Tutorial

Prepared by
Tibor Baranya, Róbert Mészáros,
Márton Iványi, Rajmund Bocsi
and
Viktor Erdélyi

July 2010, Budapest

© 2009 AITIA International Inc.

Table of content

1 Introduction 3
1.1 Iterated Prisoner's Dilemma 3
2 Creating and Running the Batch Version of the Model 5
2.1 Platform Selection 5
2.2 Model Selection 6
2.3 Method Selection 7
2.4 Parameters 8
2.5 Data Collection 10
2.6 Monitoring the Simulation 12
3 Importing the Results 14
4 Creating a View Table 16
4.1 Computations 17
4.1.1 Aggregative Operations 17
4.1.2 Scripting 18
4.2 Sorting 18
4.3 View Name and Description 19
5 Visualization 20
5.1 Creating a Bar Chart 21
5.2 Creating a Line Chart 21
5.3 Zooming 22
5.4 Exporting and formatting charts 23
5.5 Default Appearances 23
5.6 Saving the Chart Collection 24
6 Export 25
7 Notes 26
8 References 27
9 Appendix 28
9.1 PrisonersAgent.java 28
9.2 PrisonersModel.java 28
9.3 PrisonersModelGUI.java 30

1 Introduction

This tutorial introduces you to the MEME program through an example RepastJ model. You will be shown how MEME works and how it can help you to run simulations in batch mode and then to store, to sort and to visualize data from multiple runs of simulations.

MEME is a tool for running simulations in batch mode, using various parameter sets and handling the thus created data. It enables the user to store and systemize the raw data in databases(s), create proper datasets and visualize them.

Please consult the MEME User Manual (*MEME_Manual.pdf*) for general and installation information.

We will use an example model, called Iterated Prisoner's Dilemma to present the use and advantages of MEME. This tutorial guides you through the creation and run of batch version of a RepastJ model, the import of data created by the batch running and the analysis and visualization of the data extracted from the batch mode runs.

1.1 Iterated Prisoner's Dilemma

In game theory, the prisoner's dilemma (PD) is a type of non-zero-sum game in which two players can "cooperate" with or "defect" (i.e. betray) the other player. In this game, as in all game theory, the only concern of each individual player ("prisoner") is maximizing his/her own payoff, without any concern for the other player's payoff. In the classic form of this game, cooperating is strictly dominated by defecting, so that the only possible equilibrium for the game is for all players to defect. In simpler terms, no matter what the other player does, one player will always gain a greater payoff by playing defect. Since in any situation playing defect is more beneficial than cooperating, all rational players will play defect.

The unique equilibrium for this game is a Pareto-suboptimal solution—that is, rational choice leads the two players to both play defect even though each player's individual reward would be greater if they both played cooperate. In equilibrium, each prisoner chooses to defect even though both would be better off by cooperating, hence the dilemma.

In the iterated prisoner's dilemma the game is played repeatedly. Thus each player has an opportunity to "punish" the other player for previous non-cooperative play. Cooperation may then arise as an equilibrium outcome. The incentive to defect is overcome by the threat of punishment, leading to the possibility of a cooperative outcome. If the game result is infinitely repeated, cooperation may be Nash equilibrium although both players defecting always remain equilibrium.

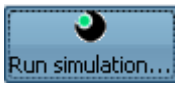
In his book The Evolution of Cooperation (1984), Robert Axelrod explored an extension to the classical PD scenario, which he called the iterated prisoner's dilemma (IPD). In this, participants have to choose their mutual strategy again and again, and have memory of their previous encounters. Axelrod invited academic colleagues all over the world to devise computer strategies to compete in an IPD tournament. The programs that were entered varied widely in algorithmic complexity; initial hostility; capacity for forgiveness; and so forth.

Axelrod discovered that when these encounters were repeated over a long period of time with many players, each with different strategies, "greedy" strategies tended to do very poorly in the long run while more "altruistic" strategies did better, as judged purely by self-interest. He used this to show a possible mechanism for the evolution of altruistic behavior from mechanisms that are initially purely selfish, by natural selection.

The best deterministic strategy was found to be "Tit for Tat", which Anatol Rapoport developed and entered into the tournament. It was the simplest of any program entered, containing only four lines of BASIC, and won the contest. The strategy is simply to cooperate on the first iteration of the game; after that, the player does what his opponent did on the previous move. A slightly better strategy is "Tit for Tat with

forgiveness". When the opponent defects, on the next move, the player sometimes cooperates anyway, with a small probability (around 1%-5%). This allows for occasional recovery from getting trapped in a cycle of defections. The exact probability depends on the line-up of opponents. "Tit for Tat with forgiveness" is best when miscommunication is introduced to the game — when one's move is incorrectly reported to the opponent.

2 Creating and Running the Batch Version of the Model

Start MEME and press the 

button or *Parameter Sweep/Run simulation…* in the menu to invoke the *Parameter Sweep Wizard*.

2.1 Platform Selection

First you have to specify the platform of the model on Platform selection page:

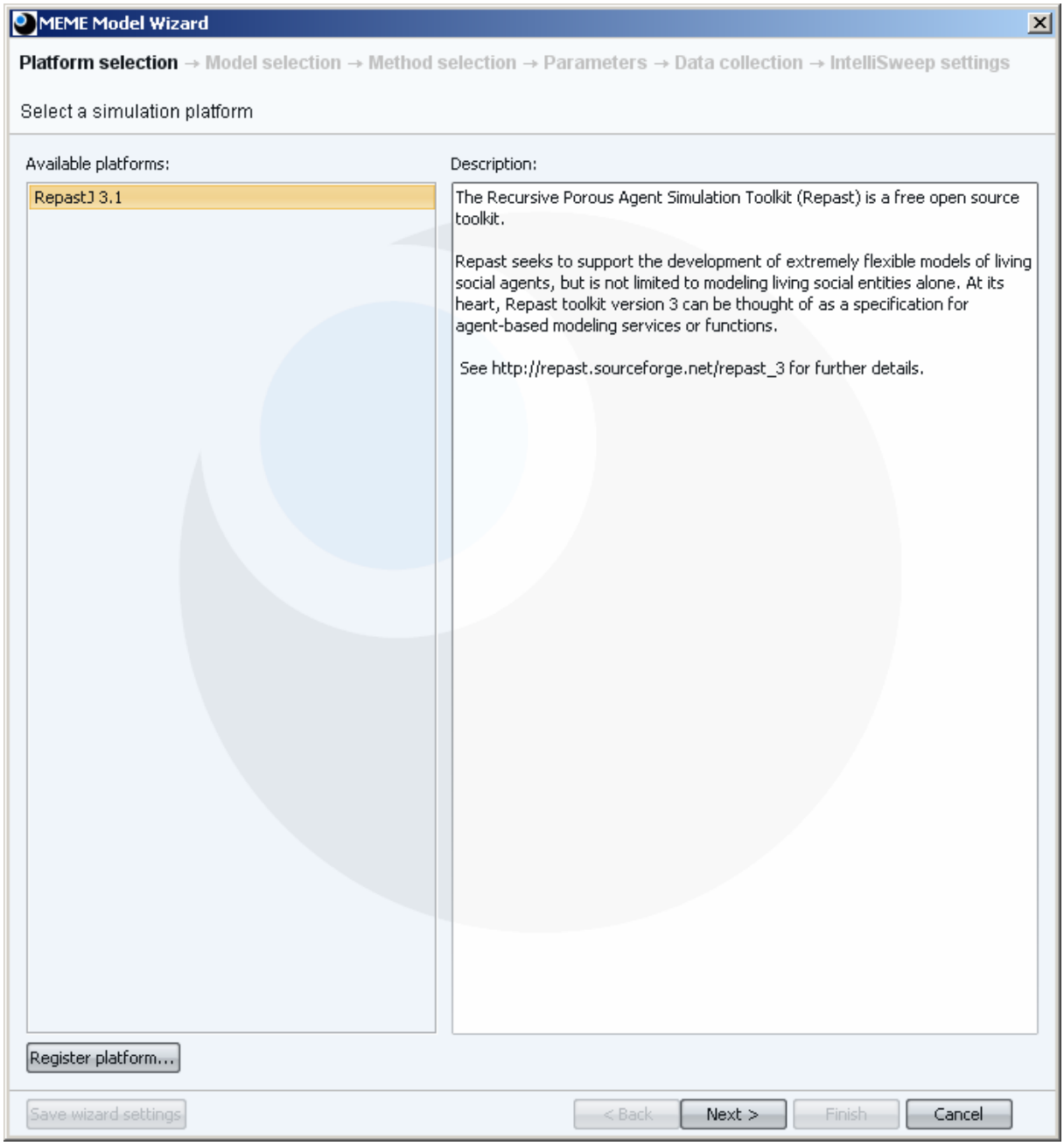


Figure 1 – Platform selection

The Iterated Prisoner's Dilemma model is written in RepastJ 3.1 so select this platform from the list on the left then click *Next* button to continue.

Please note that RepastJ 3.1 is the only built-in platform in MEME, however you can install others such as NetLogo 4.0.4 or the platform for model written in pure Java.

2.2 Model Selection

In this page you can select the model you want to run in batch mode.

