Introduction to CellNet Simulator

Juan C. Burguillo

J.C.Burguillo@uvigo.es

Version 1.0 (January 2018)

Contents Table

1. Introducing the CellNet simulator

CellNet 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). CellNet has its origins in 2003, as a research resource created by the author to study multi-agent systems, evolutionary game theory and cellular automata simulations. Since then, it has been used in a number of research works by the author, Ph.D. students and some research colleagues from other Universities.

The main aim of CellNet is to allow researchers to quickly create a new network simulation in areas like complex networks, cellular automata, multi-agent systems, self-organization and game theory. CellNet simulating framework can be easily used by undergraduate and master students that want to enhance their knowledge within these topics in disciplines like artificial intelligence, computer science, applied mathematics, economics and engineering.

1.1 CellNet Basic Features

CellNet 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. CellNet 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 CellNet and the Book "Self-organizing Coalitions for managing Complexity"

CellNet allows a hands-on approach to simulating and modifying most of the coalition-based experiments presented in the book [1]:

Self-organizing Coalitions for Managing Complexity by J.C. Burguillo. Springer ECC series (Emergence, Complexity and Computation 29). *https://doi.org/10.1007/978-3-319-69898-4*

Following the book, the readers can test the simulations by means of:

- *Running Micro-simulations*: most of the experiments described in the research chapters (except the ones from chapter 10 about *Electrical Vehicles and SmartGrids*, which were programmed in Netlogo) 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, so the reader can explore the contents of the book, repeat some experiments or perform new ones by just selecting different parameter values.
- *Running 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.
- *Modifying the algorithms*: readers having standard Java programming skills can redesign their own algorithms, and then test the behaviours 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. CellNet 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 analysers.

1.3 A graphical description of the graphical tools provided by CellNet

Fig. 1 provides a snapshot of the CellNet 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.

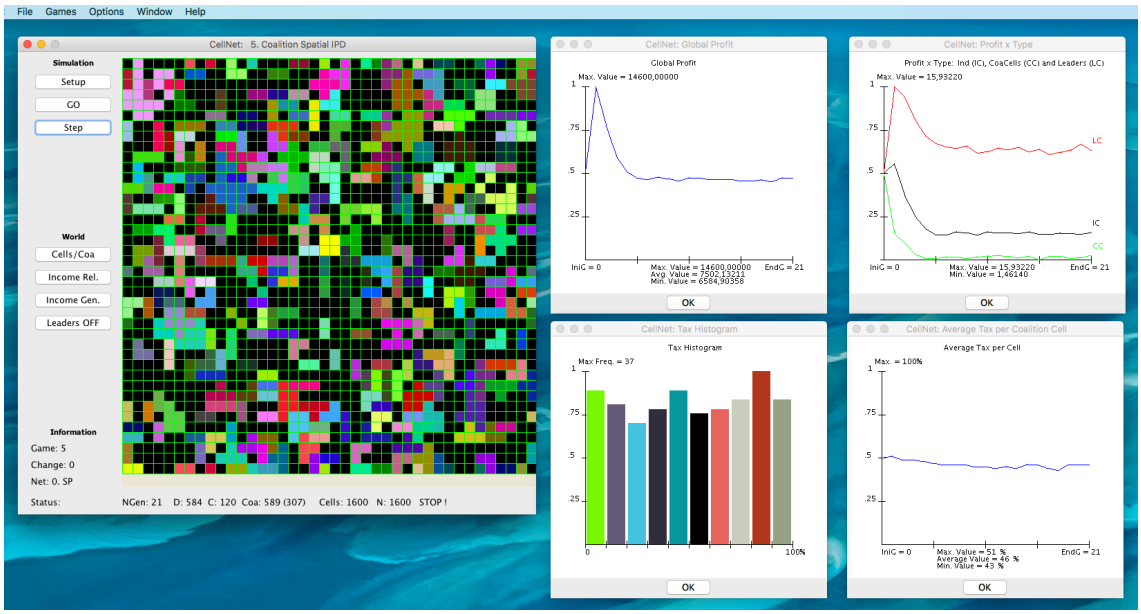
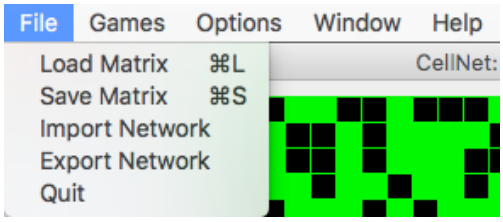


Fig. 1: A snapshot of the CellNet simulator and the graphical data windows.

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.).



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

[2] https://gephi.org

Fig. 2: The menu bar and the File menu

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

Fig. 3: The menu bar and the Games 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.

