Introduction to IPDNet Simulator

Juan C. Burguillo

J.C.Burguillo@uvigo.es

Version 1.1 (May 2022)

Table of Contents

1. Introducing the IPDNet simulator

IPDNet is a free to use open-source Java-based software for fast prototyping, developed by Juan C. Burguillo and licensed under the GNU Lesser General Public License (LGPL). This software is a spin-off from the CellNet simulation framework [1], that was used for the simulations presented in [2].

1.1 IPDNet Basic Features

IPDNet works in two modes: i) using a graphical user interface (GUI) for doing micro-simulations or ii) using a batch mode for doing macro-simulations. IPDNet also provides support for:

- Visualizing the whole set of cells and their state along each simulation iteration.
- Visualizing the simulation results in real time. A set of graphical windows is provided for every relevant simulation result.

- Importing network data to reuse particular network structures to run experiments.
- Exporting network data, to save a particular network structure. The format used for the exported files is compatible with popular network analysers such as Pajek or Gephi.

1.2 IPDNet Utilities

IPDNet allows a hands-on approach to simulating the IPD over the nets. Using IPDNet you can:

- **Run Micro-simulations**: most of the simulations can be directly run in a one-shot mode, selecting them directly from the main menu of the simulator. The different game simulation parameters can be reviewed and modified from the experiment option window, or from a general configuration window. No programming skills are needed to run the simulator in this mode.
- **Run Macro-simulations**: having very basic general programming skills allows the reader to configure some batch files to execute a set of experiments, involving multiple runs, to analyze the average results provided by such set of game simulations.
- **Modify the algorithms**: readers having standard Java programming skills can redesign their own algorithms, and then test the behaviors obtained by performing new experiments. For this, new algorithms can be created from scratch, or more commonly, the algorithm files already included can be inherited and used as templates. IPDNet code includes support for generating different types of complex topologies, using several machine learning techniques, performing evolutionary meta-decisions, generating real-time visualizations and interacting with external network analyzers.

1.3 A graphical description of the graphical tools provided by IPDNet

Fig. 1 provides a snapshot of the IPDNet main window together with some of the windows used for displaying graphical data corresponding to a particular game, in this case the Coalition Iterated Prisoners Dilemma [/REF].

On the left side of the figure appears the main window that provides a menu bar (described in Figs. 2-6), a control section on the left side with several buttons, a status bar with information at the bottom of the window, and the main grid full of coloured cells; which can be linked among them either spatially (as a lattice) or using any complex topology (small-world, scale free, random network, etc.).

On the right side of Fig. 1 appear four smaller windows describing some graphical data corresponding to the game under run, in this case: the global profit (social welfare), the average tax per cell, the profit per type of cell and finally at the right-bottom the tax histogram for the whole number of cells. The main window contains a menu bar with several menus described in next figures.