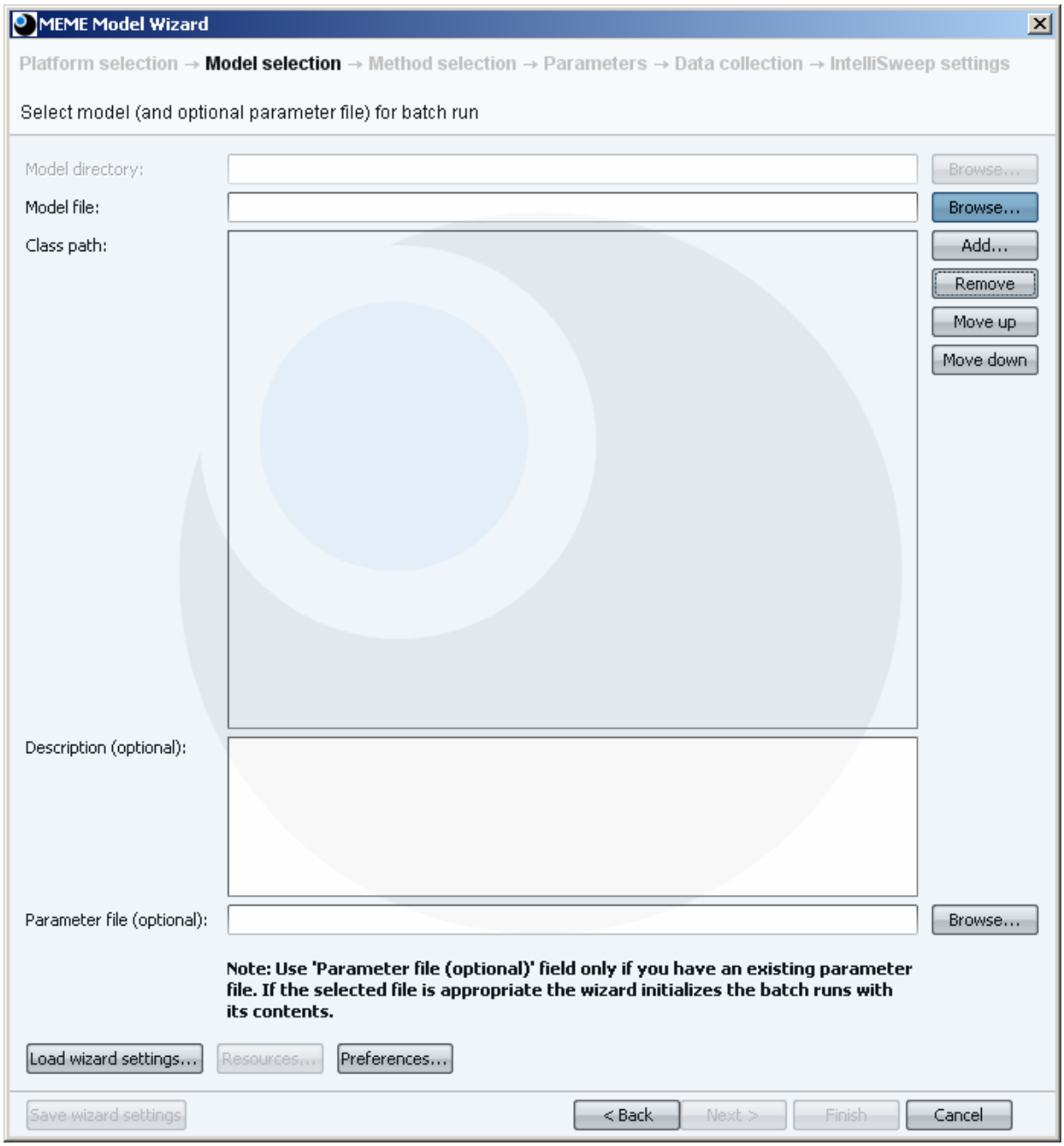


Figure 2 – Model selection

Press the *Browse…* button and select the *PrisonersModel.class* file (see below). The default destination of the file is *C:\Program Files\MASS\MEME\documents\prisoners_demo\demo\prisoners\PrisonersModel.class*. If you choose a different installation folder it is implicitly *…\documents\prisoners_demo\demo\prisoners\PrisonersModel.class*.

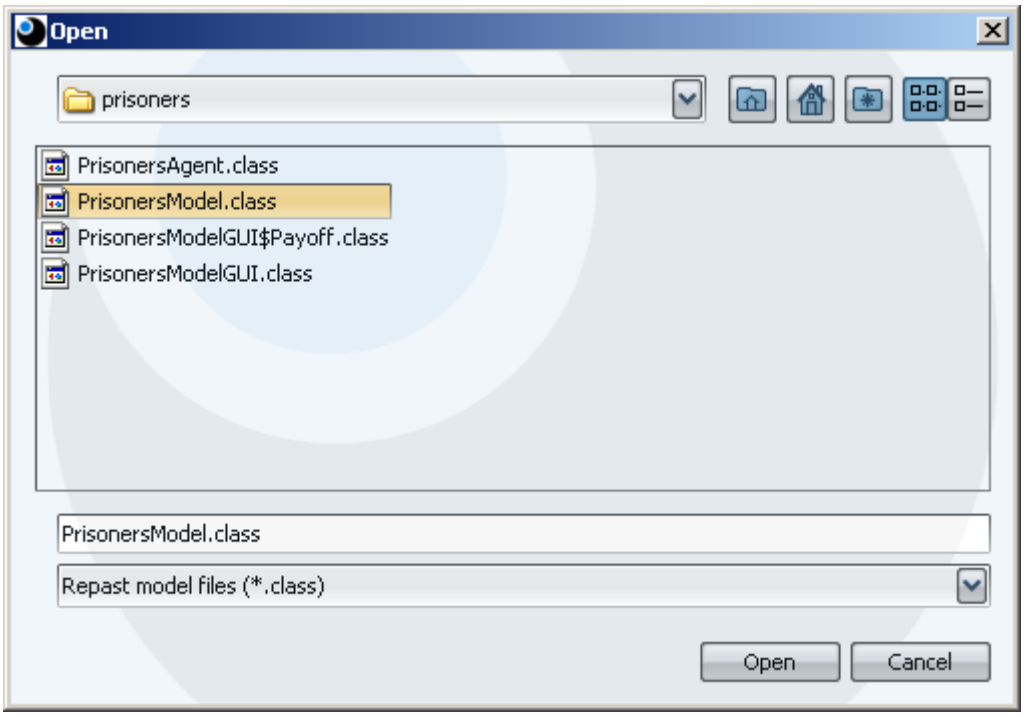


Figure 3 – Locate PrisonersModel.class

Upon pressing the *Open* button the wizard loads the model. Note that the class path is extended automatically with the directory of the model.

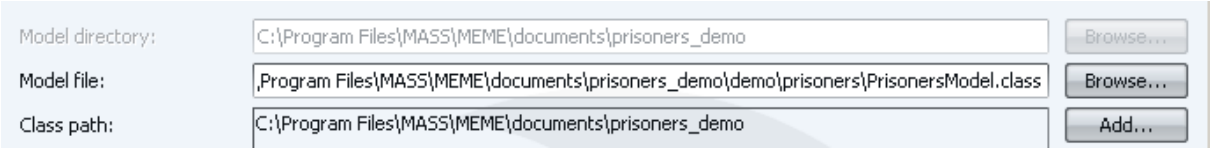


Figure 4 – Model loaded

You can optionally provide a description in the *Description* field (see below).

Figure 5 – Description

Press *Next* button to continue.

2.3 Method Selection

You can select the exploration method in this page (see below). This tutorial shows you the *Manual method*, so simply press the *Next* button again.

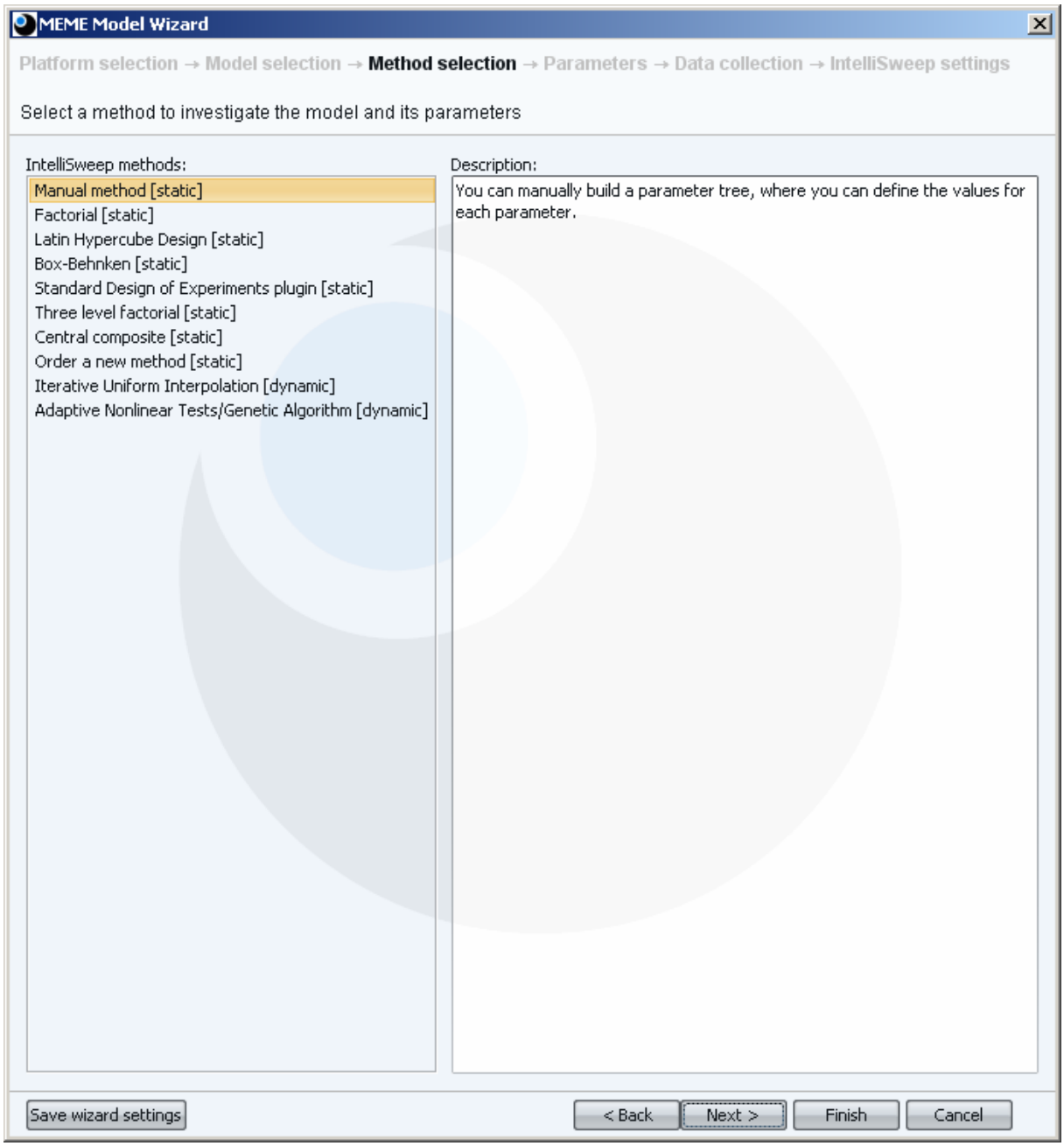


Figure 6 – Method selection

2.4 Parameters

The parameter space the model is going to be run on can be defined on this page (see Figure 7). The parameter combinations that are going to be explored are defined, in other words the parameter tree is created here.

