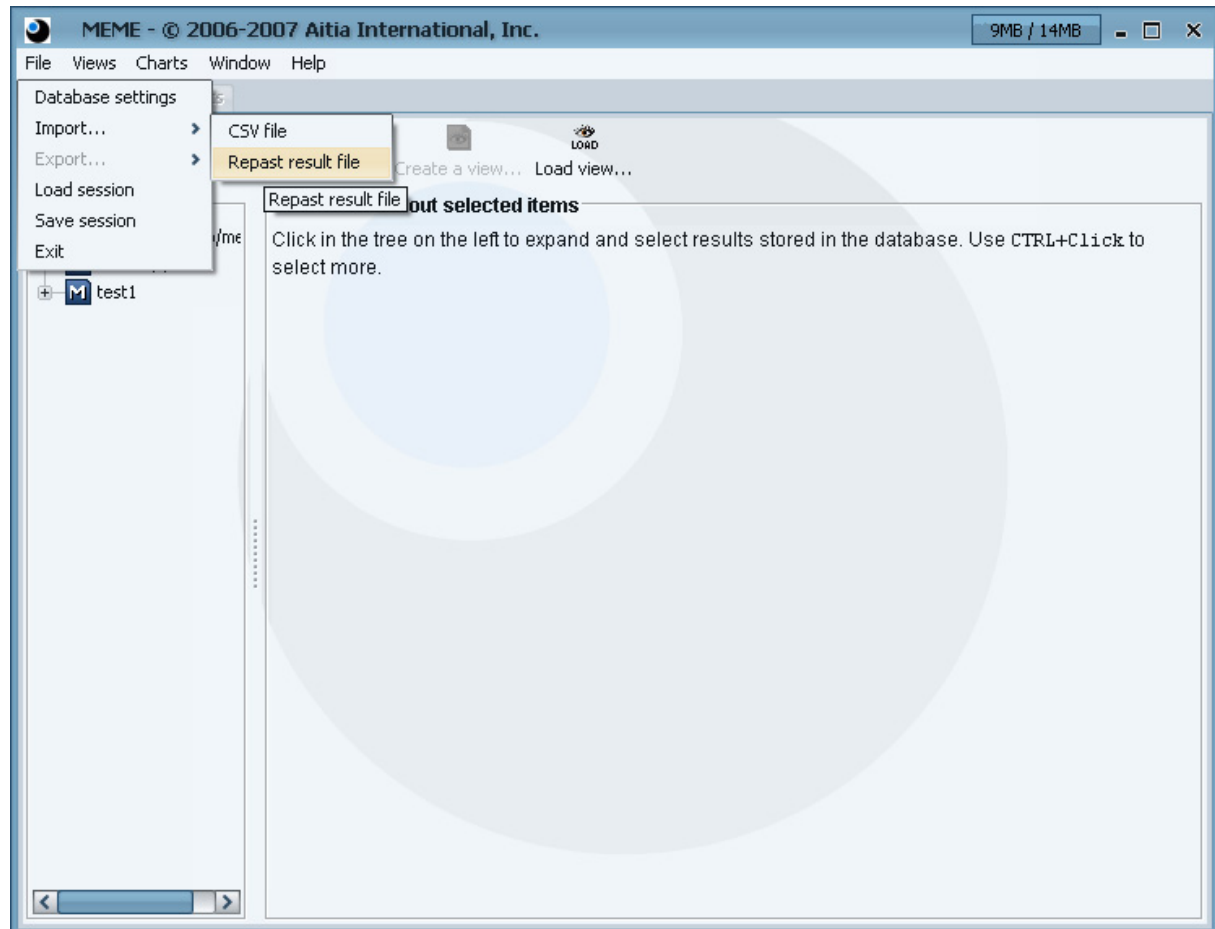
1.5 A tutorial felépítése

Ebből a tutorialból megtanulhatja:

- hogyan importáljon adatokat a MEME adatbázisába egy Repast eredményfájlból. (Mi biztosítunk egy megfelelő eredményfájlt, de Ön is elkészítheti azt a MASS/MEME Parameter Sweep Tool alkalmazás segítségével, részletekért lásd [5]-öt!)
- hogyan rendszerezze az importált adatokat a MEME *View Creation Wizard*-jával;
- hogyan jelenítse meg a rendszerezett adatokat, és
- hogyan exportálja a rendszerezett adatokat az adatbázisból más alkalmazások számára további elemzés céljából.

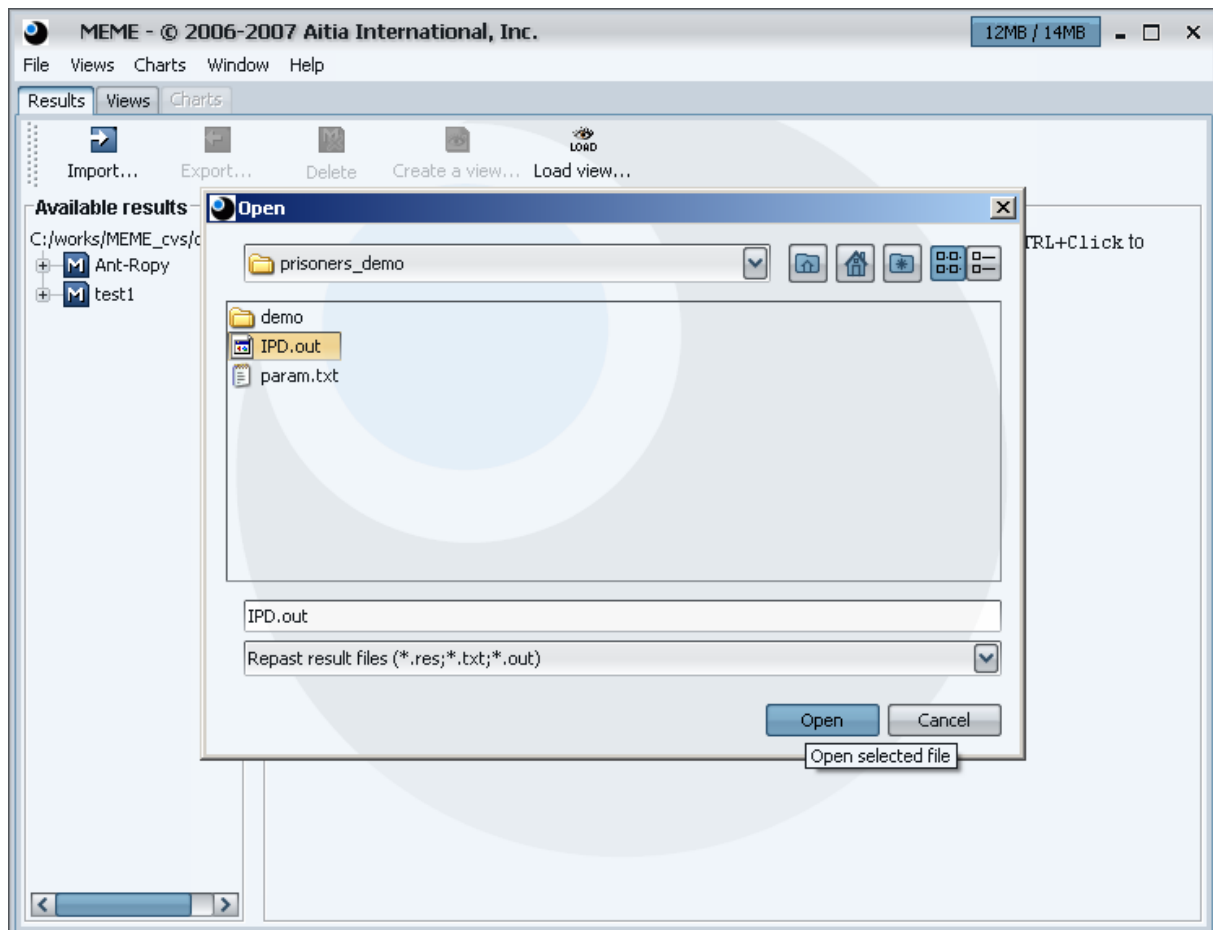
2 Importálás

A MEME indítása után válassza ki a *File/Import/RePast result file* menüpontot (lásd lent!).