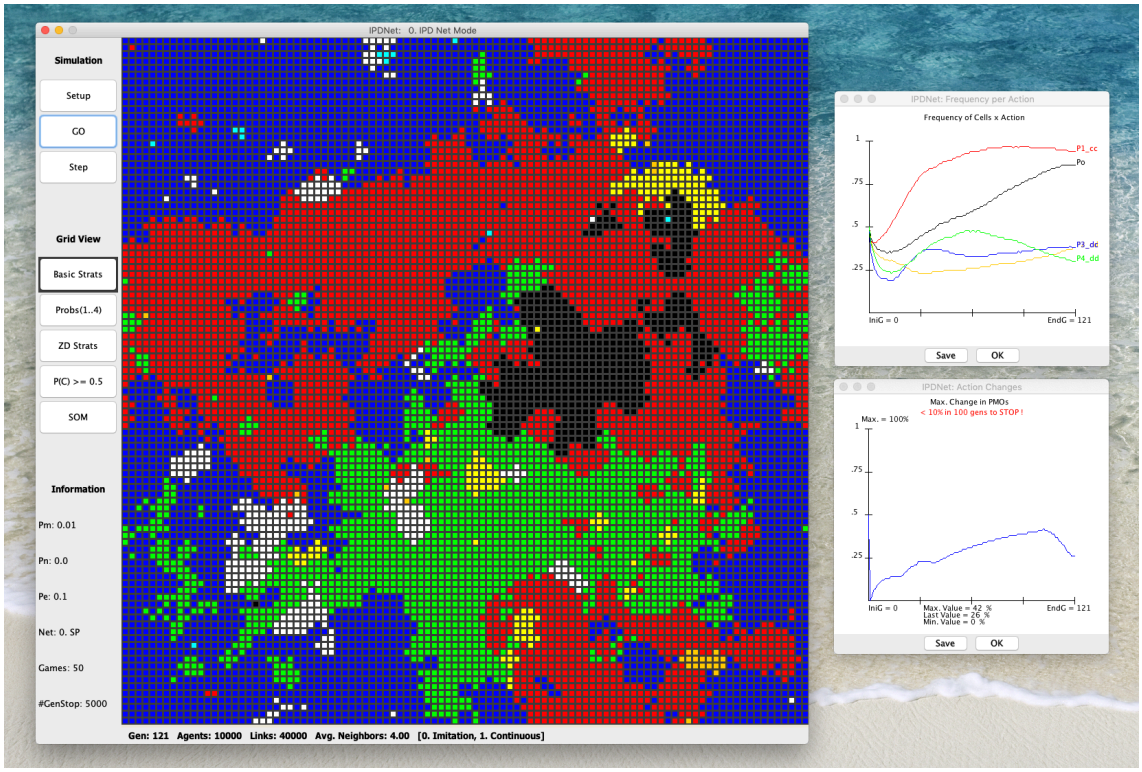

Fig. 1: A snapshot of the IPDNet simulator and the graphical data windows.

Fig. 2 presents the File menu with several items available to load/save the simulator state and topology. Besides it also allows to import/export the network topology to an external file in a format to be processed by external network tools (Pajek[1], Gephi[2], etc.).

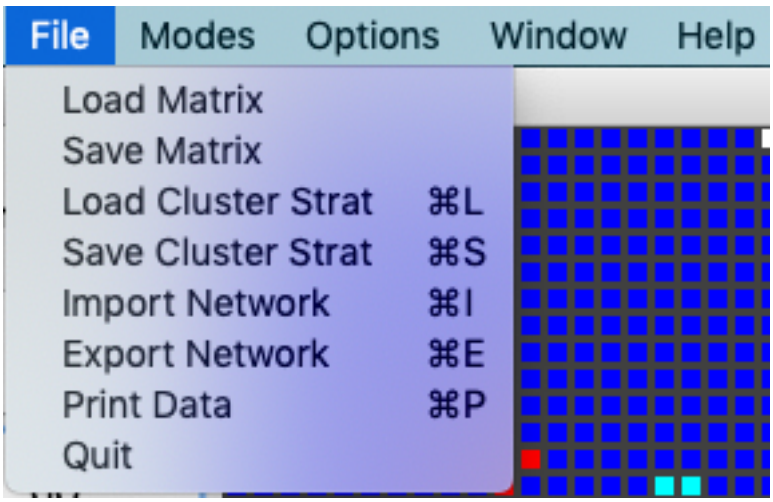

Fig. 2: The menu bar and the File menu

Fig. 3 presents the items available from the Games menu, which allows to execute different modes described in the Sect. 2 of this document.

[1] http://vlado.fmf.uni-lj.si/pub/networks/pajek/
[2] https://gephi.org

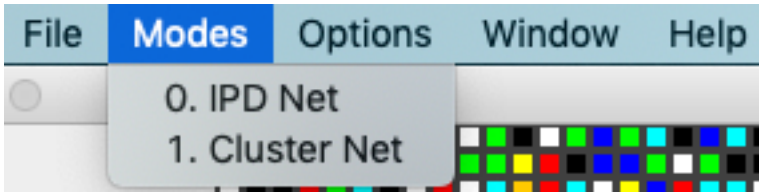

Fig. 3: The menu bar and the Modes menu

Fig. 4 presents the Options menu, which provides access to the general configuration, the network parameters and the payoff matrix. These are the general configuration parameters that appear in independent windows to be configured. Besides, the Options submenu also gives access to the particular parameters used to configure each individual game.

