



## **A Simulator for Repeated Games**

User Manual

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Getting started</b>	<b>5</b>
2.1	Edit the configuration . . . . .	5
2.2	Start a simulation . . . . .	7
2.3	View the simulation results . . . . .	7
2.3.1	Detailed Output . . . . .	8
2.3.2	Abstracted Output . . . . .	9
2.4	Save the result . . . . .	11
<b>3</b>	<b>Multiconfigurations</b>	<b>12</b>
<b>4</b>	<b>Extensions</b>	<b>14</b>
4.1	Create new strategies . . . . .	14
4.2	Create new games . . . . .	16
4.3	Create new groups . . . . .	16
4.4	Create new populations . . . . .	18
4.5	Important Information on Saving and Exporting Extensions . . . . .	20
<b>5</b>	<b>The Plugin-System</b>	<b>21</b>

# 1 Introduction

This program is designed for scientific research in the field of game theory. It is a simulation environment that can be used to investigate the formation of equilibriums at repeated games of multiple agents. An explanation of the underlying simulation process follows, see also Fig. 1.

A simulation consists of multiple iterations. An iteration begins with the initialisation of agents with strategies, capital and group affiliation<sup>1</sup>. Amount of agents, assignment mechanisms for initials strategies and capital as well as the group structure can be configured by the user.

Afterwards, so called adaption steps are repeated until either an equilibrium is reached or a configurable maximum number of adaption steps was executed. The equilibrium criterion can be specified by the user as well.

An adaption step consists of a fix amount of rounds. In each round, the agents are matched to pairs by a configurable pair building algorithm. The agents of each pair then play the underlying game (f.ex. the prisoners dilemma) according to their current strategies and receive their payoffs.

After the last round is finished, the agents are ranked based on their performance in the elapsed rounds and they adapt their strategies. The goal that motivates their choice of strategy adaption is not the maximisation of their total capital but to obtain as high of a rank as possible. Both the success quantification and the strategy adaption mechanism can be configured by the user.

After all iterations have finished, the results of the simulation are displayed.

---

<sup>1</sup>In each simulation, the agents are partitioned in a fix amount of groups. The group of an agent determines how he is initialised with strategies and capital. Also, strategies can refer to other agents group affiliation. Two agents are said to be affiliated, if both are members of the same group and that group is "cohesive". Agents of uncohesive groups are thus effectively "groupless" in the eyes of other agents.