2. ábra - Import > Repast result file

Válassza ki a MEME telepítő által tartalmazott Repast eredményfájlt. A fájl helye a C:\Program Files\MASS\MEME\documents\prisoners_demo\IPD.out alapértelmezésben, más telepítési könyvtár esetén a documents\prisoners_demo\IPD.out relatív elérési úton található.



3. ábra - IPD.out helyének megadása

Az *Open* gomb megnyomása után megjelenik a *Repast Import Settings* párbeszédablak (lásd lent!).

Repast Import Settings

Please adjust the input and output parameter sets using the arrow buttons below the table!

Filename: C:\Program Files\MASS\MEME\documents\prisoners_demo\IPD.out ...

Model name: Prisoner's Dilemma Version: 1 (New)

Batch description: IPD batch runs

Number of runs: 375 Max. number of records per run: 1

Input parameters

Name	Status	Value
Both	const.	1
Looser	const.	-3
Neither	const.	0
Payoff1	const.	0
Payoff2	const.	0
StopAt	const.	10000
Stratv1	const.	0
Winner	const.	4
Strat2		<<varies between runs>>
Strat1		<<varies between runs>>
RngSeed		<<varies between runs>>
payoff1		<<varies between runs>>
payoff2		<<varies between runs>>
stratv1		<<varies between runs>>

↓ ↑

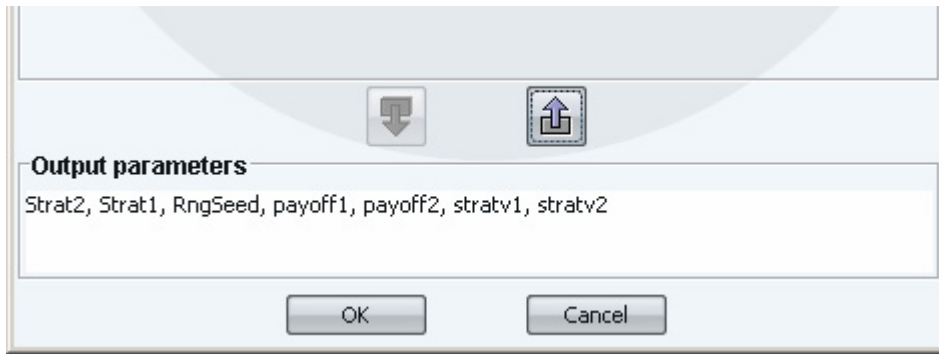
Output parameters

stratv2

OK Cancel

4. ábra - Importálás beállításai

A *Model name* mezőbe írja be, hogy *Prisoner's Dilemma*, amint az a fenti ábrán is látható. A *Batch description* mezőben megadható a modell leírása. Utána kattintson a nyílra az *Output parameters* és *Input parameters* mezők között hatszor, hogy a *Strat1*, *Strat2*, *payoff1*, *payoff2*, *stratv1* és *stratv2* paraméterek bekerüljenek az output értékek közé:



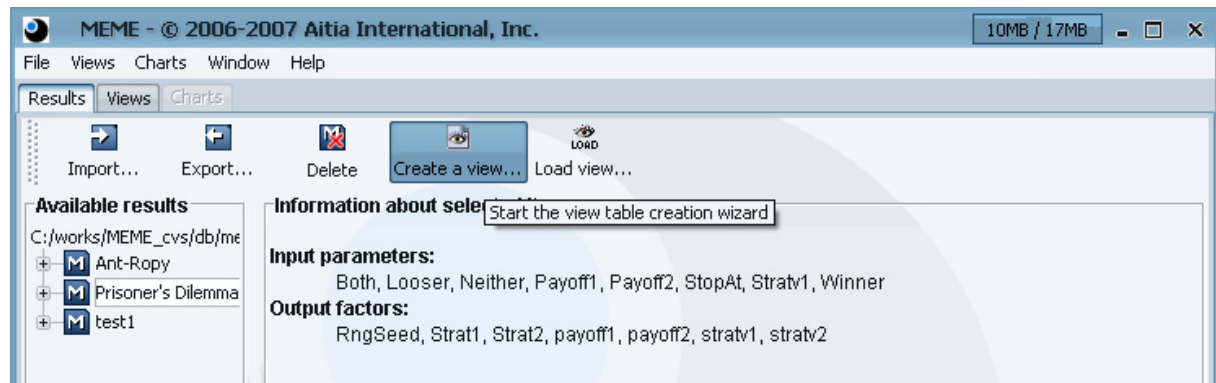
5. ábra - Hozzáadás az output paraméterekhez

Az *OK* gomb megnyomására az eredményfájl tartalma bekerül a MEME adatbázisába.

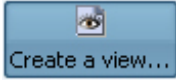
3 Adatok rendszerezése

A beimportált adatokból feldolgozott részhalmazok készíthetők. Ezeket a részhalmazokat a program nézet-táblának hívja. A nézet-táblák megjeleníthetők grafikonokon vagy más alkalmazásoknak kiexportálhatók további elemzésre. Nézet-tábla készítésekor az importált adatok a felhasználó igényei alapján feldolgozásra és rendezésre kerülnek.

Válassza ki a *Prisoner's Dilemma* modellt a *Results* fülön.

