Appendix B

User manual

This is the user manual for any-network simulation. Circle-simulation works very similarly and its features are a subset of features described in this manual, hence circle-simulation is not described here. If you wish to use or improve circle-simulation, please consult documentation that includes its low-level description.

Table of Contents

1 Creating a network

Simulation requires some information about how to connect agents. This is achieved by providing it with a Network object. This is in fact a graph, where each node contains its unique id (in range 0 to number of nodes minus one), X and Y display position, and a list of adjacent agents' ids.

1.1 via text file editing (*.network file extension)

Network object can be saved to a file or loaded from a file. These files use *.network file extension and are just text files that can be edited by the most simple text editors such as notepad, gedit or nano. Each line of the file is saved in the following format:

ID (X; Y): A, B, C

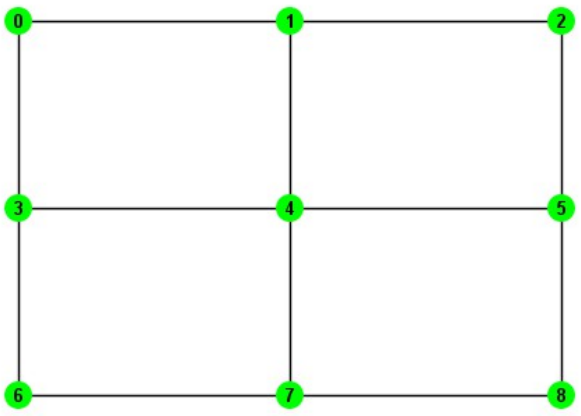
- ID is a unique agent's id. It has to be in range between 0 and N-1 (both inclusive), where N is the number of nodes in the network.
- X and Y are double coordinates of the node, position is proportional to the sizes of the window where the network is displayed, hence they should be in range 0 to 1 (both inclusive). For example, if X = Y = 0.5, the node will be displayed in the centre of the screen.
- A, B, C are comma separated neighbours' ids. This can be empty – colon will then be a last character in the line.

Worth to know:
- all whitespaces are ignored
- empty lines are ignored
- lines starting with # character are considered comments and are ignored
    - if you add a comment, load the network into GUI network creator and save it back, all comments, empty lines etc. will be removed
- trailing comma is ignored (unless it follows a colon with no id in between)
- if agent A is a neighbour of agent B then also agent B is a neighbour of agent A – if a file is edited in that way that it is not true, the correct behaviour of the simulation is not guaranteed

Example network file (3x3 grid):

0 (0.25; 0.25): 1, 3
1 (0.5; 0.25): 0, 2, 4
2 (0.75; 0.25): 1, 5
3 (0.25; 0.5): 0, 4, 6
4 (0.5; 0.5): 1, 3, 5, 7

5 (0.75; 0.5): 2, 4, 8
6 (0.25; 0.75): 3, 7
7 (0.5; 0.75): 4, 6, 8
8 (0.75; 0.75): 5, 7