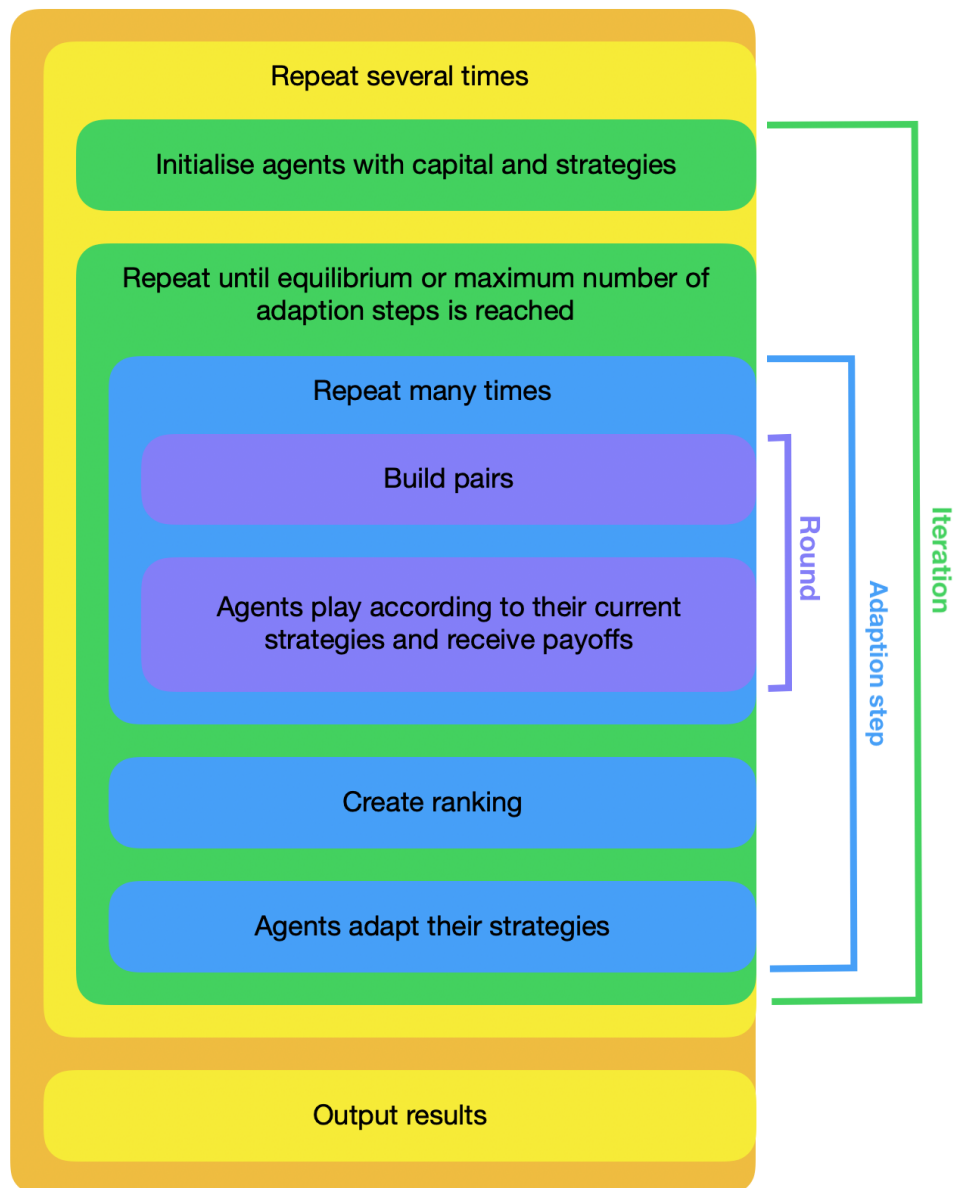


Figure 1: The simulation process



## 2 Getting started

When the program is first started, the home window will open as in Fig. 2. Since no simulations have been executed yet, it is mostly empty.

The home window is roughly divided into three areas. In the top area, a short summary of the currently active configuration is displayed ((1) in Fig.2). If the user hasn't created a configuration of his own yet, a predefined one will be active. On the right-hand side of the summary reside two buttons ((2) and (3)). The one labeled with a cog symbol opens up the configuration window, in which the currently active configuration can be modified (see 2.1). Pressing the Play-button will start a simulation with the currently active configuration. The left-hand side area ((4) in Fig.2) will contain a list of all running, finished and cancelled simulations. If a finished simulation is selected, detailed information about its results will be displayed in area (5) (This area will from now on be referred as the "output view").

### 2.1 Edit the configuration

To edit the active configuration, press the cog-labeled button in the home window. This will open up the configuration window (see Fig.3). On the left-hand side, all configurable parameters of a simulation can be modified. If the selected pairing algorithm, success quantification, strategy adaption mechanism or equilibrium criterion has configurable parameters, they can be entered below the corresponding dropdown menu (see (1) in Fig.3). On the right-hand side, a short description and a table containing the payoffs of the selected game are displayed (see (2) in Fig.3). Below ((3) in Fig.3), a multiconfiguration can be activated, see section 3.

The  - button will reset all settings to the default configuration. Pressing the  - button will close the configuration window and apply the made changes to the active configuration.

It is possible to export the configuration to an external file by navigating to "File → Save configuration" in the menu bar of the home window. Such a configuration file can be loaded and set as the active configuration via "File → Load configuration". For important information on saving and exporting files see section 4.5.