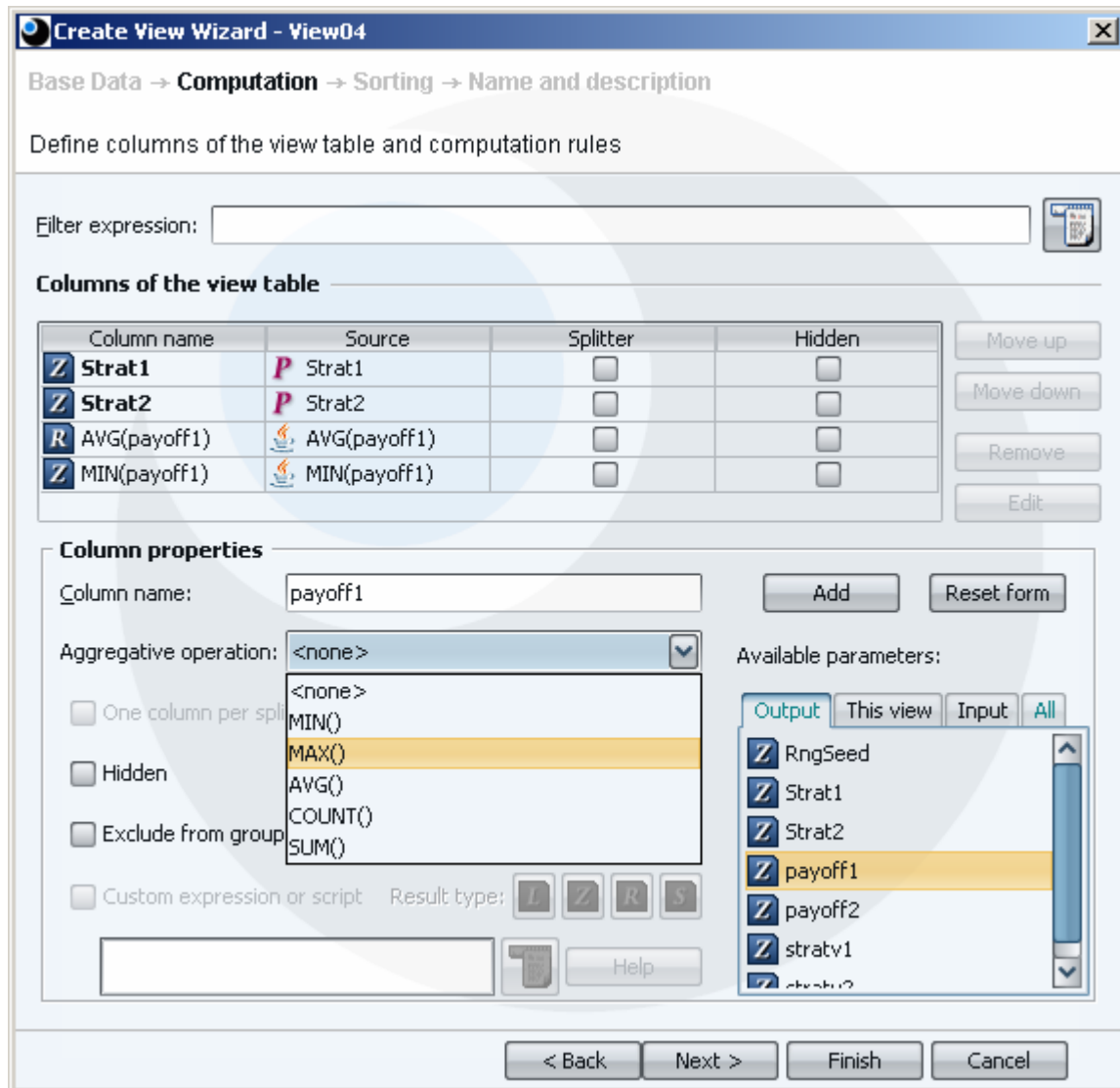


6. ábra - Batch kiválasztása

A *Create View Wizard* indításához nyomja meg a  gombot az ikonsoron, vagy válassza a *Views > Create a view...* menüpontot.

3.1 Számítások

A *Computation* oldalon (lásd lent!) beállíthatóak a nézet-tábla oszlopai. Megjegyzés: az importálás során outputnak választott paraméterek listája az *Available parameters* lista *Output* feliratú fülén található.




7. ábra - Create view wizard

Adja hozzá a táblához a *Strat1* és *Strat2* paramétereket, a sorok kiválasztása után az *Add* gomb megnyomásával (a két paramétert egyidejűleg kijelölve egyszerre is hozzáadhatja azokat a táblához).

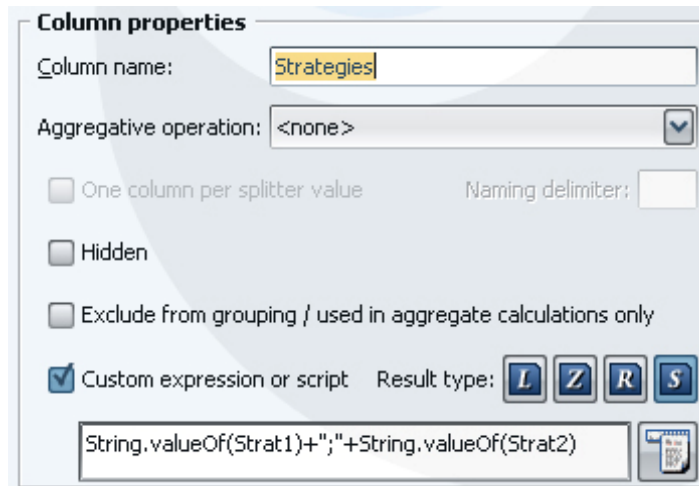
3.1.1 **Összetett (aggregative) műveletek**

Válassza ki a *payoff1* paramétert, állítsa be az *AVG()*-t az *Aggregative operations* mezőben, és nyomja meg az *Add* gombot. Esetünkben a 0. stratégiában véletlenszám-generáláson keresztül kiválasztásra az 1-4. stratégiák egyike. Ennek folyamánként érdemes több alkalommal is lefuttatni egy-egy stratégia párt, máskülönben az eredmények félrevezetőek lennének. Emiatt minden stratégia-pár 10-10 alkalommal került futtatásra, ezért az adatok elemzéséhez ajánlott az átlagolás használata. Ismétlje meg a fent leírtakat a *payoff1* paraméterre a *MIN()* és *MAX()* műveletekkel, majd ezután végezze el ugyanezeket előlről a *payoff2* paraméterrel. A program alapértelmezett módon beállítja az oszlopneveket a felhasznált paraméter nevéből. A mi esetünkben érdemes a generált *payoff1*, *payoff2*,..., *payoff6* neveket lecserélni a *Source* oszlopban (lásd fent!) látható művelet- és paraméternévből képzett elnevezésre.

3.1.2 Scriptelés

Egyedi oszlop készítéséhez válassza ki a *Custom expression or script* opciót a *Column properties* panelen és nyomja meg a  gombot az új oszlop típusának megadásához (most: szöveg). Gépelje be a következőt:

```
String.valueOf(Strat1)+" "+String.valueOf(Strat2)
```

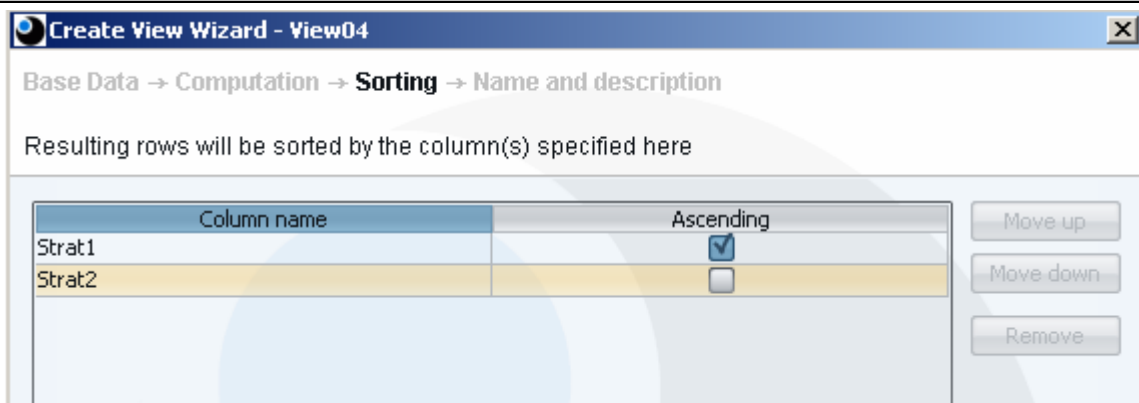


8. ábra - Scriptelés

Ez az egyszerű script létrehozza a *Strat1;Strat2* értékeket, amire később, a grafikonok készítésénél lesz szükség. Adja meg az oszlop nevét (*Strategies*) és kattintson az *Add* gombra.

Az előzőleg már a táblához adott oszlopok szerkesztéséhez válassza ki a megfelelő oszlopot, majd nyomja meg az *Edit* gombot, vagy egyszerűen kattintson duplán az oszlopra. A változtatások bevitele után ne felejtse el a *Modify* gomb segítségével menteni a változtatásokat. A beállítások végeztével nyomja meg a *Next* gombot.

3.2 Rendezés

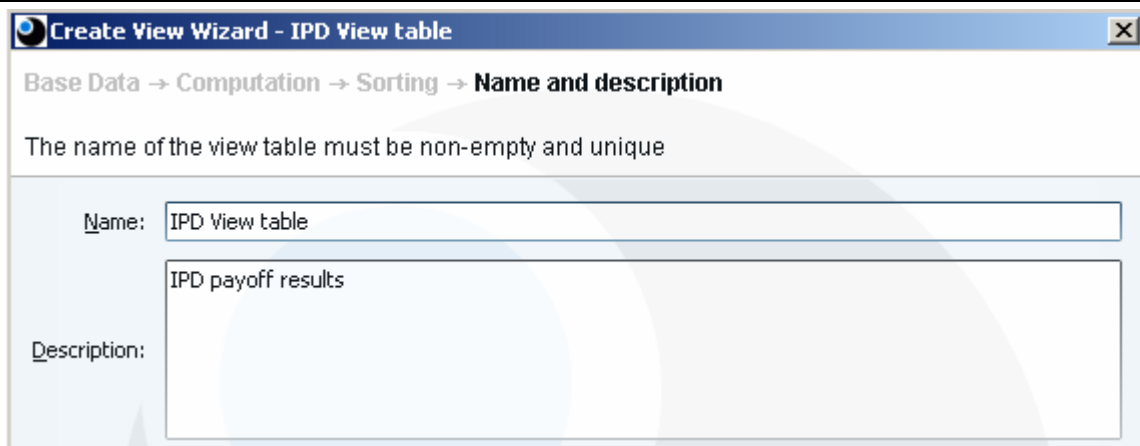


Column name	Ascending
Strat1	<input checked="" type="checkbox"/>
Strat2	<input type="checkbox"/>

9. ábra - Rendezés

Átlátható tábla készítéséhez rendezze a sorokat a *Strat1* és *Strat2* oszlopok szerint, növekvő (*Ascending*) sorrendbe. A művelet végeztével a folytatáshoz kattintson a *Next* gombra.