The parameter tree is shown in the right side of the page. Select `Both` parameter from the tree and press the *Edit* button. You can change the value of the parameter on the left panel. Set the *Constant value* to 1 (see the figure below) and press the *Modify* button to apply the changes. Then change the values of parameters `Looser` and `Winner` to -3, 4, respectively. Do not forget to press the *Modify* button after every modification. Note that you do not need to change the constant value of the `Neither` parameter because the default value is right.

Figure 7 – Parameters

Then select the `Strat1` parameter and press the *Edit* button. Change the type of the parameter to *Increment* and set *Start value* to 0, *End value* to 4 and *Step* to 1 (see below). Press the *Modify* button to apply the changes.

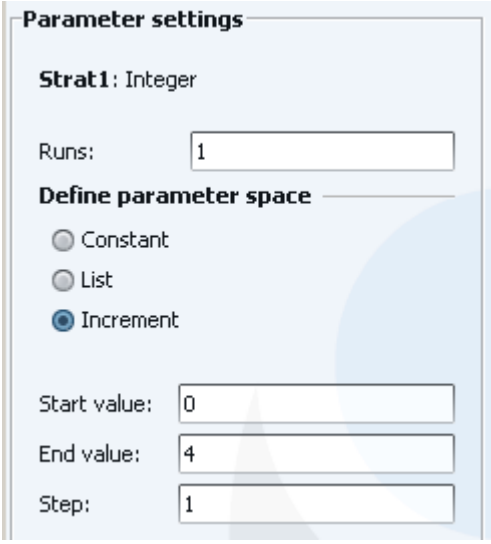


Figure 8 – Settings of Strat1 parameter

Do the same with `Strat2`.

Finally, select the `RngSeed` parameter and change its type to *Increment* then set its *Start value* to 1, its *End value* to 15 and its *Step* to 1.

Your last task in this page is to specify the parameter combinations. First drag the `Strat2` parameter and drop on the `Strat1` parameter. With this technique you move `Strat2` "under" `Strat1`. Do the same with `RngSeed`. If you did everything correctly you see the following parameter tree (and a 'A total of 375 runs' message under it):

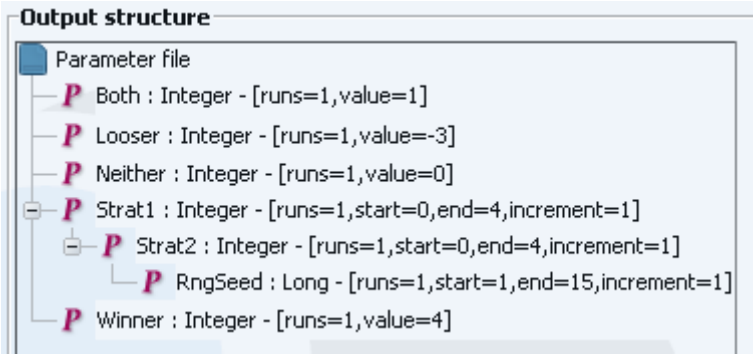


Figure 9 – The final parameter tree

When done, press *Next*.

2.5 Data Collection

The *Data collection* page of the wizard is shown on Figure 10. This page helps specifying what kind of information is to be collected during the experiment and allows defining the stop condition for each simulation.

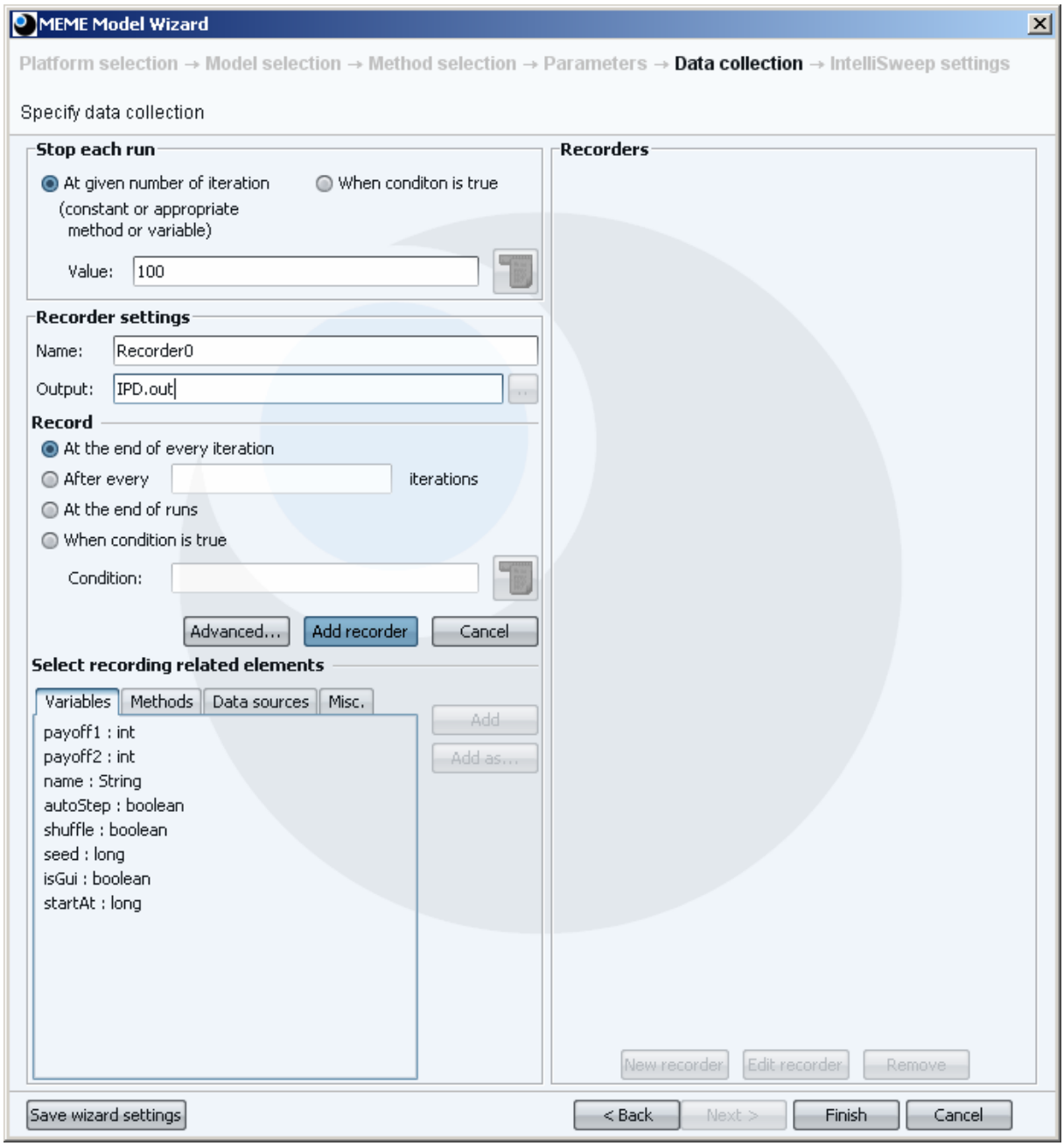


Figure 10 – Data collection

In the *Output* field (on the *Recorder settings* panel) enter *IPD.out* as shown on the figure above. Then select the *At the end of runs* radio button and then press the *Add recorder* button to create a recorder.

Then select `payoff1` and `payoff2` from the *Variables* list and press the *Add* button (see the figure below) to specify the recordable elements of the recorder defined earlier.

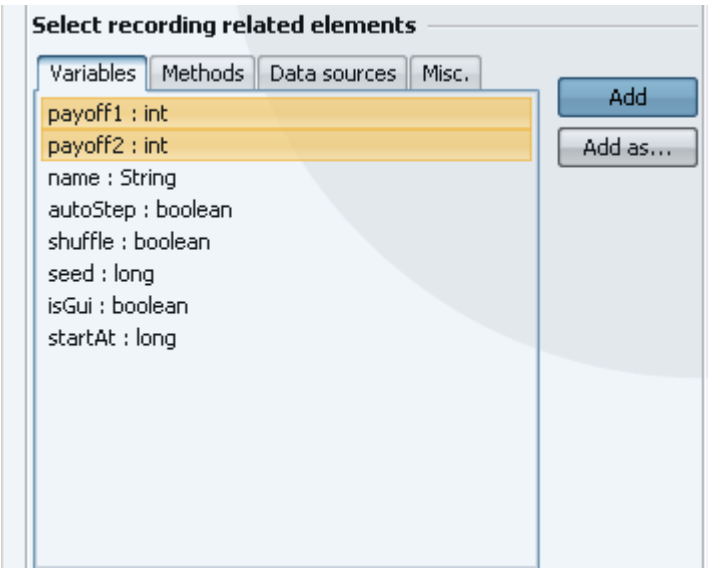


Figure 11 – Add recordable variables

If you did everything correctly you see the following recorder tree:

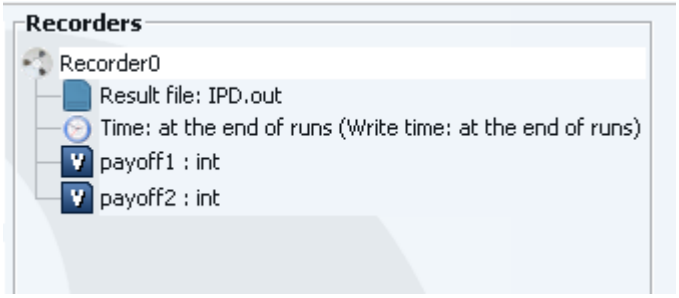


Figure 12 – The recorder tree

One task remains: you have to specify a stopping condition for the simulation. Enter 10000 in the *Value* field (on the *Stop each run* panel) to achieve this (see below).

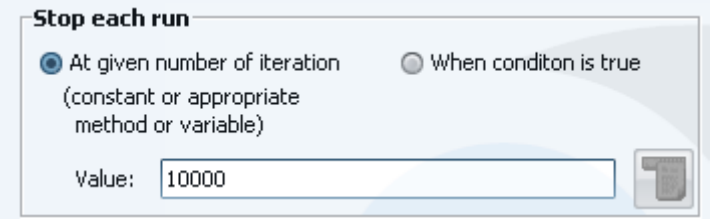


Figure 13 – Stopping condition

Press the *Finish* button to create and run the batch version of the selected model.

2.6 Monitoring the Simulation

Upon pressing the *Finish* button the *Local Monitor panel* appears (see below).