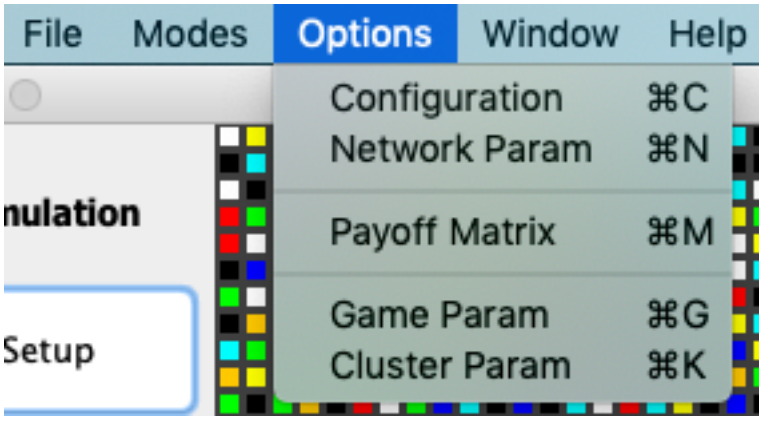

Fig. 4: The Options menu

Fig. 5 displays the Window menu, which gives access to several data graphical windows to present dynamically and on real time the data generated by the different simulator games. Every game normally uses a particular set of graphical windows, which are remarked with boldface letter.

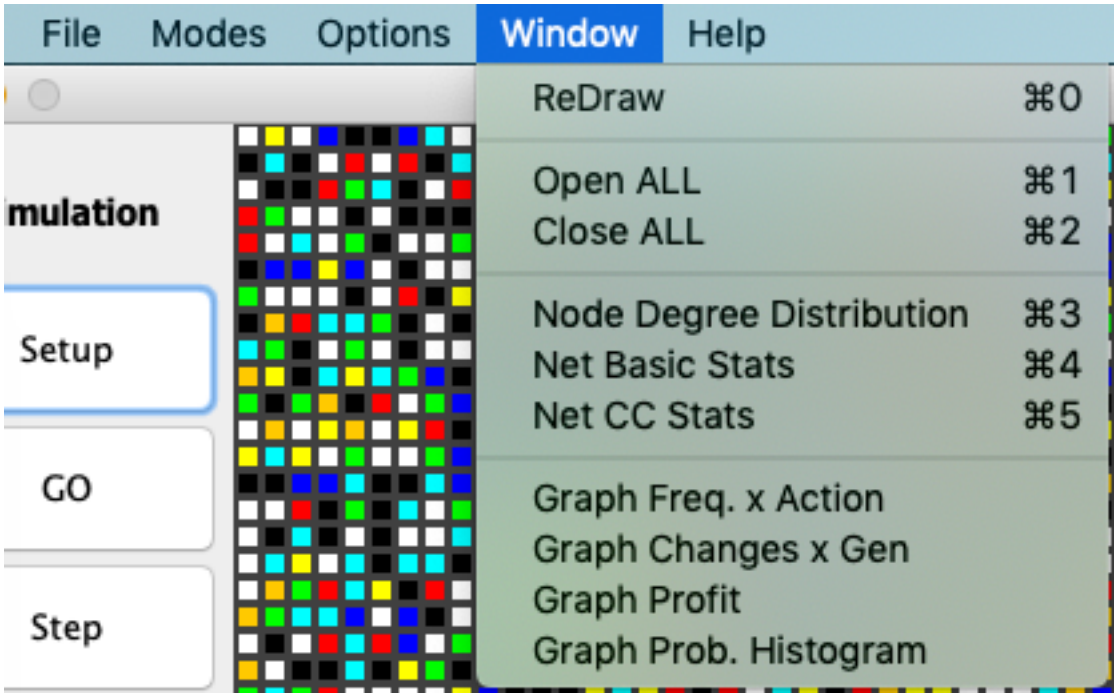

Fig. 5: The Window menu

Finally, Fig. 6 presents the help menu which enables/disables the verbose mode, gives access to the IPDNet Help item and the About item, which gives contact information and the present version of the game.