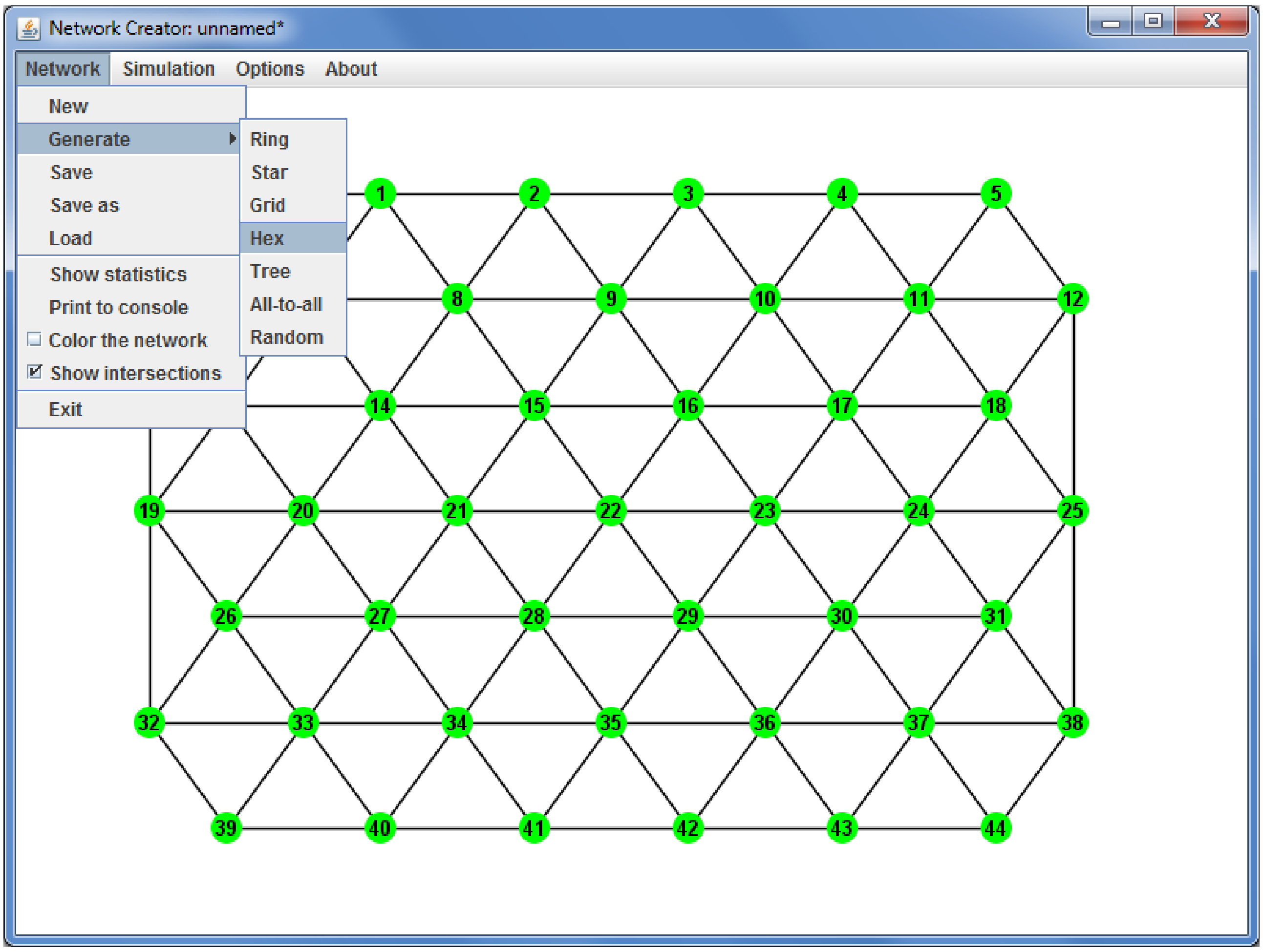
3x3 grid network

Example network file (single node):

0 (0.5; 0.5):

1.2 via graphical user interface

Recommended way of creating a network is via Network Creator window. It is enough to create an object of class network.GUI.creator.MainWindow and it will display a window. This is also an entry point of the application.



Network creator window

Mouse interface is described in about/help menu. It includes five actions:

CREATE NODE
Double click with left mouse button.

DELETE NODE
Double click with right mouse button.

DELETE NODE - KEEP CONNECTIONS
Ctrl + double right click.
Deletes node, however for each two nodes A and B that were connected via deleted node, after deletion nodes A and B will be connected directly.

(DIS)CONNECT TWO NODES
Drag from one node to another while
keeping left mouse button down.
If there is no node at the destination, a new one will
be created and connected to previously selected.

MOVE NODE
Press with right mouse button on the node
and keeping the button down move to the
new destination.

It is possible to generate some predefined types of networks in network/generate menu. Feel free to experiment with them!