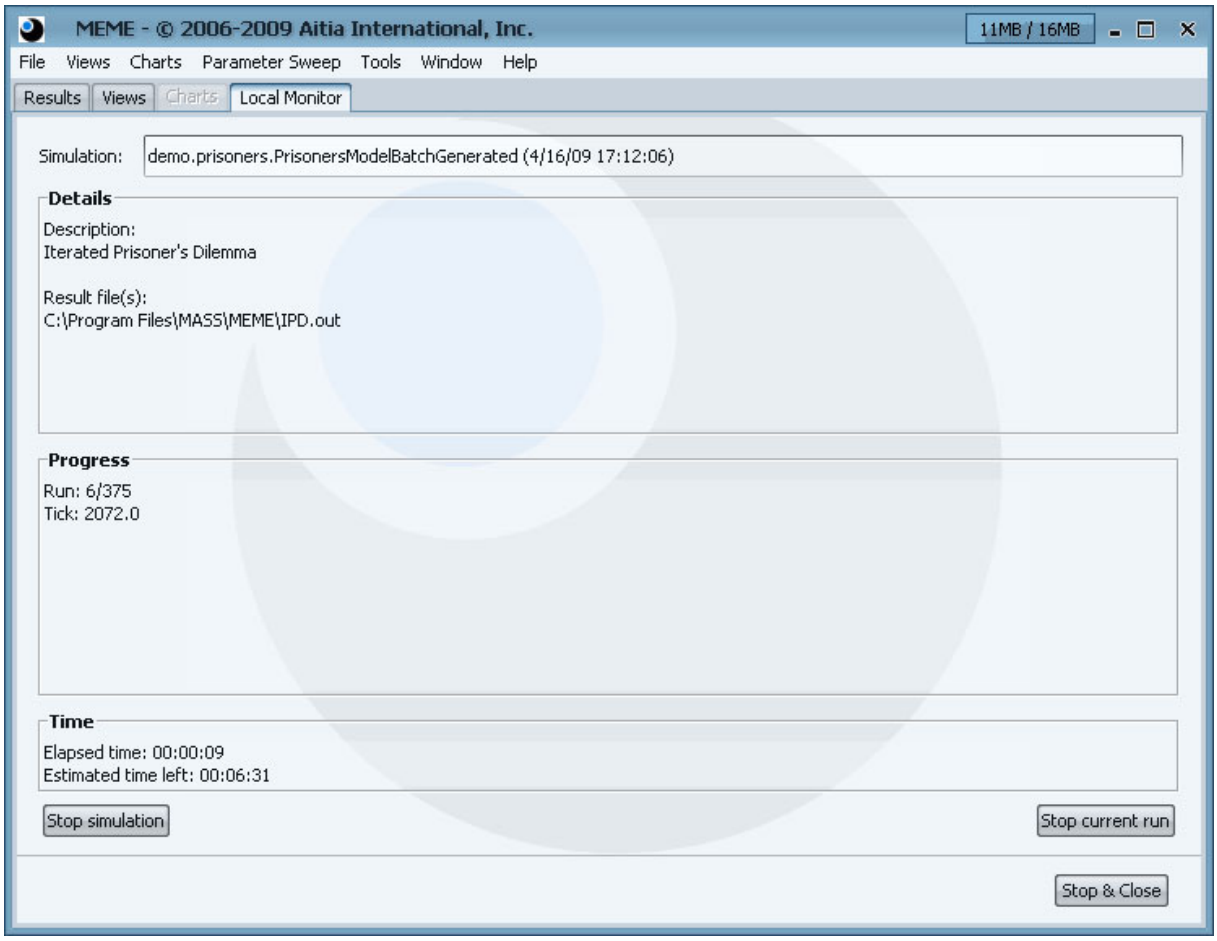


Figure 14 – The Local Monitor

3 Importing the Results

When the simulation is finished the *RepastJ Import Settings* dialogue automatically appears (see below).

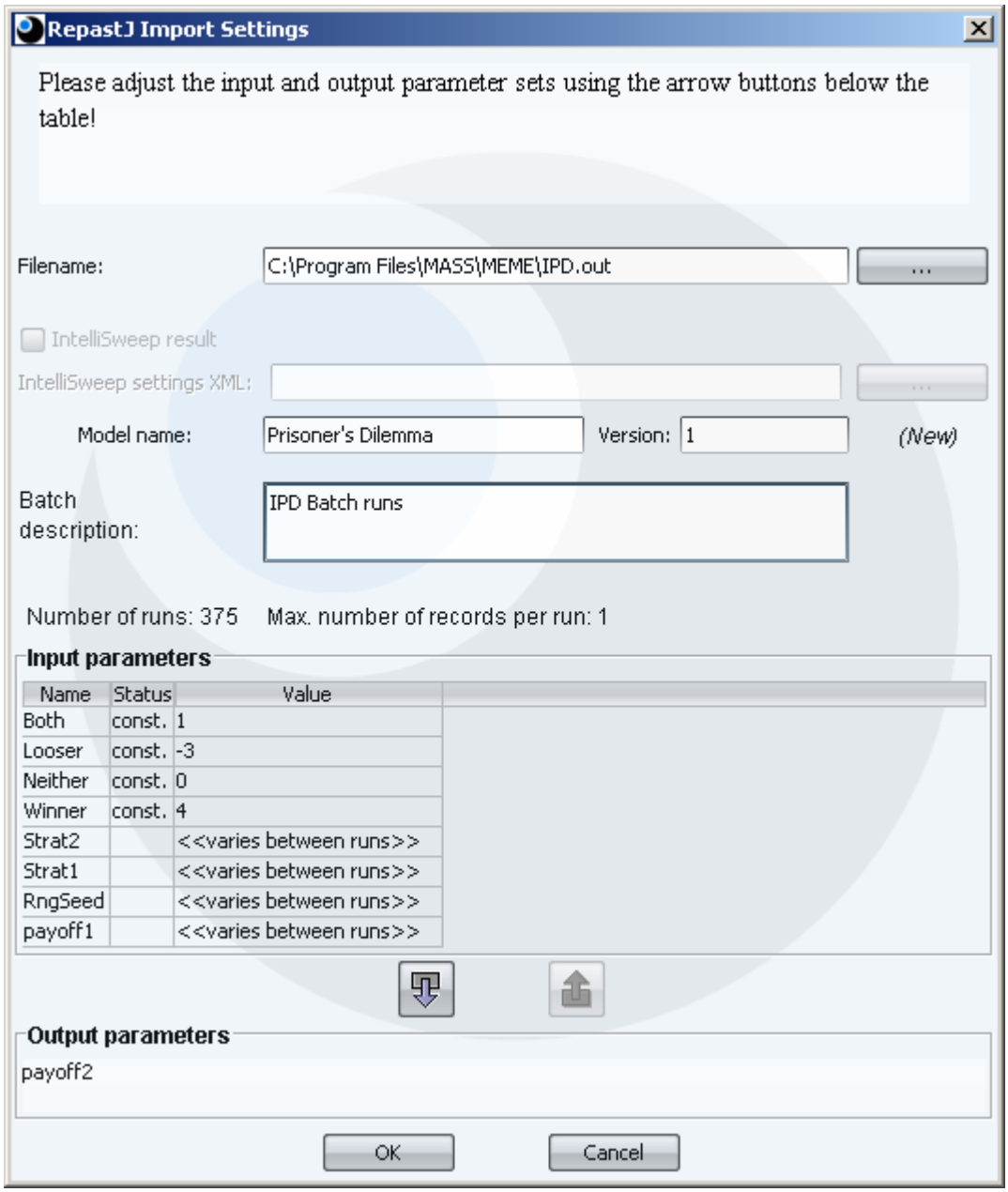


Figure 15 – Import Settings

In the *Model name* field enter Prisoner's dilemma as shown in the above picture. You can also provide (optional) a description in the *Description of the batch* field. Click the add button between the *Output parameters* and *Input parameters* boxes four times to have, `Strat1`, `Strat2`, `RngSeed` and `payoff1` added as output parameters.

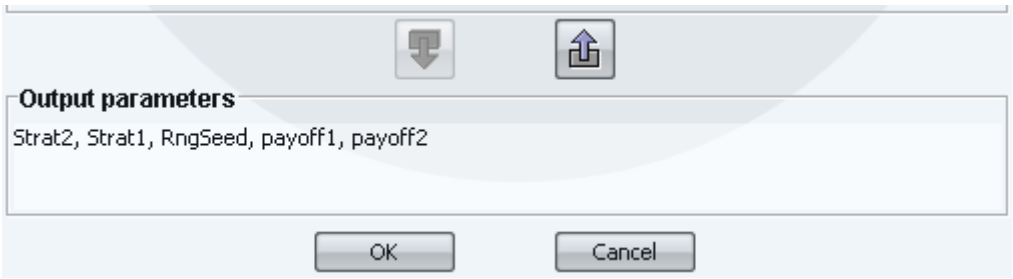


Figure 16 – Move to output

Press the *OK* button thus adding “Prisoner’s Dilemma” to the MEME database.

4 Creating a View Table

Before displaying a chart a view table must be created from the imported data. When creating a view table the imported data is processed and sorted according to the needs of the modeler.

Select the *Prisoner's Dilemma* model on the *Results* tab in MEME.