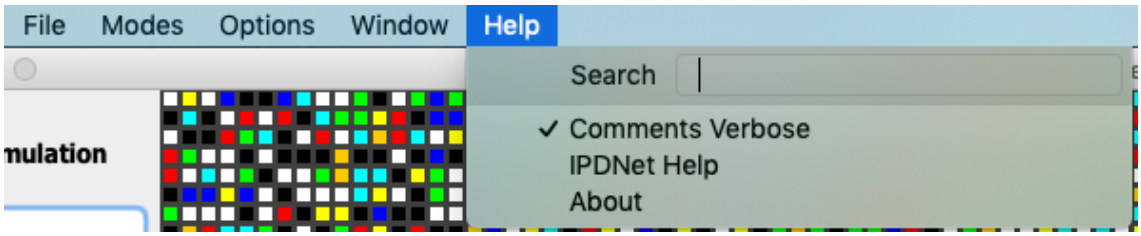

Fig. 6: The Help menu

The verbose mode outputs on the console information about the simulation depending on the simulation mode. There are two different types of information:

a) If the user is simulating on the IPD Net mode, then when the simulation is stopped it outputs some statistics. Fig. 7 shows a display of such information that first outputs the simulation parameters, followed by the more popular strategies at that simulation generation. As we can see in the figure, in the generation 122 there were only 15 survival strategies (the output is limited to the top 32 popular strategies). For each strategy, the system outputs the PMO, the number of agents using it (relative and cumulative values in the parenthesis), then the name of the strategy (exact one marked with symbol "==" and the closer one marked with "->"). Then a set of values obtained for each strategy in the last generation are printed, corresponding to the average payoff obtained the agents using such strategy, the cooperation rate, and finally the frequencies for

having CC, CD, DC and DD; respectively. Afterwards, the output shows the number of agents included in the table (which can be less than the whole population), the criteria used to stop the simulation, and the average PMO values of the whole population.

Finally it displays a table that includes the dynamics of state change in the generation it stopped. Such table shows the frequency of movements from a particular state to another. For instance in Fig. 7 almost all the agents that start from CC stay in CC, which is coherent with the popular strategies (most of them related with *c.Spiteful*).

```
Mode: 0. IPD Net                        Agents: 10000                          GamesxEncounter:50
ChangeType: 0. Imitation                ChangeModel: 0. Not the Best           ProbModel: 0. Discrete
NetType: 0. SP                          Avg. Neighborhood: 4.00                Gen: 122
Pn:0.0   Pe:0.1   Pm:0.01   Pca:1.00   Psr:0.10   OwnWeight:0.5   Ipmo:0.5

Popularity of the Top 15 strats (P0,P1,P2,P3,P4) == (P0,Pcc,Pcd,Pdc,Pdd) == (P0,Pr,Ps,Pt,Pp)

Rank [P0 |P1 |P2 |P3 |P4 ]  Pop (Rel,Tot)      Name     |           Avg.Payoff  C.Rate  Fcc     Fcd     Fdc     Fdd
01.  [1.0|1.0|0.5|0.0|0.0]  3064 (30%,30%)  -> c.Spiteful (Extortioner) 3.00    1.00    1.00    0.00    0.00    0.00
02.  [1.0|1.0|0.0|0.0|0.0]  2080 (20%,50%)  == c.Spiteful (Extortioner) 3.00    1.00    1.00    0.00    0.00    0.00
03.  [1.0|1.0|0.0|0.0|0.5]  1525 (15%,65%)  -> c.Spiteful               3.00    1.00    1.00    0.00    0.00    0.00
04.  [1.0|1.0|0.0|0.5|0.0]  1422 (14%,79%)  -> c.Spiteful               3.00    1.00    1.00    0.00    0.00    0.00
05.  [1.0|1.0|0.0|1.0|0.0]  903 (9%,88%)    == c.TFT (Extortioner)      3.00    1.00    1.00    0.00    0.00    0.00
06.  [1.0|1.0|0.0|0.5|0.5]  592 (5%,93%)    -> c.Spiteful               3.00    1.00    1.00    0.00    0.00    0.00
07.  [1.0|1.0|0.5|0.0|0.5]  289 (2%,95%)    -> c.Spiteful               3.00    1.00    1.00    0.00    0.00    0.00
08.  [1.0|1.0|0.0|0.0|1.0]  73 (0%,95%)     == c.Pavlov                 3.00    1.00    1.00    0.00    0.00    0.00
09.  [1.0|1.0|0.0|0.5|1.0]  22 (0%,95%)     -> c.Pavlov                 3.00    1.00    1.00    0.00    0.00    0.00
10.  [1.0|1.0|1.0|0.0|0.0]  12 (0%,95%)     -> c.Spiteful (Extortioner) 3.00    1.00    1.00    0.00    0.00    0.00
11.  [1.0|1.0|0.5|0.5|0.0]  11 (0%,95%)     -> c.Spiteful (ZD)          3.00    1.00    1.00    0.00    0.00    0.00
12.  [1.0|1.0|0.5|0.5|0.5]  3 (0%,95%)      -> c.Spiteful               3.00    1.00    1.00    0.00    0.00    0.00
13.  [1.0|1.0|1.0|0.5|0.5]  2 (0%,95%)      -> All C                    3.00    1.00    1.00    0.00    0.00    0.00
14.  [1.0|1.0|0.0|1.0|0.5]  1 (0%,95%)      -> c.TFT                    3.00    1.00    1.00    0.00    0.00    0.00
15.  [1.0|1.0|1.0|0.0|0.5]  1 (0%,95%)      -> c.Spiteful               3.00    1.00    1.00    0.00    0.00    0.00

Num. Agents printed: 10000 (100%)          MaxProbChange: 10%

Avg. Probs.(±St.Dev.):   P(0)=100%(±0%)   P(1)=100%(±0%)   P(2)=16%(±11%)   P(3)=19%(±16%)   P(4)=13%(±11%)

Frequency of State Dynamics per Generation: 122
        CC    CD    DC    DD
CC ->  1.00  0.00  0.00  0.00  =  1.00
CD ->  0.00  0.00  0.00  0.00  =  0.00
DC ->  0.00  0.00  0.00  0.00  =  0.00
DD ->  0.00  0.00  0.00  0.00  =  0.00
-------------------------------------
Prob:  1.00  0.00  0.00  0.00
```