3.3 Nézet neve és leírása

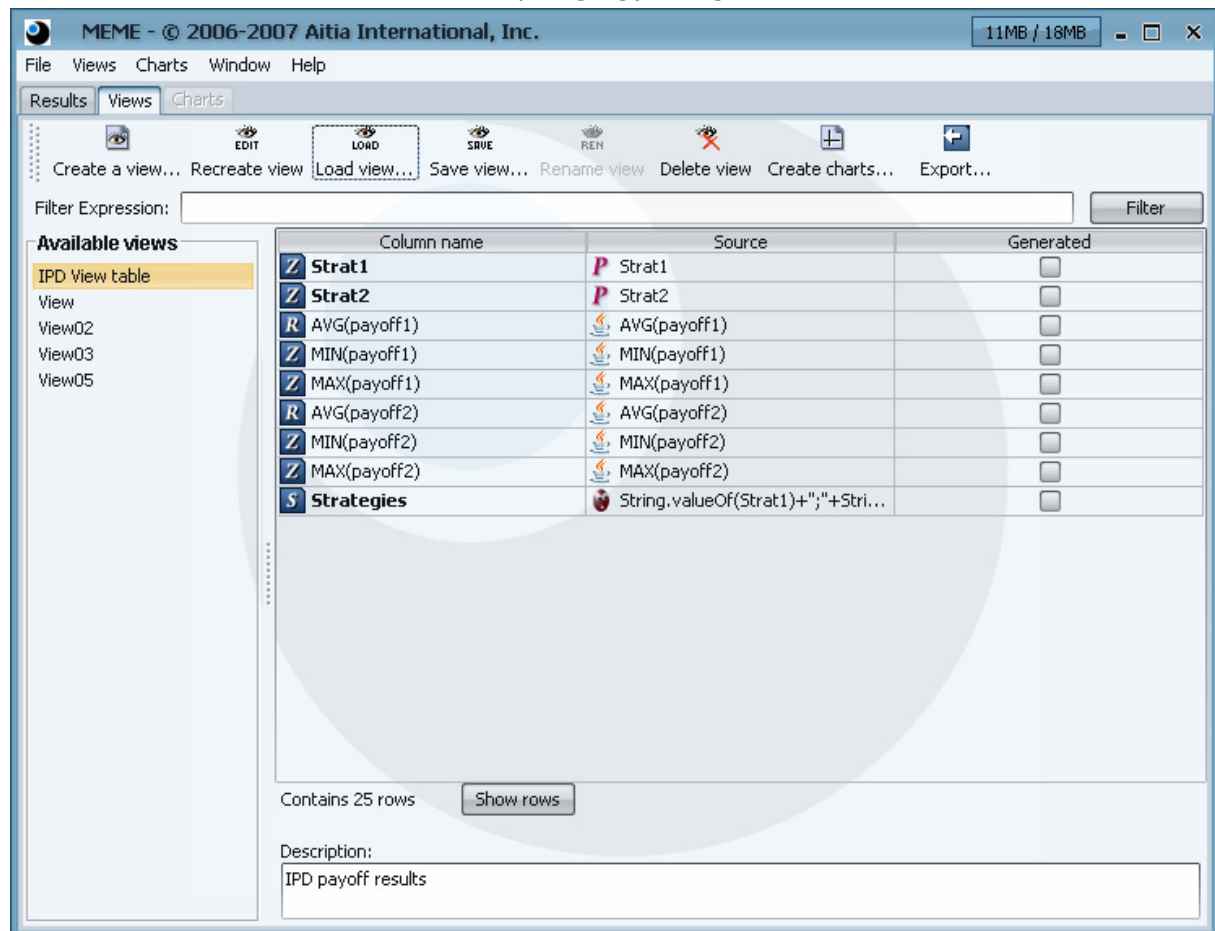


10. ábra - Név és leírás

Adja meg a nézet-tábla nevét (egyedinek kell lennie; a korábban létrehozott, azonos nevű táblát a program felülírja), amely nem lehet hosszabb 64 karakternél, illetve adja meg a tábla leírását (nincs hosszúsági korlát), ha szükségesnek érzi. A *Finish* gomb megnyomásával hozhatja létre a táblát az adatbázisban. Ezek után már készíthet grafikonokat a MEME segítségével, illetve a tábla adatai fájlba CSV exportálhatók a *File > Export* menüben.

4 Diagramok létrehozása

Az imént létrehozott nézet-tábla már szerepel az *Available views* listában. Kilenc oszlopa van, ebből kettő eredeti, a másik hét pedig egyedileg definiált.



11. ábra - Available views

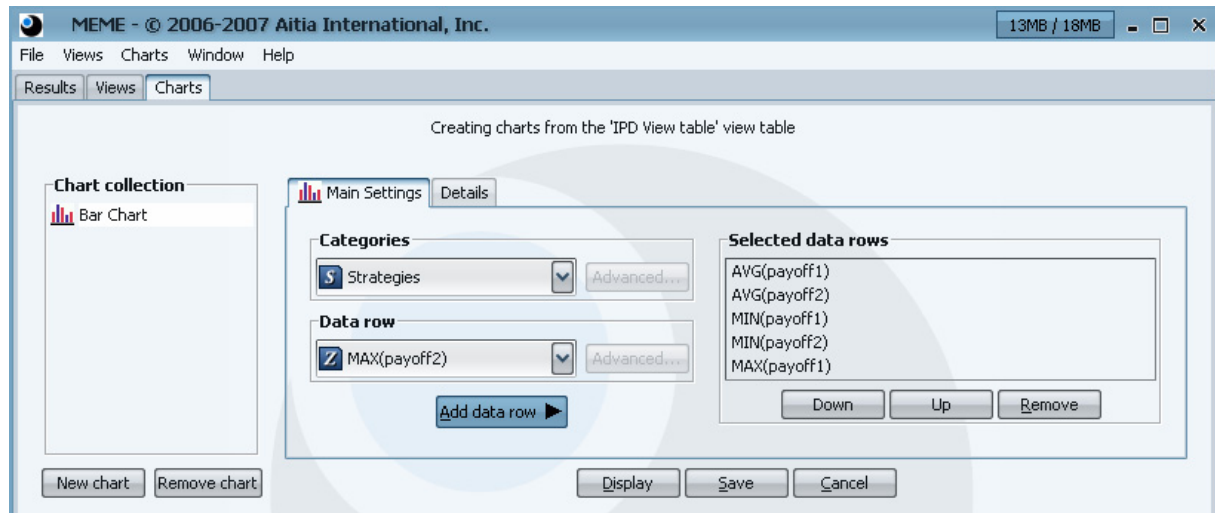
Alapértelmezésként az oszlopnevek és az oszlopok különböző tulajdonságai láthatók, de a *Show rows* gomb megnyomásával a tárolt adatok is megtekinthetők.

#	Strat1	Strat2	AVG(payload1)	MIN(payload1)	MAX(payload1)	AVG(payload2)	MIN(payload2)	MAX(payload2)
0	0	0	4973	4520	5414	5026.666667	4574	5367
1	0	1	-14956	-15174	-14811	19941.333...	19748	20232
2	0	2	25023	24785	25277	-10028.333...	-10367	-9711
3	0	3	4982.466667	4892	5080	4977.333333	4885	5080
4	0	4	5173.6	4549	5865	4826.866667	4136	5452
5	1	0	20117.6	19776	20468	-15088.2	-15351	-14832
6	1	1	0	0	0	0	0	0
7	1	2	40004	40004	40004	-30003	-30003	-30003
8	1	3	4	4	4	-3	-3	-3

12. ábra - Sorok megtekintése

Grafikonok készítéséhez az *IPD View table* kiválasztása után nyomja meg a gombot az ikonsoron, vagy válassza a *Charts > Create charts...* menüpontot.