Worth to know:

- random network generator has 1 second timeout – after this time it returns whatever it generated so far
- random network generator does not guarantee to generate given number of nodes and edges (although it tries hard to do so), possible reasons are:
    - too small window to avoid nodes overlapping
    - more edges requested than can be created by connecting all agents to all others
- if generating a network with more than a few hundred edges it is recommended to disable anti-aliasing in options menu

2 Starting the simulation

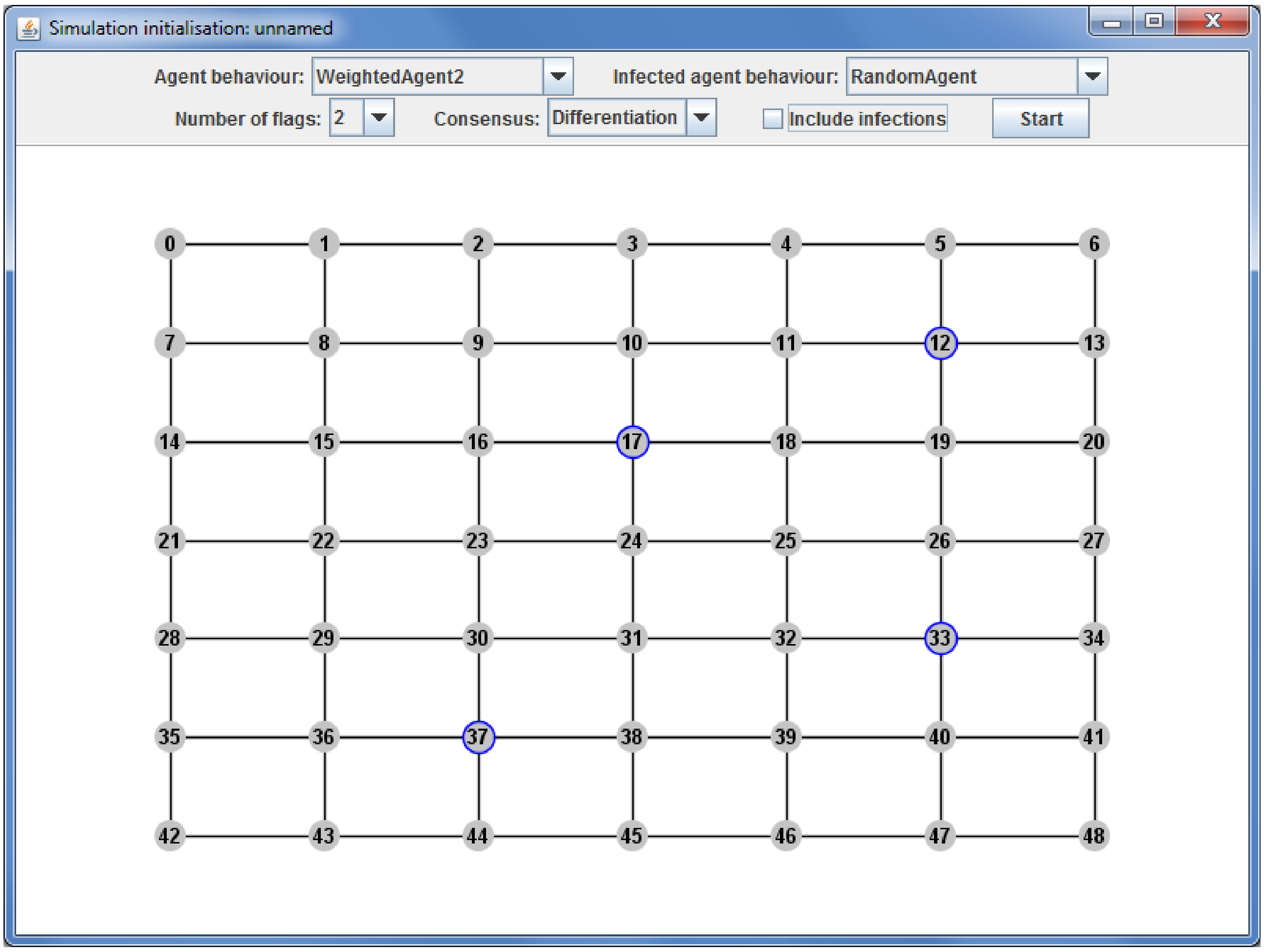
Before the actual game is started there is only one main step left – choosing agent's behaviours. Once in network creator press simulation/initialise to bring up a new window that will display possible options to choose. Alternatively, you can create initialisation window programmatically by creating network.GUI.simulation.InitialisationWindow, which requires only a Network object and optional name of this network.