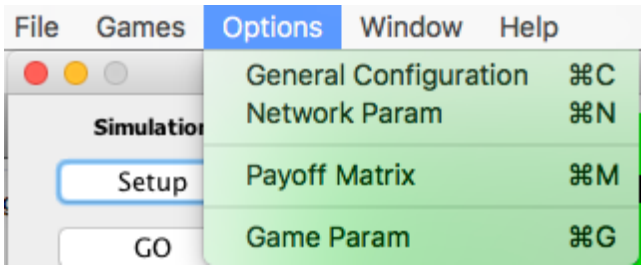


Fig. 4: The Options menu

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.

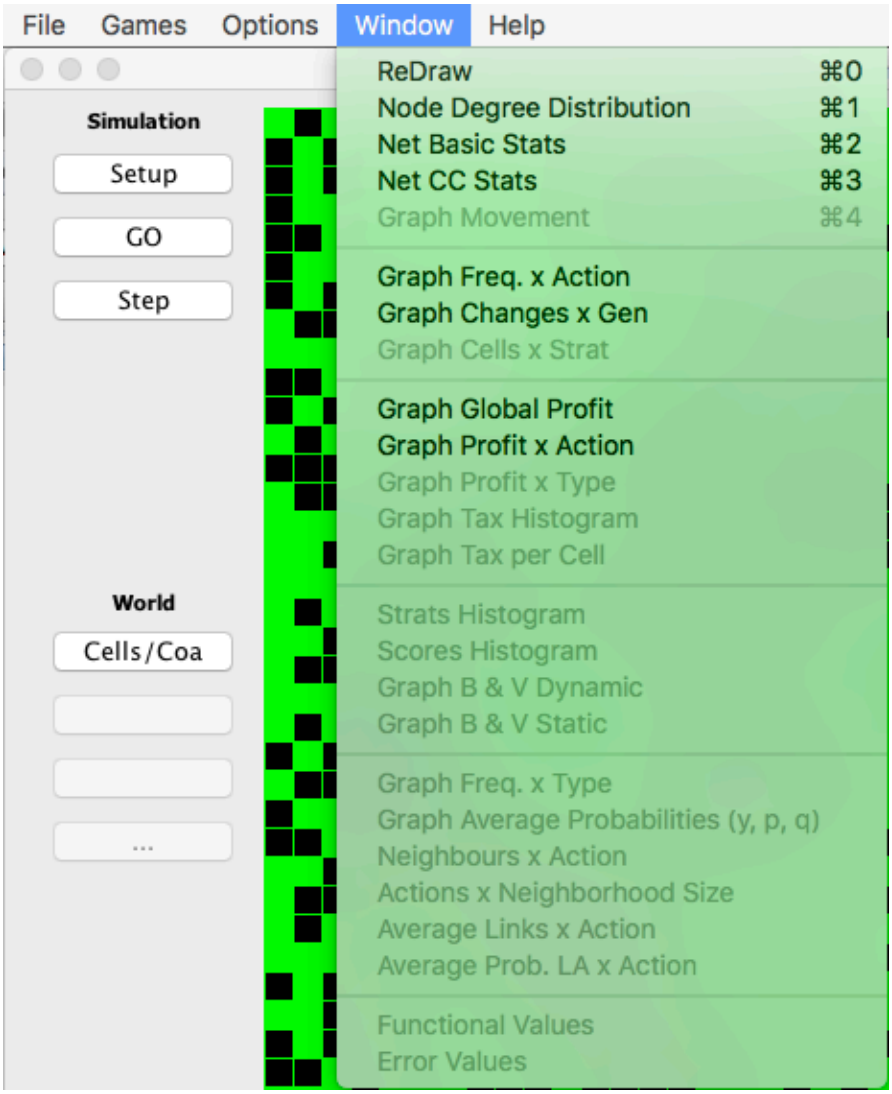


Fig. 5: The Window menu

Finally, Fig. 6 presents the help menu with access to search facilities, a CellNet Help item and the About item, which gives contact information and the present game version.

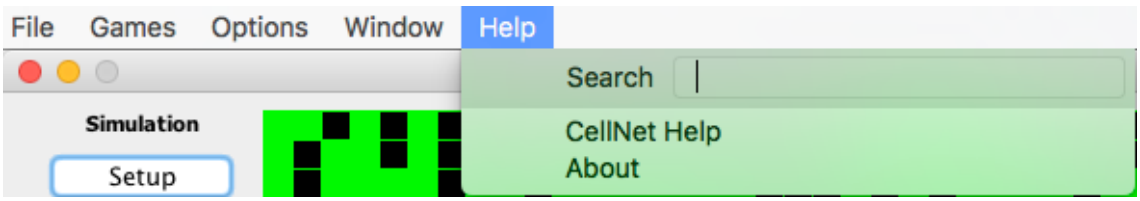


Fig. 6: The Help menu

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

2. Games included into CellNet

As described in Fig. 2, CellNet provides a user-friendly access to several in-built games ready to be executed with a single mouse click. These games are shortly described next.