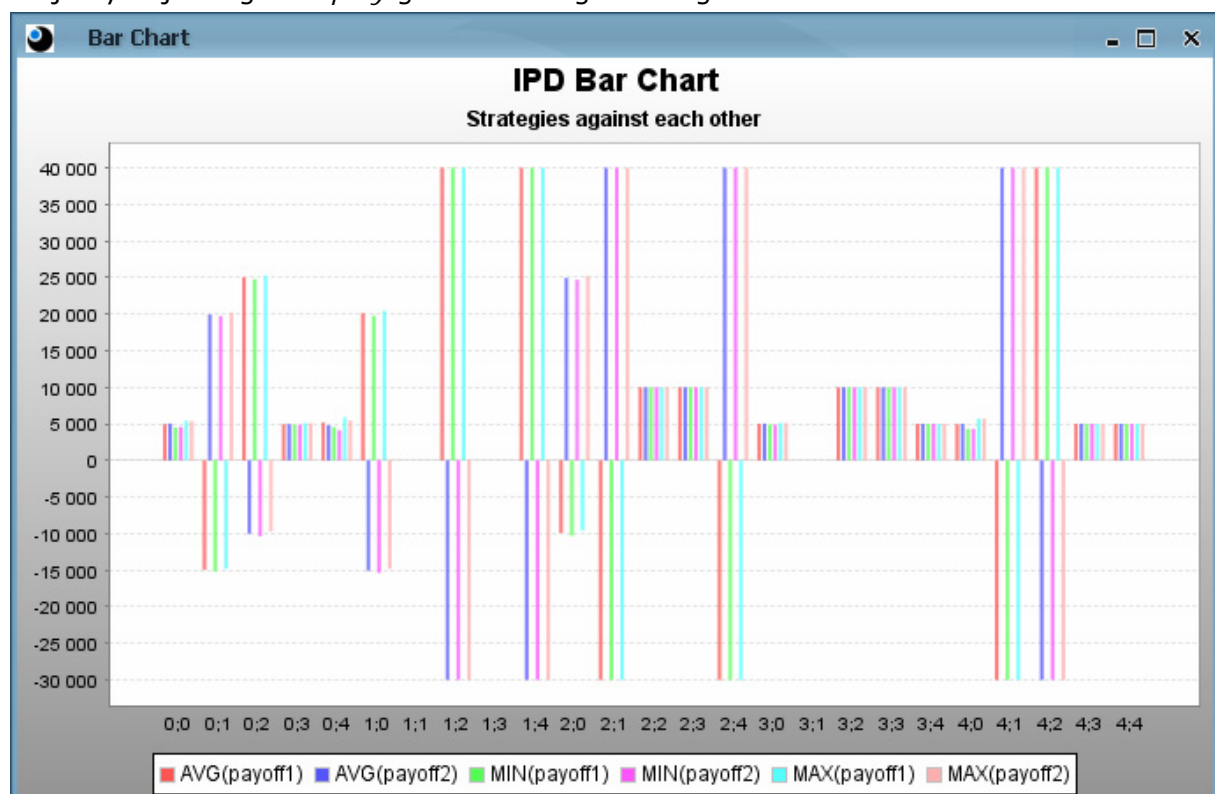
Válassza a *Bar Chart*-ot az *Available Chart types* listáról, és kattintson a *Create* gombra.



13. ábra - Oszlopdiaagram készítése

4.1 Oszlopdiaagram készítése

A *Main Settings* fülön állítsa be a *Strategies* oszlopot a *Categories* mezőben, és a *Data row* legördülő listából adja hozzá az *AVG(payload1)*, *AVG(payload2)*, *MIN(payload1)*, ..., *MAX(payload2)* oszlopokat a *Selected data rows* listához. A sorrend megtartása ajánlott az oszlopok könnyebb összehasonlítása végett. Adjon meg címet és alcímet a *Details* fülön, majd nyomja meg a *Display* gombot a diagram megtekintéséhez.



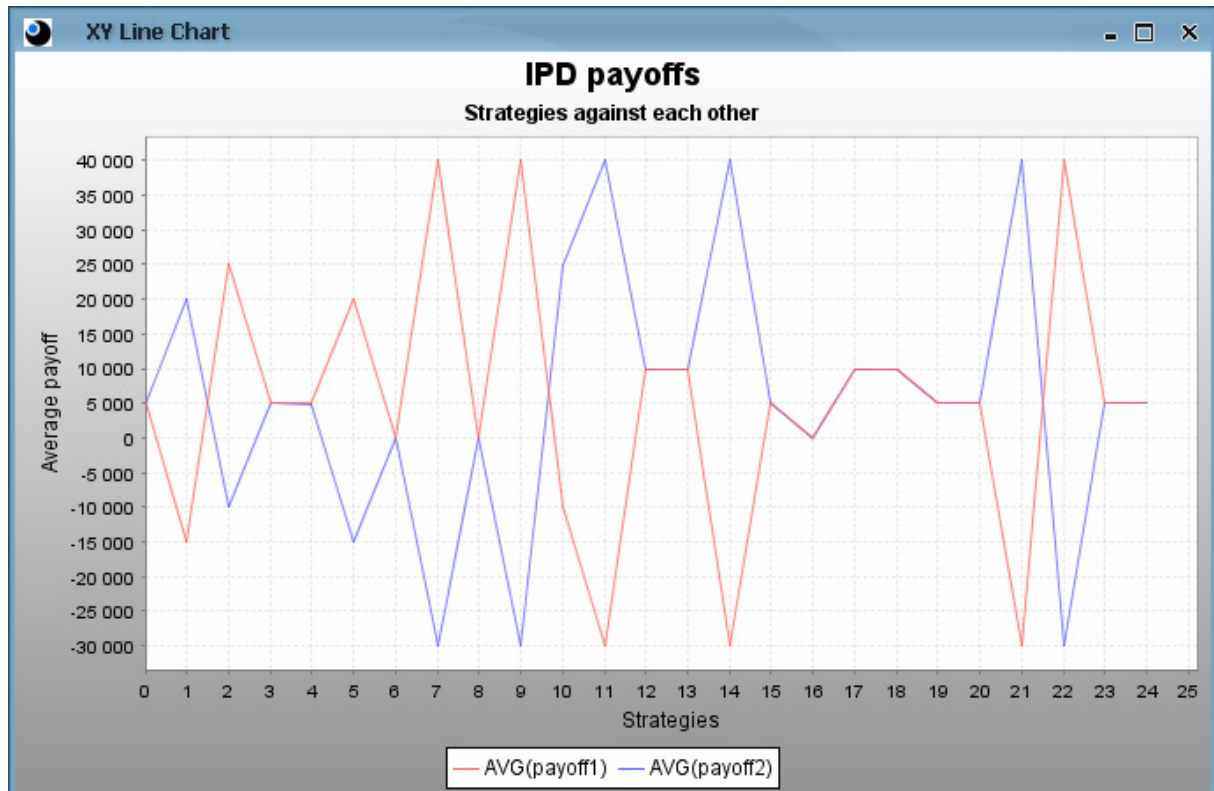
14. ábra - Oszlopdiaagram

Ahogy az a fenti ábrán látható, az egy payoff-hoz tartozó átlag, minimum, és maximum értékek lényegében megegyeznek.