Simulation initialisation window allows to choose:

- behaviour of normal agent
- behaviour of infected agents
- infected agents (by left clicking on chosen nodes)
- number of flags
- consensus mode
- whether to include infected agents in consensus checking

Once you decide on these options press “start” button.

If you would to create a simulation window programmatically, you first have to create a Simulation object (for further information on how to do this refer to API and Design). Once you create a simulation simply use it in a SimulationWindow constructor among an optional name.



Agent behaviour: WeightedAgent2
Infected agent behaviour: RandomAgent
Number of flags: 2
Consensus: Differentiation
Include infections

Simulation initialisation window

Worth to know:

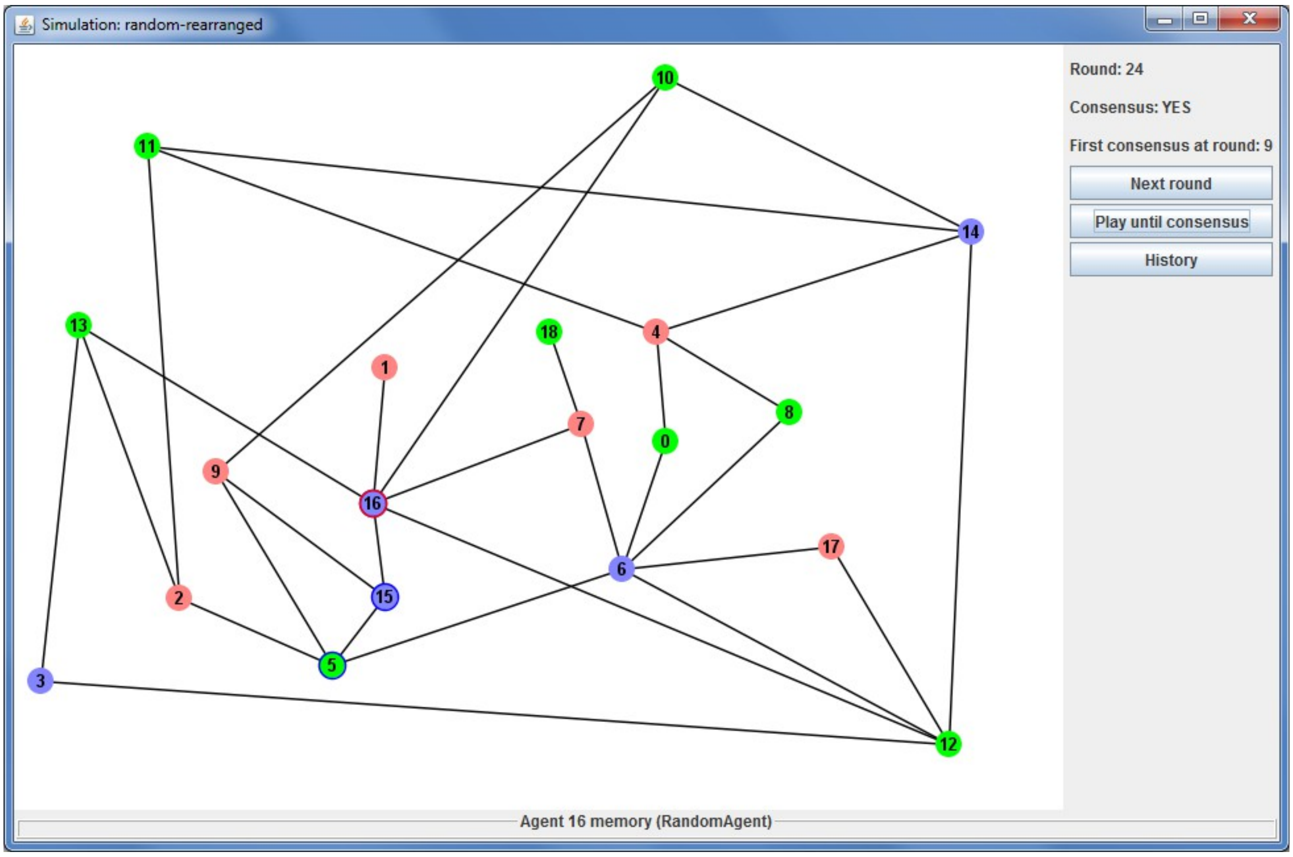
- if you hover mouse pointer over the combo box with flag number, it will display a tooltip with recommended number of flags to use
- agents do not know what type of game (consensus mode) is played – this is defined in their behaviour. If agent A is programmed to play differentiation while agent B to play colouring, you may set the simulation to differentiation and use agents B as infected so they will be trying to achieve opposite goal
- agents classes available to choose are defined in InitialisationWindow in a field Class[] agentClasses