0. *Game Social Groups:* this simple game illustrates cell movement. There are two social groups of cells (green and black) over a blue background. Each cell tries to stay with similar cells of the same colour, and in each generation it can randomly move around to find some group colleagues. After some generations cell types are clustered in the grid.

1. *Game LIFE & Coalition LIFE:* this is the classical Conway's Game of Life extended with coalitions in the way described in [2].

2. *Game Sat Ball:* This simulation models a match between two teams that use a ball. The game is a template to explore simulations including cell movement and reinforcement learning.

3. *Game Iterated Prisoner's Dilemma (IPD):* this is a network version of the famous iterated prisoner's dilemma, based in the model described in [3].

4. *Game Hawk-Dove-Possessor-Trader:* this is a network version of the economic game model initially described in [4], explored as a spatial game in [5] and as a more general model with complex networks in [6].

5. *Game Coalition Spatial IPD:* This is a coalitional version of the spatial IPD, following the model described in [7].

6. *Game Coalition Indirect Reputation:* This game models indirect reciprocity based on the seminal work published in [8] extended by the model described in [9] an afterwards upgraded in [10].

7. *Game Coalition Meta-norms:* This is an experimental simulation based in the work done by Axelrod in [11], and extended in a collaboration with Samhar Mahmoud from King's College London (KCL), who has published a series of papers with variations of the initial model.

8. *Function Optimization:* This is a simulation of the possibilities to use coalitional cellular networks in function optimization [12].

9. *Function Prediction:* This is a simulation of the possibilities to use coalitional Self-organizing maps (CASOM) for Time Series Prediction (TSP) [13].

The first four games (0 to 3) are aimed to serve as development templates, showing several features available at the simulator. The rest of the games (4-9) correspond to models published as original work in international journals and conferences, and afterwards compiled and upgraded in [1].

3. A description of the API

This section describes the Application Programmer's Interface (API) of CellNet. 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 CellNet packages and main files

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

a) Basic files:

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

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

CellNet.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.

./problems: describe optimization and prediction problems to be used in games.

./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 games 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

CellNet 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 CellNet batch execution

CellNet 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 CellNet in batch mode, just modify the file *bCellNet.java*, located in the *src* folder to run batch executions of any game.

7. References

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

[2] Burguillo J.C. (2018) A Coalitional Game of Life. In: Self-organizing Coalitions for Managing Complexity. Emergence, Complexity and Computation, vol 29. Springer, Cham.

[3] Schweitzer, F., Behera, L., & Mühlenbein, H. (2002) Evolution of cooperation in a spatial prisoner's dilemma. Advances in Complex systems, 5(02n03), 269-299.

[4] Yee, K. K. (2003). Ownership and trade from evolutionary games. International Review of Law and Economics, 23(2), 183-197.

[5] Burguillo, J.C., Peleteiro, A. (2010) Ownership and Trade in Spatial Evolutionary Memetic Games. Lecture Notes in Computer Science 4490, pp. 455--464.

[6] Burguillo J.C. (2018) Ownership and Trade in Complex Networks. In: Self-organizing Coalitions for Managing Complexity. Emergence, Complexity and Computation, vol 29. Springer, Cham.

[7] Burguillo, J.C. (2009) A Memetic Framework for Describing and Simulating Spatial Prisoner's Dilemma with Coalition Formation. Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS).

[8] Nowak, M.A., Sigmund, K. (1998) Evolution of indirect reciprocity by image scoring. Nature 393, 573–577.

[9] Peleteiro, A., Burguillo, J.C., Chong, S.Y. (2014) Exploring indirect reciprocity in complex networks using coalitions and rewiring. In: Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems, pp. 669–676. International Foundation for Autonomous Agents and Multiagent Systems.

[10] Burguillo J.C., Peleteiro A. (2018) Promoting Indirect Reciprocity Using Coalitions. In: Self-organizing Coalitions for Managing Complexity. Emergence, Complexity and Computation, vol 29. Springer, Cham.

[11] Axelrod, R. (1986) An evolutionary approach to norms. American political science review, 80(4), 1095-1111.

[12] Burguillo J.C., Dorronsoro B. (2018) Optimization Models with Coalitional Cellular Automata. In: Self-organizing Coalitions for Managing Complexity. Emergence, Complexity and Computation, vol 29. Springer, Cham.

[13] Burguillo J.C., García-Rois J. (2018) Time Series Prediction Using Coalitions and Self-organizing Maps. In: Self-organizing Coalitions for Managing Complexity. Emergence, Complexity and Computation, vol 29. Springer, Cham.