4.2 Vonaldiagram készítése

Kattintson a *Chart collection* mező alatt lévő *New Chart* gombra, válassza az *XY Line Chart* elemet az *Available chart types* listából, majd nyomja meg a *Create* gombot. Az *X*

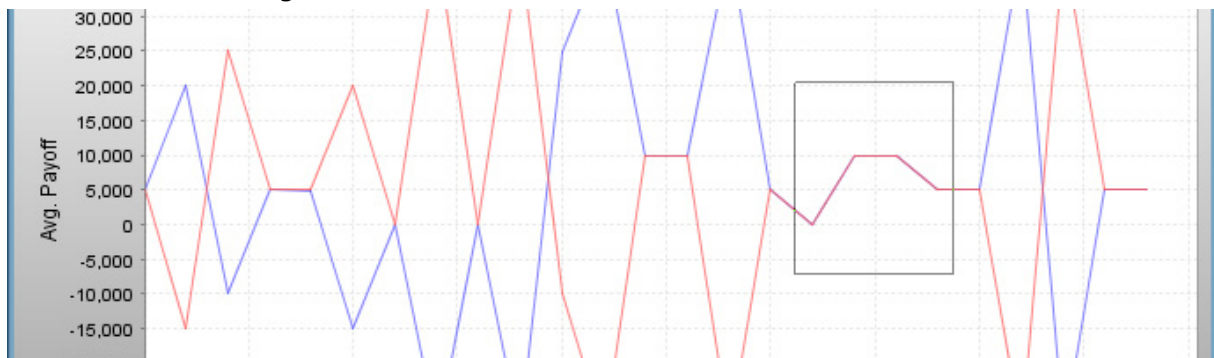
values mezőben állítsa be a *Strategies* oszlopot, ezután az *Y values* mezőben válassza ki az $AVG(\text{payoff1})$ oszlopot, majd nyomja meg az **Add** gombot. Az előzőekhez hasonlóan adja hozzá a diagramhoz az $AVG(\text{payoff2})$ oszlopot, majd adja meg a diagram címét, alcímét, és a tengelyek feliratait (ez utóbbi adatok megadása nem kötelező).



15. ábra - Vonaldiagram

4.3 Nagyítás

A diagram érdekesebb részeinek nagyításához adja meg a nagyítani kívánt téglalapot: ezt az egér bal gombjának nyomva tartása mellett, a kurzor jobbra és lefelé történő húzásával teheti meg.



16. ábra - Terület kiválasztása

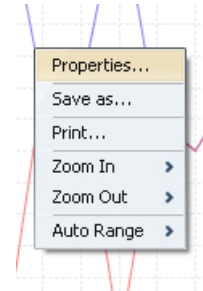
Többszöri nagyítás után kiderült, hogy a 4:0 stratégia esetén a *payoff2* valamivel magasabb, mint a *payoff1*. A nagyításból történő visszalépéshez az egér gombjának nyomva tartása mellett húzza a kurzort balra és fölfelé.



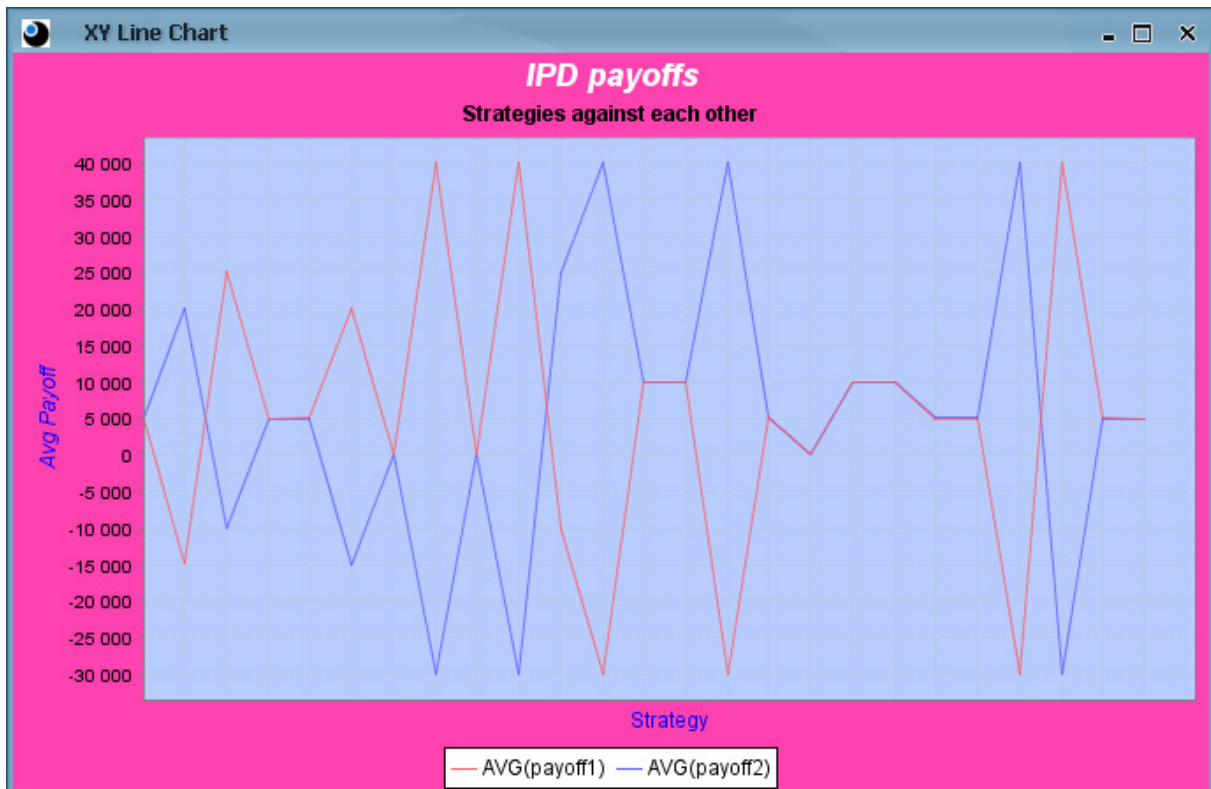
17. ábra - Kis eltérés

4.4 Ábrák exportálása és formázása

A diagramra jobb egérgombbal kattintva felugrik egy menü. Ebből a menüből nagyíthat/kicsinyíthet (*Zoom In/Out*), EPS vagy PNG fájlba mentheti (*Save as...*) és nyomtathatja (*Print...*) a grafikont, illetve a átformálhatja annak megjelenését (*Properties...*). A *Properties* ablakban a címek, feliratok változtathatók meg, és beállíthatók a színek, a betűtípus, a keret, illetve a tengelyek tulajdonságai.



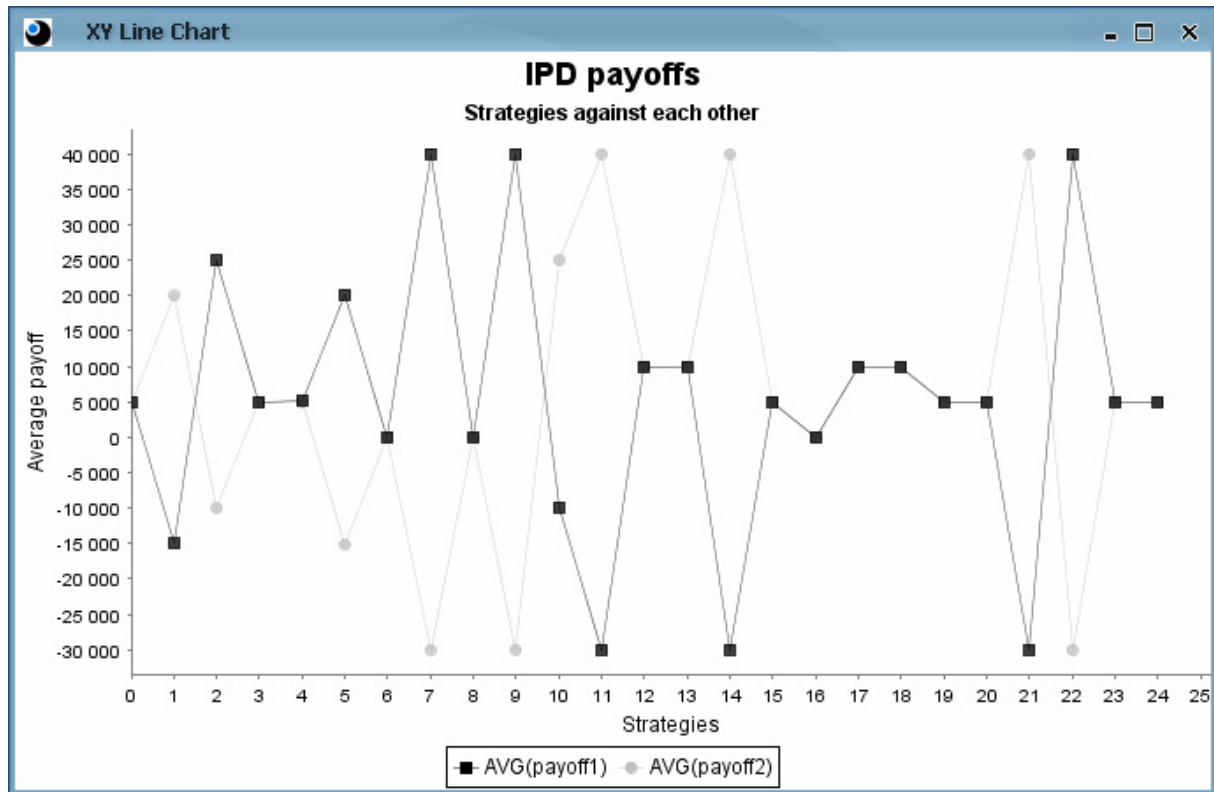
A cél, hogy a lenti ábránál szebb végeredmény készüljön:



18. ábra - Egy nem szépségdíjas alkotás

4.5 Beépített megjelenítések

A *Details* fülön négy előre elkészített megjelenítési beállítás közül lehet választani. A *Basic black-and-white* például megfelelő formátum fekete-fehér cikkekhez.



19. ábra - Basic black-and-white megjelenítés

4.6 A diagramok mentése

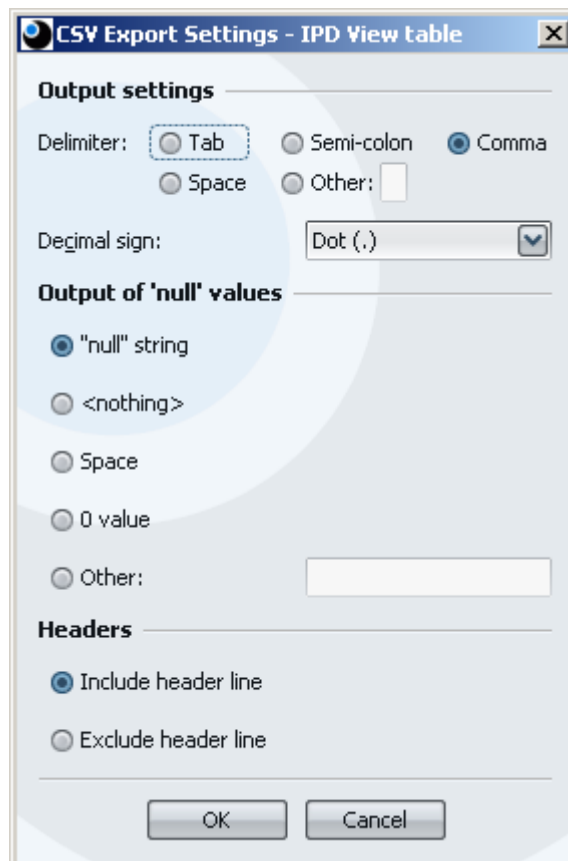
Az ábrák elkészítésének végeztével a nézet-táblához készített diagramok a *Save* gomb megnyomásával elmenthetők. A MEME a diagramot leíró információkat a felhasználó által meghatározott helyre menti, xml formátumban. A diagramok megnyitásakor (*Charts > Open chart*) gondoskodjunk róla, hogy az adatokat tartalmazó nézet-tábla az adatbázisban van. Különben, az elmentett diagramok nem fognak megfelelően megjelenni (vagy egyáltalán nem jelennek meg).

5 Exportálás

A feldolgozott eredmények adatbázisból való kiexportálásához más alkalmazások számára váltson vissza a *Views* fülre. Az *IPD View table* kiválasztása után nyomja meg az



gombot az ikonsoron, vagy válassza a *File > Export...* menüpontot. A *CSV Export Settings* dialógusablak (lásd lent!) megnyitásához mindkét esetben válassza a *CSV file* opciót a felugró menüből.



20. ábra - CSV Export Settings dialógusablak

Alapértelmezett beállítások esetén a mezőket vesszők fogják tagolni, és a tizedes elválasztó a pont lesz. Mivel a mostani nézet-táblában nincs „null” érték, ennek a kezelése nem fontos, de mint a fenti ábrán látható, öt lehetőség közül lehet választani. Alapértelmezésként az oszlopnevek is a fájlba kerülnek, de ez is megváltoztatható.

Az *OK* gomb megnyomása után egy fájlválasztó ablak nyílik meg, ahol megadható a fájl neve és helye. Az exportáláshoz kattintson a *Save* gombra.

6 Megjegyzések

A MEME installer tartalmazza az iterált fogolydilemma Java implementációját (a forráskódot és a bináris állományokat egyaránt). Ezek alapértelmezésben a C:\Program Files\MASS\MEME\documents\prisoners_demo\demo\prisoners\ könyvtárba kerülnek. Más telepítési könyvtár esetén azon belül a documents\prisoners_demo\demo\prisoners mappában keresse őket.

A fenti modell más paraméterfájllal való futtatásához használhatja a MEME Parameter Sweep Tool alkalmazást (a MEME installer tartalmazza). Ehhez válassza ki a PrisonersModelBatch.class fájlt a program első oldalán. A részletekért lásd [5]-öt! A MEME Parameter Sweep Tool lehetővé teszi Ön számára, hogy elosztott módon futtassa a modellt egy griden vagy klaszteren, de a helyi gépen való futtatás is támogatott. Mindkét esetben nyomon lehet követni a szimuláció haladását.

A közeljövőben a MEME-t képessé tesszük a MASS/Repast szimulációk közvetlen futtatására és az eredmények belső gyűjtésére, illetve lehetővé tesszük, hogy olyan kifinomult statisztikai szoftverekkel működjön együtt, mint a Matlab, vagy az R. Ezek a fejlesztések tovább fogják egyszerűsíteni a modellezés folyamatát, aminek eredményeképpen a felhasználó még inkább magukra a modellekre koncentrálhat.

7 Referenciák

- [1] MASS – Multi-Agent Simulation Suite. © AITIA International Inc.
http://www.aitia.ai/services_and_products/simulation_systems/mass
- [2] Repast - Recursive Porus Agent Simulation Toolkit
<http://repast.sourceforge.net/>
- [3] Beanshell Scripting
<http://www.beanshell.org/>
- [4] Iterated Prisoner's Dilemma
http://en.wikipedia.org/wiki/Iterated_Prisoners_Dilemma
- [5] Parameter Sweep Eszköz Felhasználói kézikönyv

8 Függelék

8.1 PrisonersAgent.java

```
package demo.prisoners;

import uchicago.src.sim.util.Random;

/** Az Agent osztály az iterált fogolydilemma játék emlékezzettel rendelkező
(vagy nem rendelkező) játékosát reprezentálja. */
public class PrisonersAgent {

    /** Stratégia: Véletlen */
    public static final int RND = 0;
    /** Stratégia: Mindig defekt */
    public static final int ALLD = 1;
    /** Stratégia: Mindig kooperál */
    public static final int ALLC = 2;
    /** Stratégia: „Tit-for-tat” */
    public static final int TFT = 3;
    /** Stratégia: Anti-„Tit-for-tat” */
    public static final int ATFT = 4;

    /** A játékos aktuális stratégiája */
    protected int strategy;
    /** A másik játékos utolsó lépése */
    protected boolean enemyLast;

    public PrisonersAgent(int strategy) {
        Random.createUniform();
        this.strategy=strategy;
        enemyLast=true;
    }

    /** Az ellenfél utolsó lépésének megjegyzése. */
    public void setEnemyLast(boolean b) {
        enemyLast = b;
    }

    /** Kooperálás esetén igazzal tér vissza. */
    public boolean cooperate() {
        switch (strategy) {
            case TFT:    return enemyLast;
            case ATFT:   return !enemyLast;
            case ALLD:   return false;
            case ALLC:   return true;
            case RND:    return //Random.uniform.nextBoolean();
                uchicago.src.sim.util.Random.uniform.nextBoolean();
        }
        return true;
    }
}
```