3 During game

Except memory accessor and history (described below), simulation GUI offers very few possible actions for the user. These are provided via two buttons:

- next round – that progresses the simulation by one round
- play until consensus – that freezes GUI and plays in background until consensus is achieved or until 5 seconds elapse, and then shows current round

Simulation GUI also shows the current round, whether the agents are currently in consensus and at which round consensus was achieved for the first time (if at all).



Simulation window (game in progress)

3.1 Agent Memory Accessor

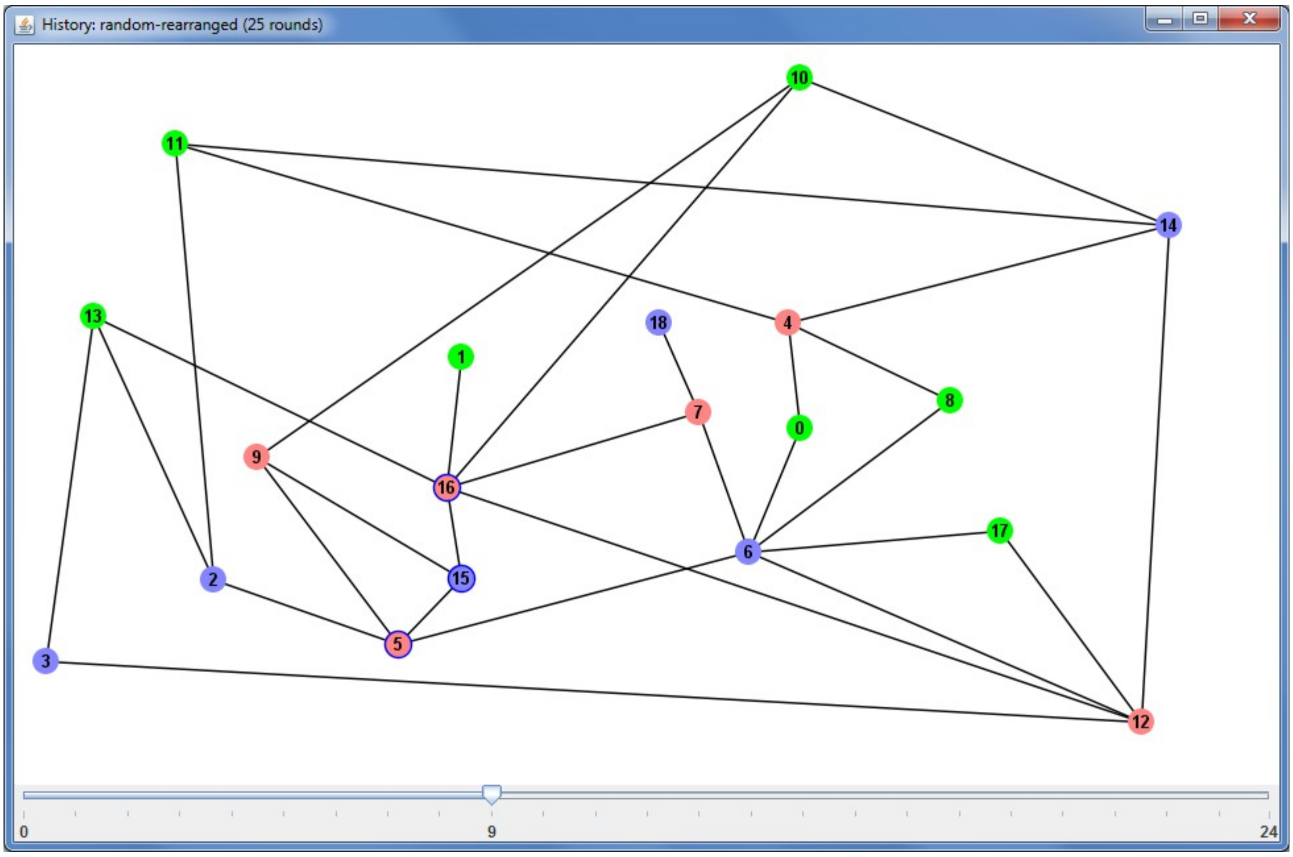
Agent memory accessor, visible at the bottom of the window, shows the fields of the class of last selected agent (to select an agent simply left click on it). If access modifiers of these fields do not prevent it, agent memory accessor also displays their values and allows to modify them. Please keep in mind that memory accessor's support is quite limited – for the list of not implemented features please refer to future work.