Fig. 7: Output on verbose mode corresponding to the IPD Net simulation mode

b) If the user is simulating on the Cluster Net mode, then it also outputs information concerning the training process of the SOM-QLA architecture. Fig. 8 shows a display of such information divided in two parts: the one corresponding the SOM training and the one corresponding the QLA training; being both are rather intuitive. Particularly, for the QLA one displays on the first line: the BMU selected (determining the QLA state), the total quality value stored (adding all the QValues), the number of strategies used in that QLA state (a value between 6 and 10), and the times that state has been useful compared with the number of times it has been used. Afterwards it gives information about every strategy stored in that state, its Q-value, its PMO, the ration between the successes vs. the number of times used, and finally the strategy name (only for the static ones).

```
-------------------- SOM --------------------
dmInput:          0.93389  0.98886  0.44490  0.31476  0.12544
dBMU_pre:         0.93671  0.99603  0.44645  0.31577  0.12586
dBMU_post:        0.93530  0.99245  0.44568  0.31526  0.12565

-------------------- QLA --------------------
BMU: [0,2]   TotQValue: 10.12   iNumStratsPMO: 9   Success/Times: (6/9)
Strat: 0   QValue: 1.000   PMOs:  0.0  1.0  0.0  0.0  0.0   Success/Times: (0/0)        d.Spiteful
Strat: 1   QValue: 1.815   PMOs:  1.0  1.0  0.0  0.0  0.0   Success/Times: (3/4)        c.Spiteful
Strat: 2   QValue: 0.500   PMOs:  0.0  1.0  0.0  1.0  0.0   Success/Times: (0/1)        d.TFT
Strat: 3   QValue: 1.000   PMOs:  1.0  1.0  0.0  1.0  0.0   Success/Times: (0/0)        c.TFT
Strat: 4   QValue: 1.788   PMOs:  0.0  1.0  0.0  0.0  1.0   Success/Times: (2/2)        d.Pavlov
Strat: 5   QValue: 1.000   PMOs:  1.0  1.0  0.0  0.0  1.0   Success/Times: (0/0)        c.Pavlov
Strat: 6   QValue: 1.000   PMOs:  0.0  0.0  0.0  0.0  0.0   Success/Times: (0/0)
Strat: 7   QValue: 1.000   PMOs:  1.0  1.0  1.0  1.0  1.0   Success/Times: (0/0)
Strat: 8   QValue: 1.019   PMOs:  0.9  1.0  0.3  0.9  0.0   Success/Times: (1/2) <
Strat: 9   QValue: 0.000   PMOs:  0.0  0.0  0.0  0.0  0.0   Success/Times: (0/0)
TOTAL Success/Times: (268/636) = 42%
---> Strat selected: 8     iNumClusterCells: 435 (-5)
```