8.2 PrisonersModel.java

```
package demo.prisoners;

import uchicago.src.sim.engine.*;

/**Az iterált fogolydilemma játék modellje. */
public class PrisonersModel extends SimpleModel {

    public PrisonersModel() {
        super();
        name = "Prisoner's Dilemma";
    }
}
```

```
}

/** A győztes jutalma. */
protected int winner;
/** Visszaadja a győztes jutalmát. */
public int getWinner() { return winner; }
/** Beállítja a győztes jutalmát. */
public void setWinner(int winner) { this.winner=winner; }
/** A vesztes jutalma. */
protected int loser;
/** Visszaadja a vesztes jutalmát. */
public int getLooser() { return loser; }
/** Beállítja a vesztes jutalmát. */
public void setLooser(int loser) { this.looser=loser; }
/** Mindkét játékos jutalma kooperálás esetén. */
protected int both;
/** Visszaadja a kooperálás jutalmát. */
public int getBoth() { return both; }
/** Beállítja a kooperálás jutalmát. */
public void setBoth(int both) { this.both=both; }
/** Mindkét játékos jutalma dupla defektálás esetén. */
protected int neither;
/** Visszaadja a dupla defektálás jutalmát. */
public int getNeither() { return neither; }
/** Beállítja a dupla defektálás jutalmát. */
public void setNeither(int neither) { this.neither=neither; }
/** Az első játékos stratégiája. */
protected int strat1;
/** Beállítja az első játékos stratégiáját. */
public int getStrat1() { return strat1; }
/** Visszaadja az első játékos stratégiáját. */
public void setStrat1(int strat1) { this.strat1=strat1; }
/** A második játékos stratégiája. */
protected int strat2;
/** Beállítja a második játékos stratégiáját. */
public int getStrat2() { return strat2; }
/** Visszaadja a második játékos stratégiáját. */
public void setStrat2(int strat2) { this.strat2=strat2; }

/** Az első játékos jutalma.*/
protected int payoff1;
public void setPayoff1(int i) { payoff1 = i; }
public int getPayoff1() { return payoff1; }
/** A második játékos jutalma. */
protected int payoff2;
public void setPayoff2(int i) { payoff2 = i; }
public int getPayoff2() { return payoff2; }

public String[] getInitParam() {
    String[] params = {"winner","looser","both","neither",
        "strat1","strat2","payoff1","payoff2"};
    return params;
}

public void setup() {
    super.setup();
    generateNewSeed();
    payoff1 = 0;
    payoff2 = 0;
}

@SuppressWarnings("unchecked")
public void buildModel() {
    PrisonersAgent a = new PrisonersAgent(strat1);
    agentList.add(a);
    PrisonersAgent b = new PrisonersAgent(strat2);
    agentList.add(b);
}
}
```

```

public void step() {
    PrisonersAgent a = (PrisonersAgent)agentList.get(0);
    PrisonersAgent b = (PrisonersAgent)agentList.get(1);
    boolean cA = a.cooperate();
    boolean cB = b.cooperate();
    if (cA && cB) {
        payoff1+=both;
        payoff2+=both;
    }
    if (cA && !cB) {
        payoff1+=looser;
        payoff2+=winner;
    }
    if (!cA && cB) {
        payoff1+=winner;
        payoff2+=looser;
    }
    if (!cA && !cB) {
        payoff1+=neither;
        payoff2+=neither;
    }
    a.setEnemyLast(cB);
    b.setEnemyLast(cA);
}

public void atEnd() {
    super.atEnd();
}
}

```

8.3 PrisonersModelBatch.java

```

package demo.prisoners;

import uchicago.src.sim.engine.BasicAction;
import uchicago.src.sim.analysis.*;

/** Az iterált fogolydilemma játék batch módú futtatáshoz.*/
public class PrisonersModelBatch extends PrisonersModel {

    /** A modellleáll, ha az aktuális lépésszám eléri ezt az értéket.*/
    protected int stopAt;
    protected int stratv1;
    protected int stratv2;

    public int    getStopAt()           { return stopAt; }
    public int    getStratv1()          { return stratv1; }
    public int    getStratv2()          { return stratv2; }
    public void   setStopAt(int stopAt) { this.stopAt=stopAt; }
    public void   setStratv1(int i)     { stratv1=i; }
    public void   setStratv2(int i)     { stratv2=i; }

    protected DataRecorder recorder;

    public String[] getInitParam() {
        String[] sparams = super.getInitParam();
        int l = sparams.length;
        String[] params = new String[l+3];
        int i;
        for (i=0; i<l; ++i) {
            params[i]=sparams[i];
        }
        params[i ]="stopAt";
        params[i+1]="stratv1";
        params[i+2]="stratv1";
        return params;
    }
}

```

```

    }

    public void setup() {
        stopAt = 1000;
        super.setup();
    }

    public void buildModel() {
        super.buildModel();
        buildRecorder();
        this.run();
    }

    protected void buildRecorder() {
        class NumDataSource implements NumericDataSource {
            private int parIdx;
            PrisonersModel model;
            NumDataSource(int parIdx) {
                this.parIdx = parIdx;
                this.model = model;
            }
            public double execute() {
                double ans = 0;
                switch (parIdx) {
                    case 0 : ans = getPayoff1(); break;
                    case 1 : ans = getPayoff2(); break;
                    case 2 : ans = getStratv1(); break;
                    case 3 : ans = getStratv2(); break;
                }
                return ans;
            }
        }
        recorder = new DataRecorder(this.getName()+".out",this);
        recorder.setDelimiter(" -- ");
        recorder.addNumericDataSource("payoff1", new NumDataSource(0));
        recorder.addNumericDataSource("payoff2", new NumDataSource(1));
        recorder.addNumericDataSource("stratv1", new NumDataSource(2));
        recorder.addNumericDataSource("stratv2", new NumDataSource(3));
    }

    public void buildSchedule() {
        super.buildSchedule();
        class StopAction extends BasicAction {
            public void execute() {
                stop();
            }
        }
        schedule.scheduleActionAt(stopAt, new StopAction());
    }

    public void atEnd() {
        super.atEnd();
        stratv1 = strat1 * stopAt / 2;
        stratv2 = strat2 * stopAt / 2;
        recorder.record();
        recorder.writeToFile();
    }
}

```

8.4 PrisonersModelGUI.java

```

package demo.prisoners;

import uchicago.src.sim.analysis.*;

/** Az iterált fogolydilemma játék gui módja. */
public class PrisonersModelGUI extends PrisonersModel {

```

```
/** A GUI módban megjelenített gráf. */
protected OpenSequenceGraph graph;

class Payoff implements Sequence {
    private int player;
    public Payoff(int player) { this.player = player; }
    public double getSValue() {
        return (double)(player==1?payoff1:payoff2);
    }
}

public void setup() {
    super.setup();
    winner=4;
    loser=-3;
    both=1;
    neither=0;
    strat1 = PrisonersAgent.RND;
    strat2 = PrisonersAgent.RND;
}

public void buildModel() {
    super.buildModel();
    if (graph!=null) graph.dispose();
    graph=new OpenSequenceGraph("Payoff",this);
    graph.setXRange(0, 50);
    graph.setYRange(-30, 170);
    graph.setXViewPolicy(OpenSequenceGraph.SHOW_LAST);
    graph.addSequence("1st player", new Payoff(1));
    graph.addSequence("2nd player", new Payoff(2));
    graph.display();
}

public void step() {
    super.step();
    graph.step();
}
}
```