3.2 History

If you would like to see flags raised by agents in previous rounds, simply click “history” button – it will bring up a new window.

In a history window your only option is to choose a round you would like to see by selecting a value on the slider. History window is completely independent of the simulation window – it can be closed or new ones can be created and it will not affect the simulation. Also, any further changes in the simulation (such as new rounds played after history window was created) will not be shown in the history.

If you used simulation initialisation to start a simulation, history will be enabled by default. However, if you create a SimulationWindow by providing it with your own Simulation object, history button may be disabled depending on whether the history was enabled in the simulation or not.



History window

4 Creating your own agent

To create your own agent you have to extend network.simulation.AbstractAgent class. It has to provide a constructor that takes a single int parameter – the maximum number of flags to be used by an agent, as this will be provided by the simulation.

The behaviour of an agent is defined in getNewFlag(int round) method. Obviously, you can create extra methods or even objects, but the simulation will call this method to ask an agent for its choice. It is guaranteed that simulation will be calling this method with consecutive numbers starting from 0 or 1 – this depends on whether the simulation was provided with flags for initial round (in this case a first call will be with round equal 1).

Your agent can access the following fields and methods from AbstractAgent class to obtain basic information about its local neighbourhood:

- final int maxFlags – the maximum number of flags available in game (this is in fact what you have passed in your constructor to the super constructor)
- AgentDelegate[] neighbours – 1st degree neighbours
- int getFlag() – flag raised by the agent in the previous round

5 Automated testing

ExperimentScheduler provides easy, efficient and safe method of performing numerous automated tests. It is done in three simple steps:

- create ExperimentScheduler object – at this point you define maximum number of threads used for testing. This count does not include current thread that will be used to start new experiments and will be idle for most of the time.
- add experiments to the scheduler, this can be done in a few ways:
    - using methods provided in the scheduler – this will manage a few parameters, such as name, automatically
    - creating one of predefined experiments and adding it to the scheduler with addExperiment(AbstracExperiment) method
    - creating your own Experiment and adding it with the same method as above
- call execute() method – this will block current thread and wait until all experiments are finished

Result of each round, as well as summarised result, will be saved to a text file – one per experiment. If an exception is thrown the experiment is terminated and the exception is logged.

Experiment scheduler uses round limit and stops a game if agents are unable to achieve consensus within this limit. However, experiment scheduler does not prevent blocking threads (e.g. infinite loops in agent's behaviour) – in this case it will never finish its task.