Fig. 8: Output on verbose mode corresponding to the cluster brain training

The rest of this short introduction describes in more detail these graphical elements, together with the API provided to manage and create IPDNet games, to display graphical data and to manage batch executions or network topologies.

2. IPDNet simulation modes

IPDNet provides a user-friendly access to two different in-built modes, ready to be executed with a single mouse click. These modes are shortly described next:

- **IPD Net**: Iterated Prisoner's Dilemma (IPD) with Probabilistic Memory One strategies (PMO) over the Network: this is a version of the IPD over Networks [2,3] based also on the works [4,5,6,7], that uses PMO strategies to describe if a player plays cooperatively after playing CC, CD, DC or DD in the previous round. The game has four possible displays to describe the strategies playing the game.

- **Cluster Net**: this is an extension of the IPD Net scenario where, after stability, a cluster seed is introduced in the population to start a cluster. This cluster joins to its team any other agent that imitates its behaviour. The cluster uses a machine learning architecture, based on Self-Organising Maps (SOM), Quality Learning Automata (QLA) and Evolutionary Game Theory (EGT) to play adaptive strategies in order to spread through the population and eventually dominate it.

3. A description of the API

This section describes the Application Programmer's Interface (API) of IPDNet. The framework has been programmed in Java using Eclipse, and should be easily open as an Eclipse project. Once opened, we can observe several source packages in Java physically organized in the disk as file folders.

3.1 IPDNet packages and main files

The contents of the *src* folder are the next:

a) Basic files:

bIPDNet.java: is a Java file used to run batch executions of any game.

IPDNet.html: is a HTML page that allows to run IPDNet as a Java applet.

IPDNet.java: is the main Java file of the simulator.

readme.txt: provides this basic description of files and packages.

b) Packages:

./file: IO files for interacting with external tools.

./lib: .jar file library for managing window appearance.

./games: general files for creating games (inheriting from Game.java) and cells (inheriting from Cell.java) to be simulated.

./window: information, parameter and windows used by the simulator.

c) Relevant files in *./game* package:

./games/Cell.java: code that should be inherited by each cell used in a game.

./games/Game.java: code that should be inherited by each game.

Inside this package we have a set of sub-packages containing the files used to simulate the simulator modes described in Sect. 2.

d) Relevant files in *./window* package:

./window/MainWindow.java: code for the main window of the simulator, that contains the menus, buttons and the rest of the graphical elements.

./window/Dlg_NAME.java: these set of files are used to manage parameters or to display information in several windows.

./window/Visor_NAME.java: these set of files are used for displaying graphics or the main simulator grid (*VisorWorld.java*).

3.2 Game, Cells and Windows

This subsection describes in detail some of the basic files (with commented code) needed to create new games and simulations.

a) *The file Game.java*: all games should extend this file, located in the *./games* package, which contains a basis set of attributes and methods that support de development of new games. The main methods, concerning the simulation,

appear at the end of the file. There we can find methods to create different network topologies among the cells (spatial, small-world, scale-free, random). We also find the *vRunLoop()* key method, which is the basic cycle performed in any game, and called from the *MainWindow.java* file (that manages the game thread). At the end we also find the method *vSetGraphicValues()* that stores data values generated by the simulation in appropriated vectors for graphical displaying. There is also a file named *GameCons.java* that contains a set of constants used by the different games and menus.