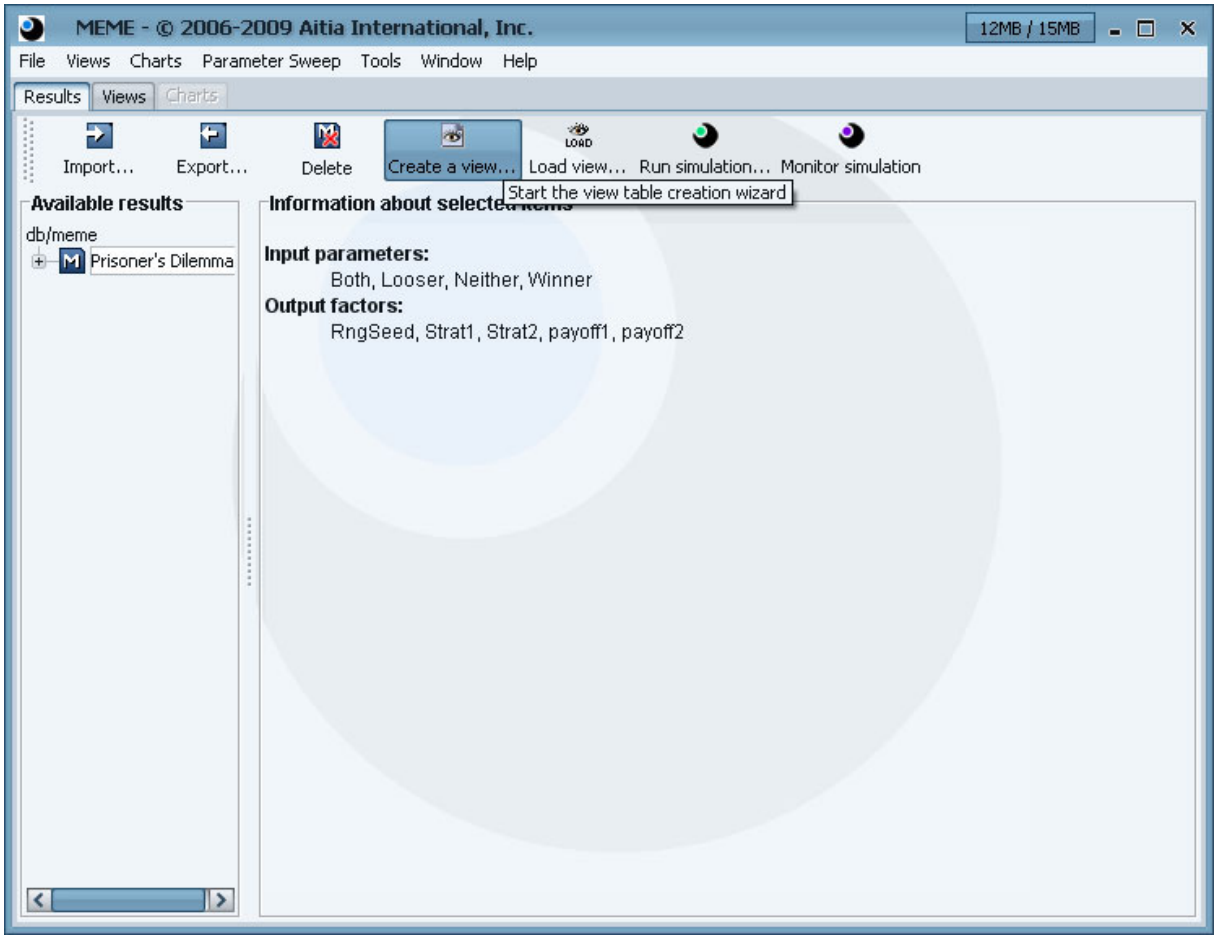
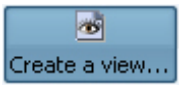


Figure 17 – Select batch

Press the 

button on the toolbar or *Views/Create a view…* in the menu to invoke the *View Creation Wizard*.

4.1 Computations

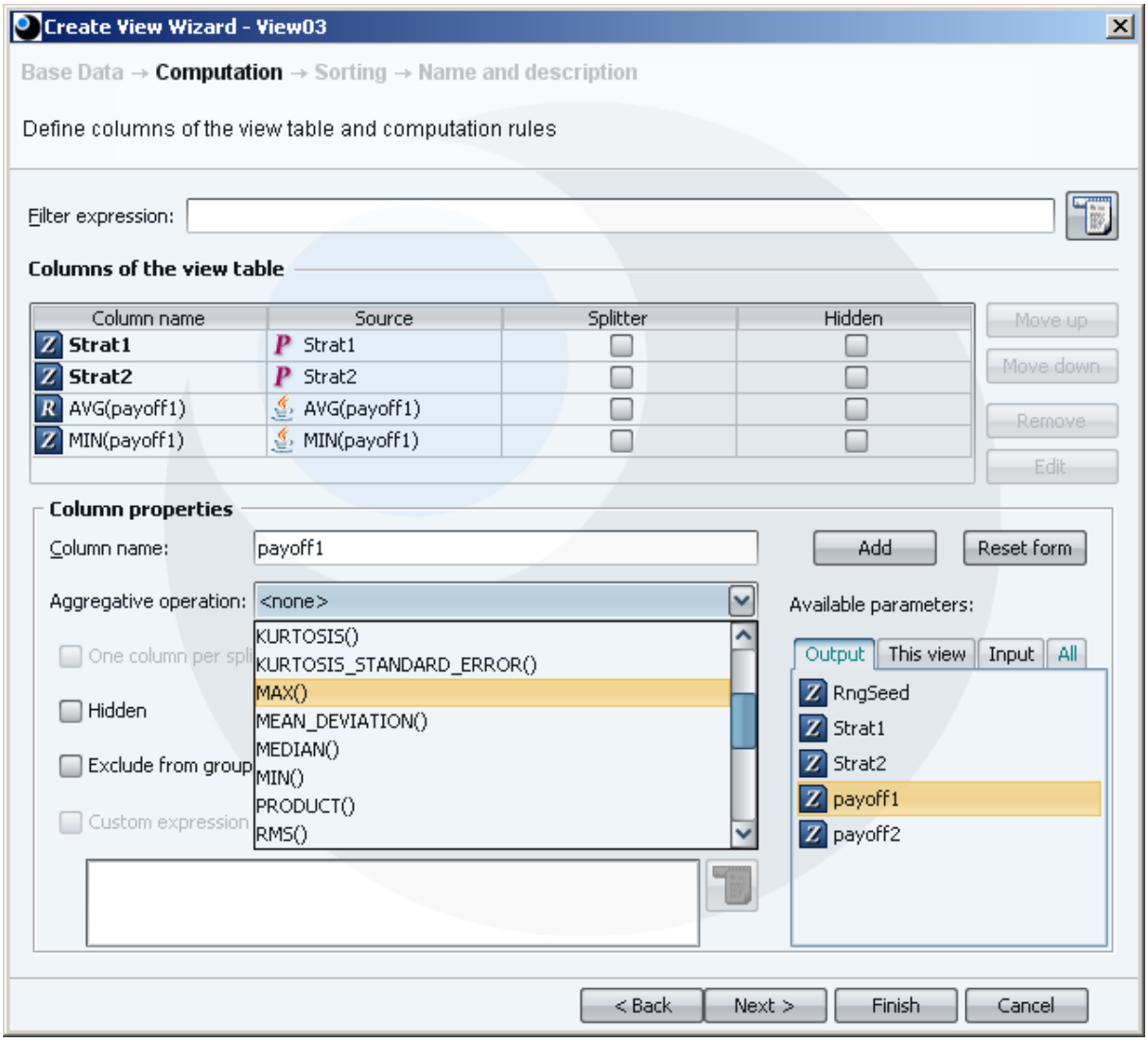


Figure 18 – View Creation Wizard

On the *Computation* page (see above) you can set the columns of the view table. Note that the parameters selected as output when importing are listed on the *Output* tab of the *Available parameters* list.

Add `Strat1` and `Strat2` by simply selecting them (two at a time if you wish) from the *Available parameters* list and pressing the *Add* button.

4.1.1 Aggregative Operations

Select `payoff1` and set *AVG()* in the *Aggregative operations* field and press *Add*. Due to the use of randomization in strategy 0 the model was executed fifteen times for every (`Strat1`, `Strat2`) pair thus the use of aggregative operations of the payoff values is advisable in order to analyze the data. Do the same with *MIN()* and *MAX()* operations for the `payoff1` parameter and repeat the whole thing with the `payoff2` values.

The program provides a default name for every column derived from the parameter it was created from. In our case it is useful to set a custom name for the payoff columns hence avoiding having columns named payoff1, payoff2, …, payoff6. In the example above the column names are made out of the name of the operation and the parameter it was done on.

4.1.2 Scripting

Now select *Custom expression or script* from the *Column properties*, press to declare the new column values as string and type in:

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

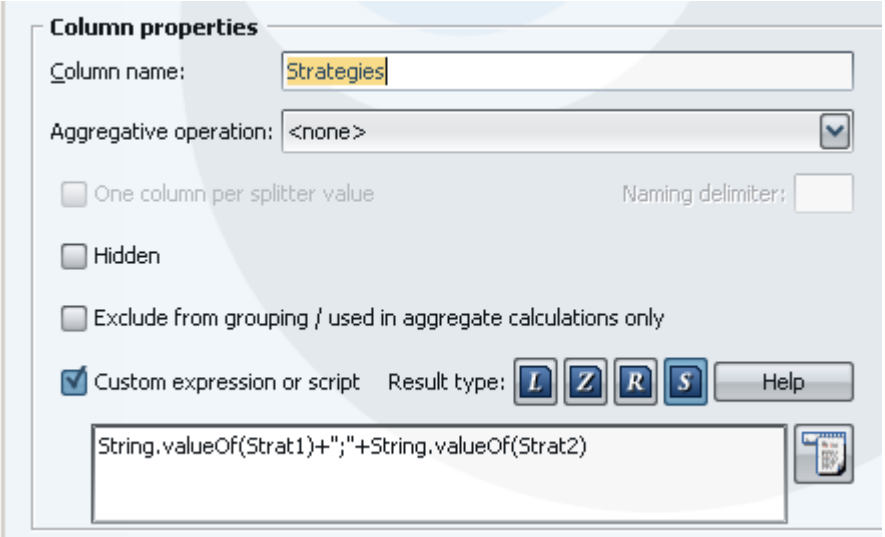


Figure 19 – Scripting

This little script creates the `Strat1;Strat2` values we will need later on when creating charts. Provide a name (`Strategies`) for the column and press *Add*.

If you wish to modify an already added column select it and use the *Edit* button or simply double click the column. Don't forget to press *Modify* when done with the changes. When done press *Next*.

4.2 Sorting

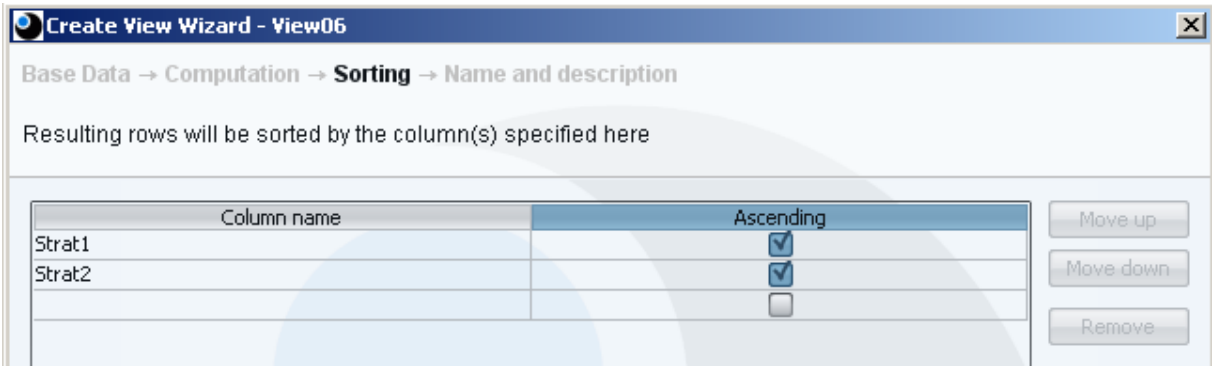


Figure 20 – Sorting

In order to create transparent view table sort the rows of the table by the columns `Strat1` and `Strat2`, both in *Ascending* order. When done press *Next* to continue.