b) *The file Cell.java*: All cells (agents) extend this file, located in the *./games* package, which contains a basis set of basic features that support de development of new cells (agents) in order to take decisions (including reinforcement learning support) in the games. In this packages, there is also a *CoaCell.java* file, that extends *CellCoa.java*, and is intended to support the development of cells (agents) in coalitional games.

c) *The file DlgGraphics.java*: this file is used to display the appropriate graphical data in a separate window.

d) *The file MainWindow.java*: this file manages the all the graphical windows used in the game, together with the menu bar and the main thread used in the simulations. At the end of the file there are two key methods: *vNormalRun()* and *vBatchRun()* that control, respectively, the normal simulation with graphical windows or the batch execution.

Besides these basic files, every game is located in its own package containing its main code (inherited from Game) and, optionally, a graphic window (DlgParam_GAME_NAME.java) to manage the particular game parameters and, if necessary, a child of Cell (Cell_GAME_NAME.java) that inherits from *Cell.java* or *CellCoa.java*; to describe the behaviour of every cell (agent).

4. Creating a new game

To create a new game, the programmer can reuse an already used game, changing the names in the appropriate package, but keeping the structure; or s/he can add a new game to the Games submenu together with including a new package with the corresponding files. Next, I describe how to manage the second case, but it can be also a description about the code parts to modify in the first one.

To add a new game, the programmer should follow these three simple steps:

1. Add the new text strings s/he would need into *GameCONS.java*. This is simple, just review the others and add the one you need for your game.

2. Initialize the menus for *games* and *parameters* in *MainWindow.java* and in *JPanelMainWindow.java*. This is also straightforward; just imitate the examples from the different games.

3. Create your own game extending *Game.java* and define your own parameters using *DlgParam.java*. If necessary, define your own cell behaviour extending *Cell.java*. Use other games as templates.

4. **Optional**: if necessary, select how to see the cells in *VisorWorld.java*. Again, the rest of the games are a good example about how to manage your cells.

5. **Optional**: if necessary store data from your game simulation, and display it using the data displaying windows available. Check out other games, as templates to see how to proceed.

5. Creating a new real-time data window

IPDNet includes multiple graphical data windows that cover most of the potential programmer needs. However, if the programmer wants to create its own graphical data window s/he should follow the next steps:

1. Include a new identifier for in *WindowCons.java*.

2. Include the new identifier and visor for *omDlGraf* and *oMIWindow* in *MainWindow.java*. Also add the item in the *Options* submenu.

3. Setup the adequate visor in *DlGraphics.java*.

4. Store new graphical data values in the game.

6. Using the IPDNet batch execution

IPDNet can be executed by means of the graphical windows, showing real time data values; but for long or exhaustive simulations it is more practical to run the simulator without graphics that consume execution time.

To execute IPDNet in batch mode, just modify the file *bIPDNet.java*, located in the *src* folder to run batch executions of any game.

7. References

[1] J.C. Burguillo. CellNet: a Hands-on Approach for Agent-based Modelling and Simulation. GitHub repository. 2018. https://github.com/jcburguillo/CellNet

[2] J.C. Burguillo. Self-organizing Coalitions for Managing Complexity. Springer ECC series (Emergence, Complexity and Computation 29). Springer International Publishing AG 2018. https://doi.org/10.1007/978-3-319-69898-4

[3] R.Axelrod, The evolution of Cooperation, Basic Books. NewYork, 1984.

[4] W.H. Press, F.J. Dyson, Iterated prisoner's dilemma contains strategies that dominate any evolutionary opponent, Proceedings of the National Academy of Sciences 109 (26) (2012) 10409–10413.

[5] C.Adami, A.Hintze,et al., Winning isn't everything: Evolutionary stability of zero determinant strategies, Nature Communications 4 (10.1038).

[6] P. Mathieu, J.P. Delahaye, Experimental criteria to identify efficient probabilistic memory-one strategies for the iterated prisoners dilemma, Simulation Modelling Practice and Theory 97 (2019) 101946.

[7] N. E. Glynatsi, V. A. Knight, A meta analysis of tournaments and an evaluation of performance in the iterated prisoner's dilemma, arXiv preprint arXiv:2001.05911.