4.3 View Name and Description

Figure 21 – Name and description

Give a unique name for the view – if there is already a view with the same name it will be overwritten – not longer than 64 characters and provide a description without limitations of length if you feel necessary. Press the *Finish* button when done thus adding the view table to the database. Now you can create charts in MEME or export the view table (*File/Export…*) as a CSV file.

5 Visualization

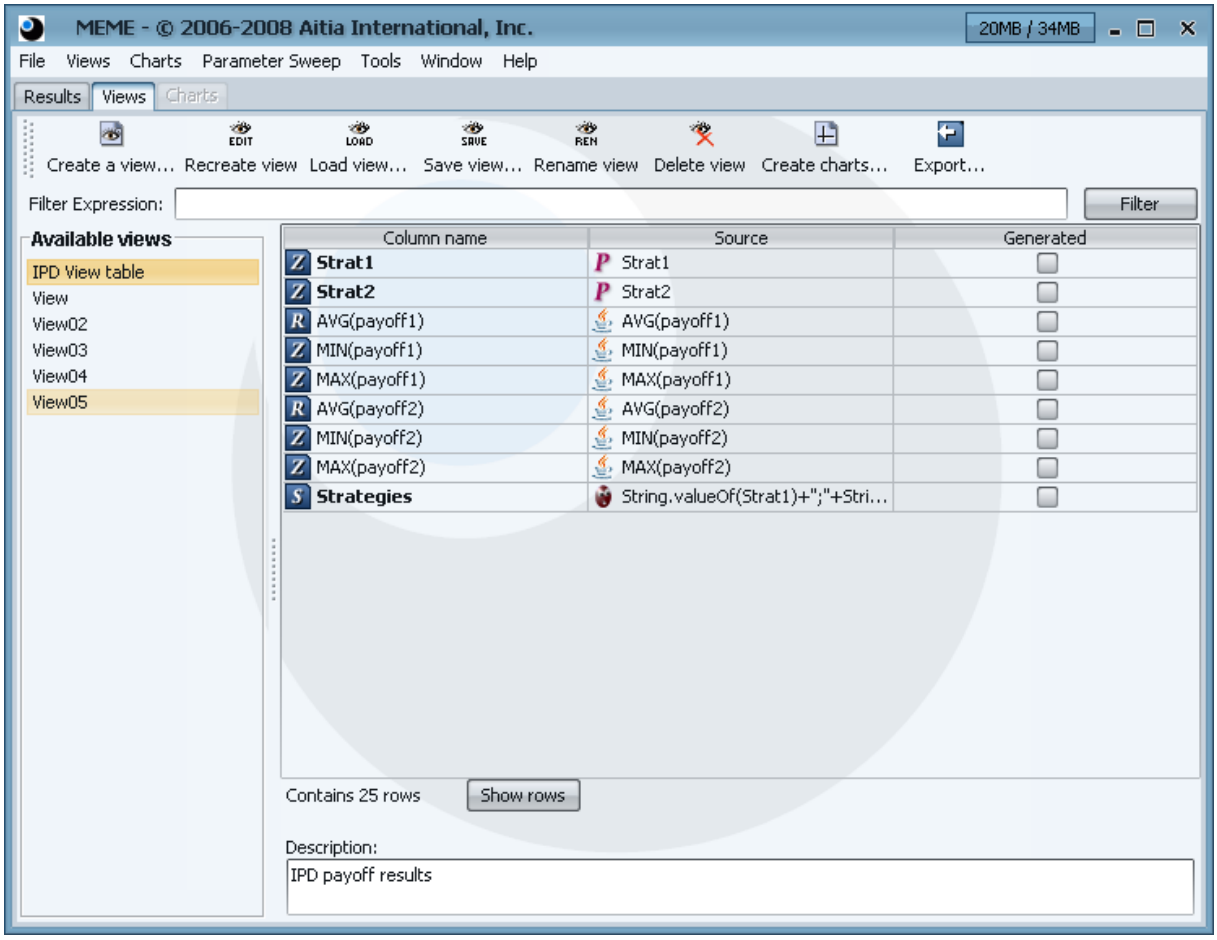


Figure 22 – Available views

The freshly created view table is now listed under *Available views*. The column names and properties are listed by default but you can observe the actual values in the table by pressing *Show rows* on the bottom of the screen.

#	Strat1	Strat2	AVG(payoff1)	MIN(payoff1)	MAX(payoff1)	AVG(payoff2)	MIN(payoff2)	MAX(payoff2)
0	0	0	5090.4	4616	5694	4910.733333	4350	5393
1	0	1	-15015.4	-15246	-14829	20020.533...	19772	20328
2	0	2	24984.6	24754	25171	-9979.466667	-10228	-9672
3	0	3	5006.2	4945	5082	5004.333333	4940	5082
4	0	4	4963	4426	5395	5036.733333	4604	5574
5	1	0	20020.533...	19772	20328	-15015.4	-15246	-14829
6	1	1	0	0	0	0	0	0
7	1	2	40000	40000	40000	-30000	-30000	-30000
8	1	3	4	4	4	-3	-3	-3

Figure 23 – Show rows

Having selected the *IPD View table* press

in the toolbar or select *Charts/Create charts…* item from the menu to switch MEME to chart creating mode. Select *Bar Chart* from the *Available chart types* menu and press *Create*.

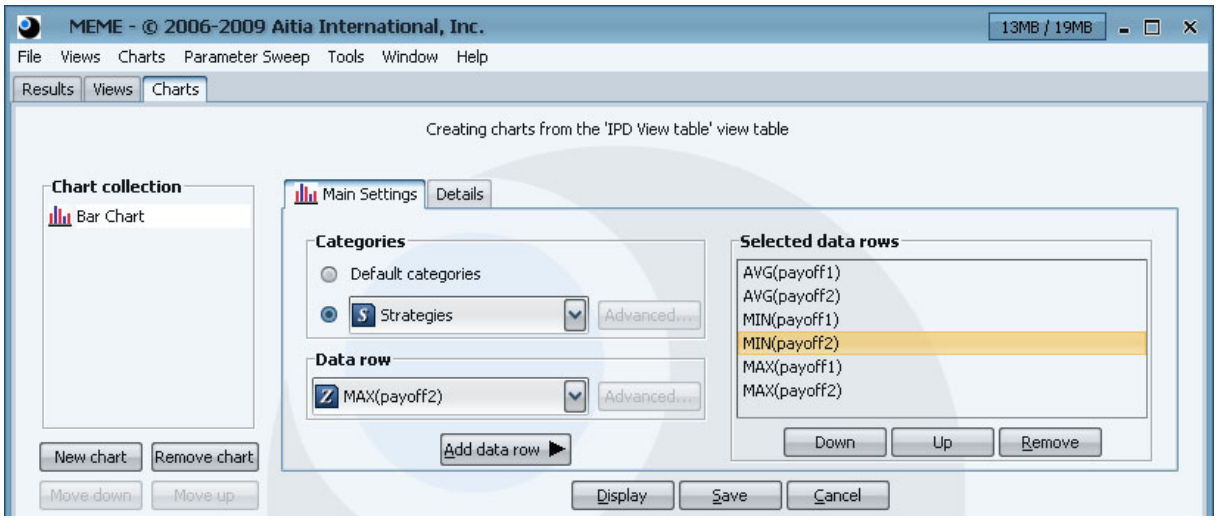


Figure 24 – Creating a Bar Chart

5.1 Creating a Bar Chart

On the *Main Settings* tab set `Strategies` as the *Categories* value and add `AVG(payoff1)`, `AVG(payoff2)`, `MIN(payoff1)`, …, `MAX(payoff2)` as data rows. It is advisable to keep this order to be able to compare the bars visually. Provide a *Title* and a *Subtitle* on the *Details* tabs and press *Display* button to see the chart.

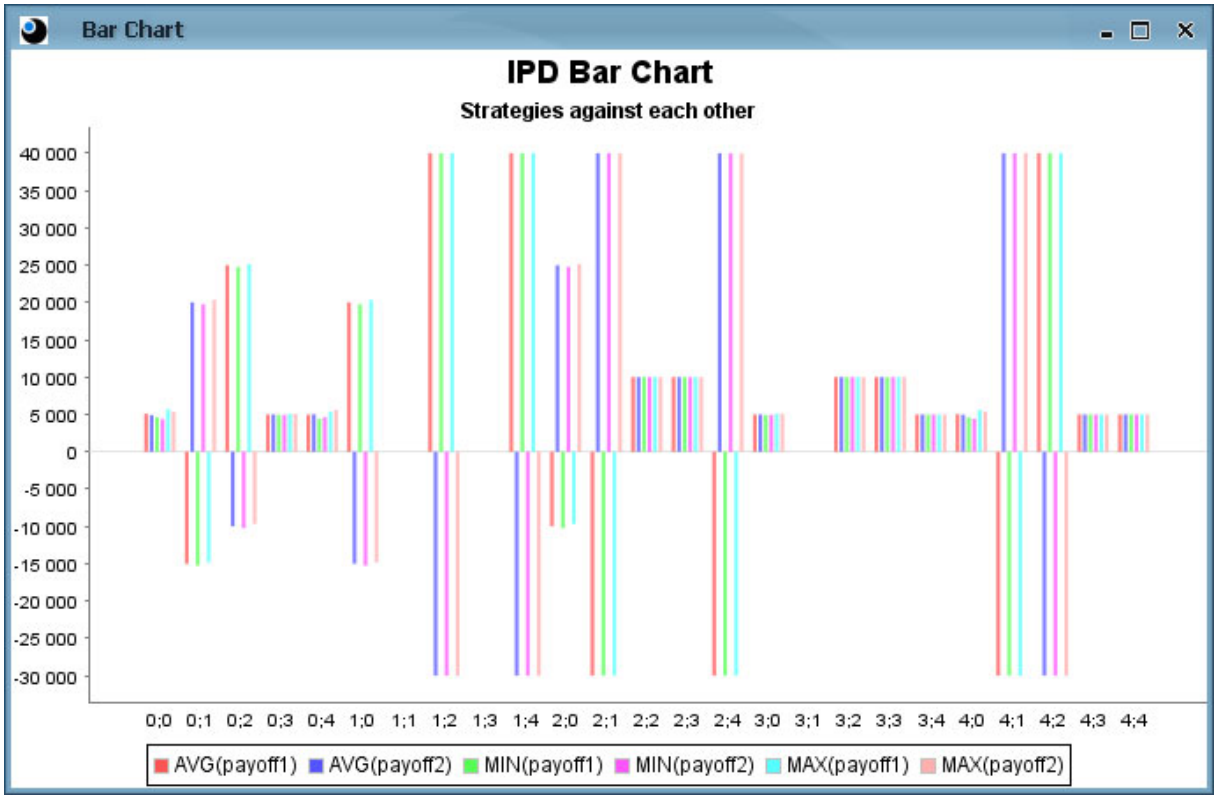


Figure 25 – Bar chart

As you can see from the resulting bar chart, the average, minimum and maximum values do not differ considerably for the given payoff in one category it is possible to analyze the results picking only one data row for each player's payoff.

5.2 Creating a Line Chart

Press *New Chart* under the *Chart collection* field, choose *XY Line Chart* from the *Available chart types* list and press *Create*. In the *X values* field set `Strategies`, select `AVG(payoff1)`

for *Y values* and press the *Add*▶ button. Repeat this for `AVG(payoff2)`, optionally provide a title, a subtitle, and axis labels for the chart.

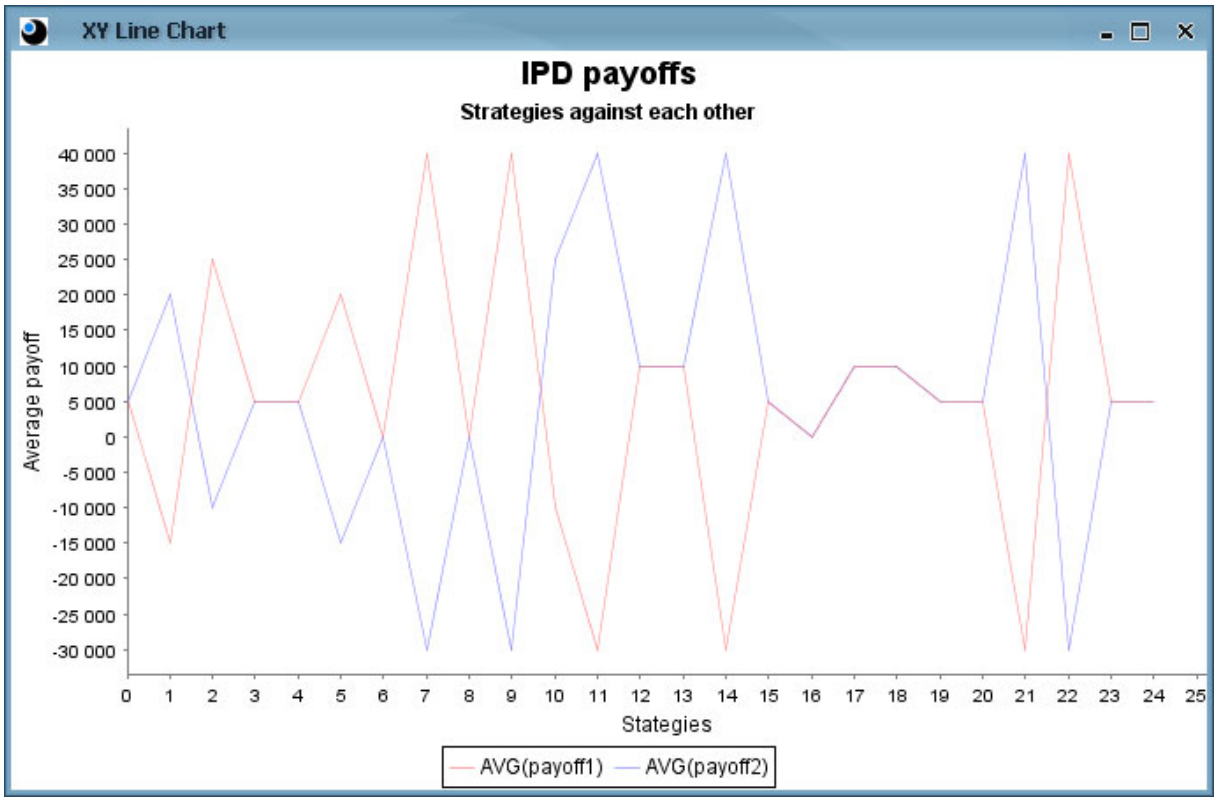


Figure 26 – XY Line chart

5.3 Zooming

To observe the more interesting parts of the chart hold down the left mouse button and drag it right and down, selecting the area to be zoomed in on with a rectangle.

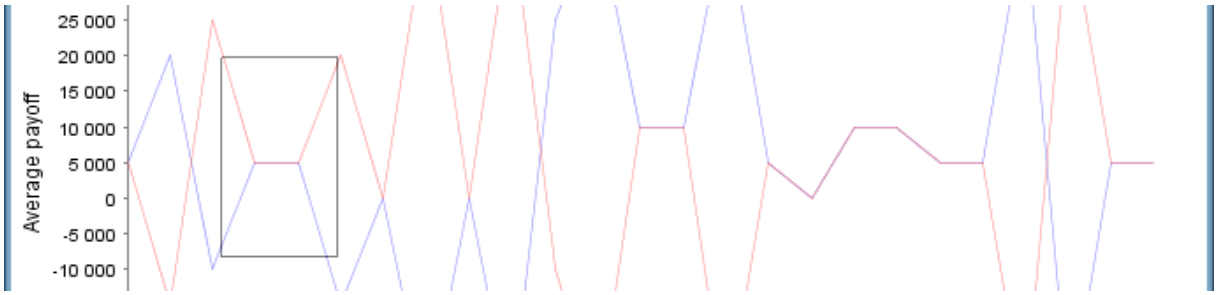


Figure 27 – Select area

After several repetitions it is revealed that strategy 0;4 has somewhat higher `payoff2` than `payoff1` averages in our results. To zoom out simply drag the mouse left and top while holding down the mouse button.

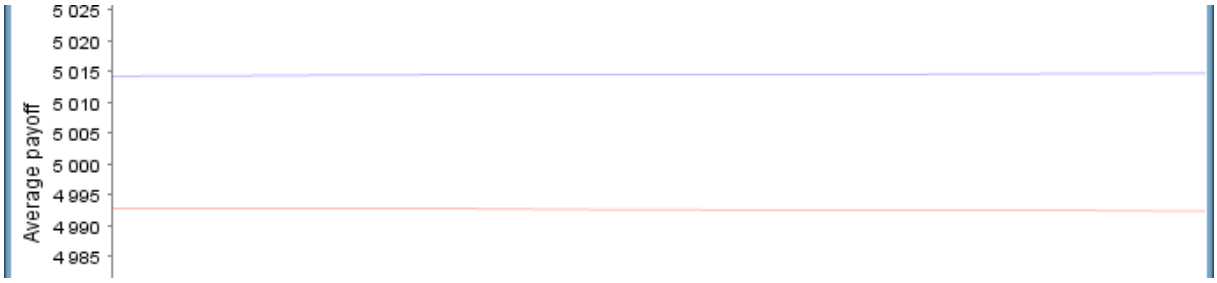
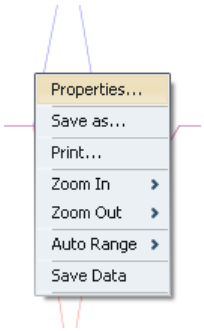


Figure 28 – Small difference

5.4 Exporting and formatting charts

By right clicking on the chart a context menu pops up. You can control the zooming (*Zoom In/Out*) from here, export (*Save as…*) the chart as an EPS or a PNG file, export (*Save Data*) the data shown on the chart as a CSV file, print the chart (*Print…*) and also format its appearance (*Properties…*). Through *Properties* titles and labels can be edited or changed, fonts, border and background colors can be set and axis properties can be customized.



Just make sure you get something nicer than we did in the end:

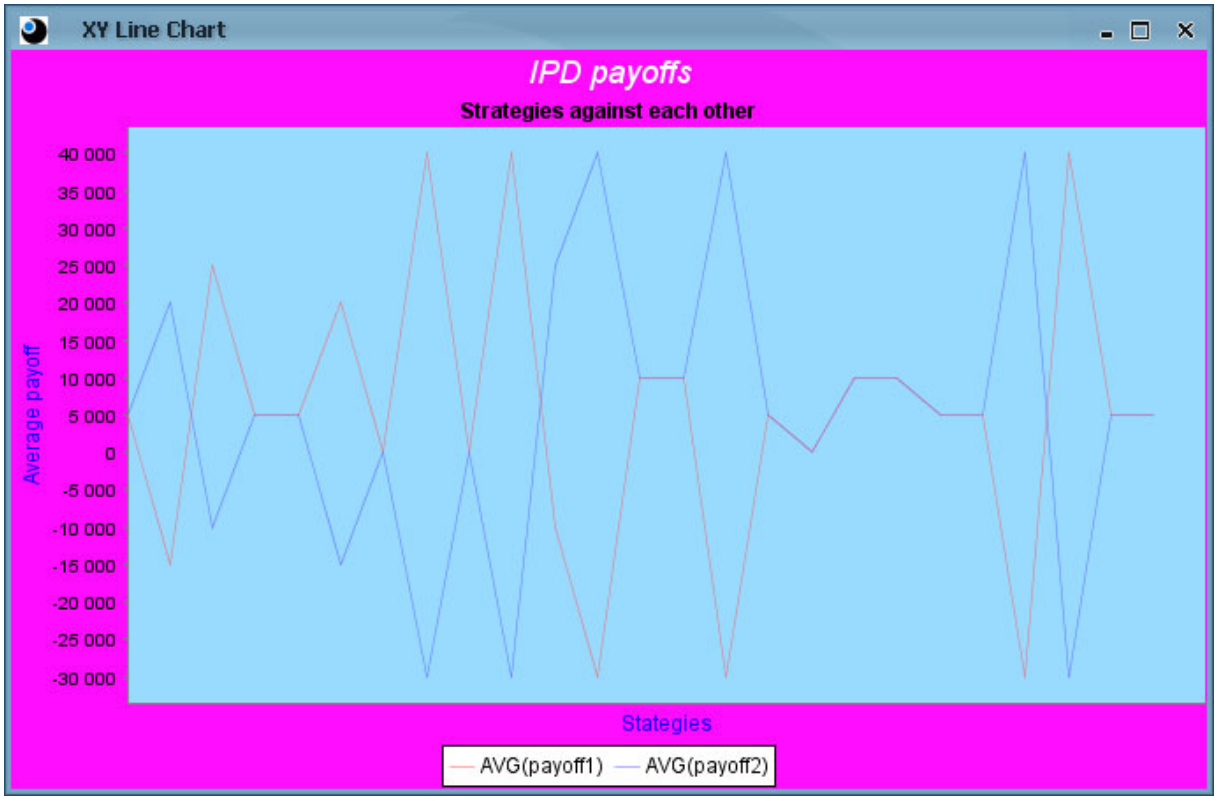


Figure 29 – This one might not win the design award

5.5 Default Appearances

On the *Details* tab different preset appearance can be chosen for charts. There are four default styles to choose from but you can create new ones. For example, the *Basic black-and-white* is appropriate for charts intended to appear in black and white publications.

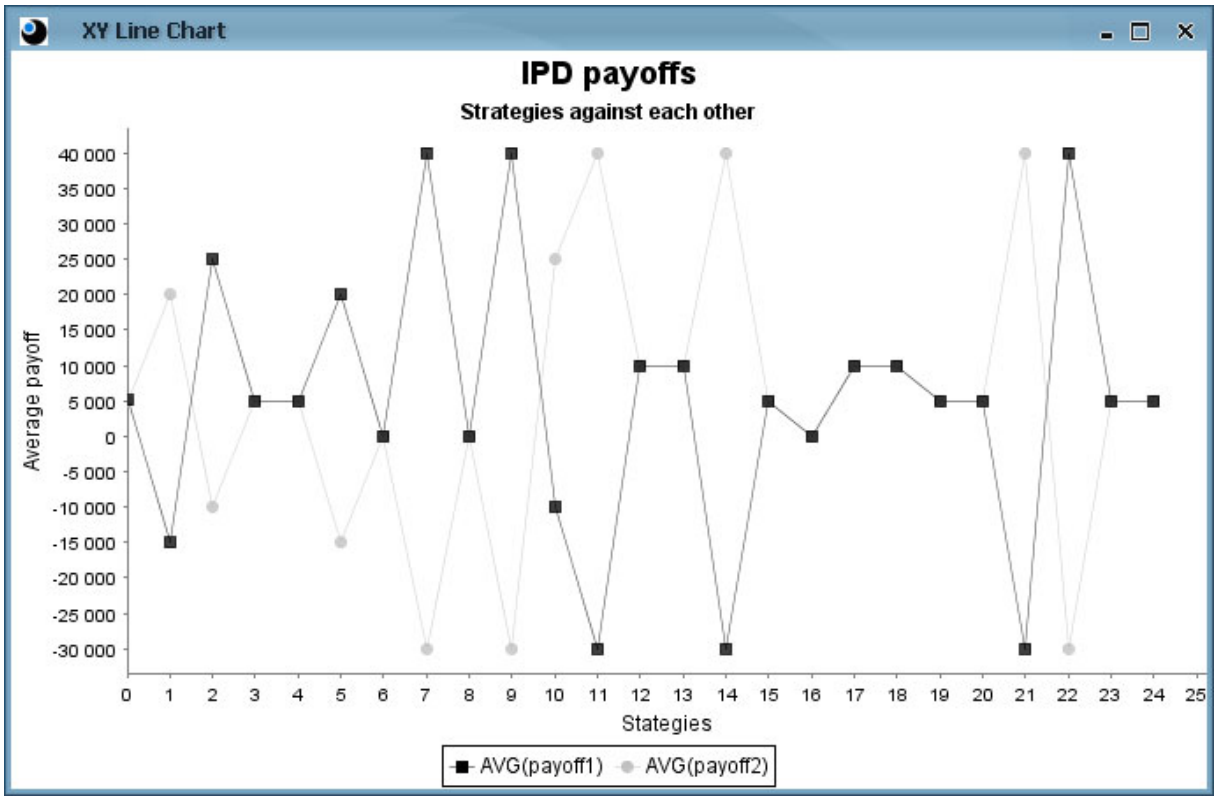


Figure 30 – Basic black-and-white appearance

5.6 Saving the Chart Collection

When finished, save the chart collection by pressing *Save*. MEME stores the data describing the chart in XML format at a user specified location. When opening (*Charts/Open chart…*) the collection again make sure, that the view table it was created from is still in the database.

6 Export

Go back to the *Views* tab. Having selected the *IPD View table*, press the button in the toolbar or select *File/Export…* from the menu. In both cases select *CSV file* from the popup menu to reach the *CSV Export Settings* dialogue appears (see below).

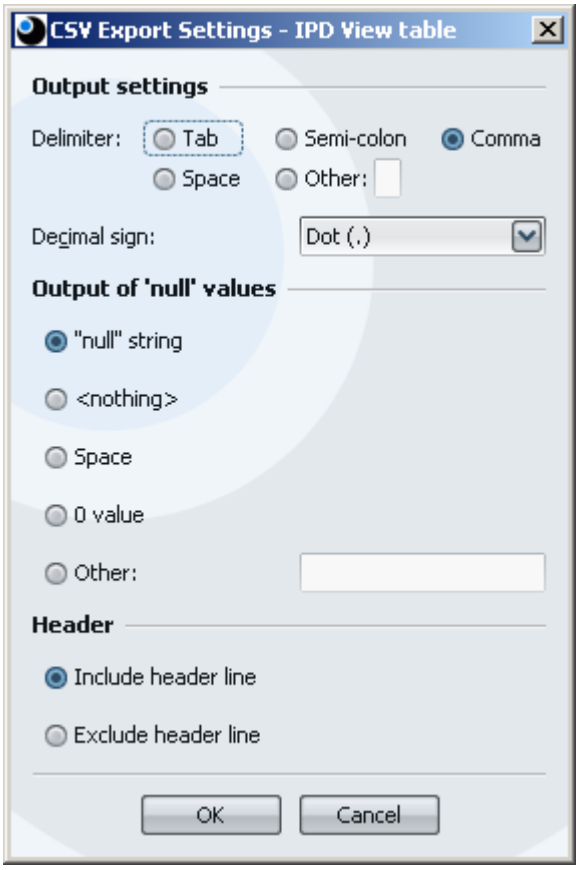


Figure 31 – CSV Export Settings

If the default settings are used the values will be separated by commas and the decimal sign will be a dot. There are no empty fields in this view table, hence null value handling is not important here, but as illustrated above, there are five options for that. By default column names (header row) are included but this can also be switched off.

Upon pressing *OK* the file selection dialogue appears where the name and location of the exported file can be assigned. Press *Save* to export.

7 Notes

MEME is a tool for dealing with batch runs of simulations and the data produced. It allows the user to store and organize the raw data in database(s), and to create distillated (computed) tables which can be visualized on charts.

Note that you can also start the tutorial from the Importing the Results chapter because the RepastJ result file – created by the *Parameter Sweep Wizard* – is included with the MEME installer. If you want to start from there press the

button in the toolbar or select the *File/Import…* from the menu. In both cases select *RepastJ result file* from the popup menu and then select the *IPD.out* file in the file dialogue displayed. The default destination of the file is *C:\Program Files\MASS\MEME\documents\ prisoners_demo\IPD.out*. If you chose a different installation folder it is implicitly *…\documents\ prisoners_demo\IPD.out*.

8 References

[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

9 Appendix

9.1 PrisonersAgent.java

```
package demo.prisoners;

import uchicago.src.sim.util.Random;

/** An Agent class representing a player with none or one-step memory in
Iterated Prisoner's Dilemma game. */
public class PrisonersAgent {

    /** Strategy: Random */
    public static final int RND = 0;
    /** Strategy: Allways defect */
    public static final int ALLD = 1;
    /** Strategy: Allways cooperate */
    public static final int ALLC = 2;
    /** Strategy: Tit-for-tat */
    public static final int TFT = 3;
    /** Strategy: Anti Tit-for-tat */
    public static final int ATFT = 4;

    /** The players current strategy. */
    protected int strategy;
    /** The last step of the other player. */
    protected boolean enemyLast;

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

    /** Memorizes the last step of the other player. */
    public void setEnemyLast(boolean b) {
           enemyLast = b;
    }

    /** Returns true if cooperates. */
    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;
    }
}
```

9.2 PrisonersModel.java

```
package demo.prisoners;

import uchicago.src.sim.engine.*;

/**Iterated Prisoner's Dilemma game model. */
public class PrisonersModel extends SimpleModel {

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

```
    }

    /** The winner's payoff. */
    protected int winner;
    /** Returns winner's payoff. */
    public int getWinner() { return winner; }
    /** Sets winner's payoff. */
    public void setWinner(int winner) { this.winner=winner; }
    /** The loser’s payoff. */
    protected int looser;
    /** Returns looser's payoff. */
    public int getLooser() { return looser; }
    /** Sets looser's payoff. */
    public void setLooser(int looser) { this.looser=looser; }
    /** Both players' payoffs if they're cooperate. */
    protected int both;
    /** Returns payoffs if both cooperate. */
    public int getBoth() { return both; }
    /** Sets payoff if both cooperate. */
    public void setBoth(int both) { this.both=both; }
    /** Both players' payoffs if neither cooperates. */
    protected int neither;
    /** Returns payoffs if neither cooperates. */
    public int getNeither() { return neither; }
    /** Sets payoffs if neither cooperates. */
    public void setNeither(int neither) { this.neither=neither; }
    /** The 1st player’s strategy. */
    protected int strat1;
    /** Sets 1st player’s strategy. */
    public int getStrat1() { return strat1; }
    /** Returns the 1st player’s strategy. */
    public void setStrat1(int strat1) { this.strat1=strat1; }
    /** The 2nd player’s strategy. */
    protected int strat2;
    /** Sets the 2nd player’s strategy. */
    public int getStrat2() { return strat2; }
    /** Returns the 2nd player’s strategy. */
    public void setStrat2(int strat2) { this.strat2=strat2; }

    /** 1st player's payoff.*/
    protected int payoff1;
    public void setPayoff1(int i) { payoff1 = i; }
    public int getPayoff1() { return payoff1; }
    /** 2nd player’s payoff. */
    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"};
        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();
    }

}
```

9.3 PrisonersModelGUI.java

```
package demo.prisoners;

import uchicago.src.sim.analysis.*;

/** Iterated Prisoner's Dilemma game for gui mode*/
public class PrisonersModelGUI extends PrisonersModel {

    /** Graph in the GUI mode. */
    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;
          looser=-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();
    }
}
```