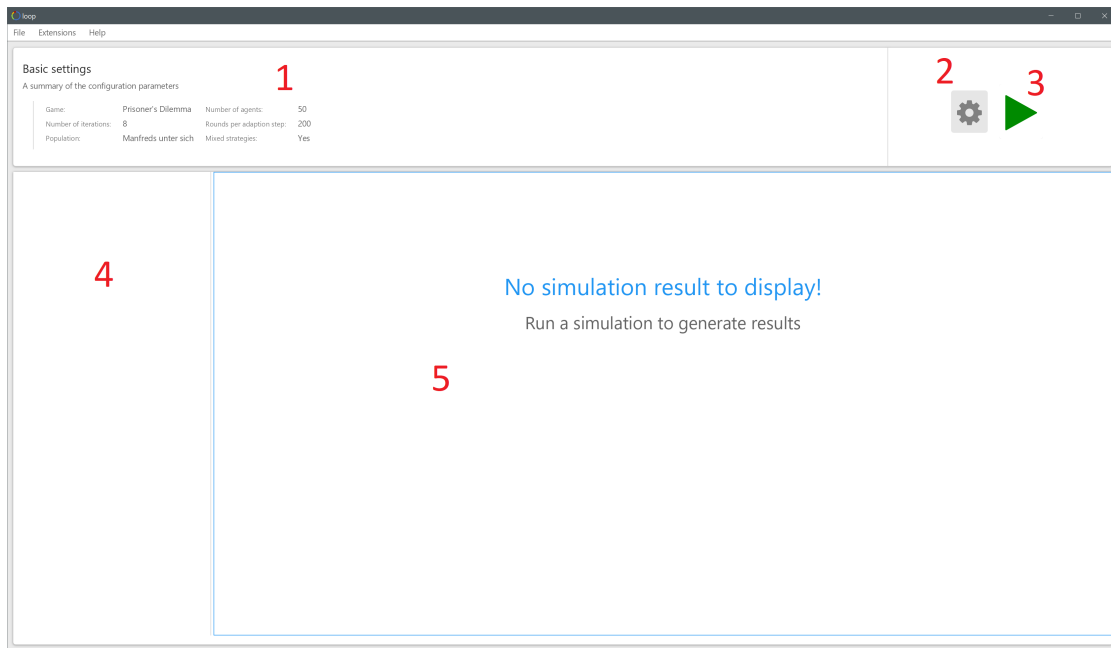


Figure 2: The home window after the first start of the program.

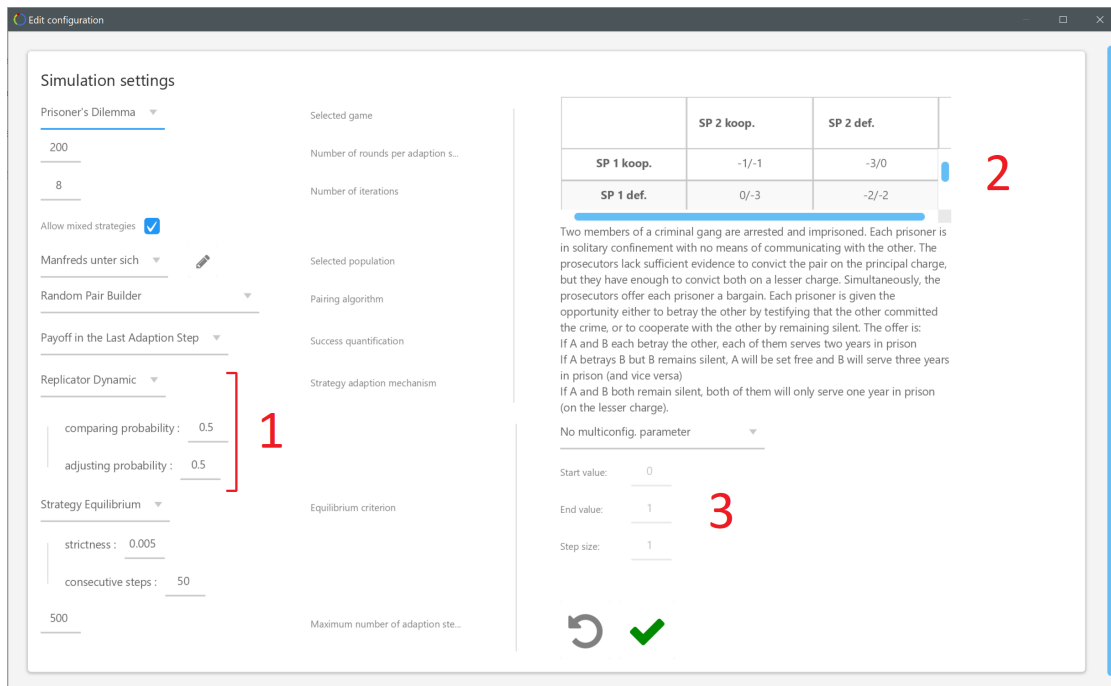


Figure 3: The configuration window.

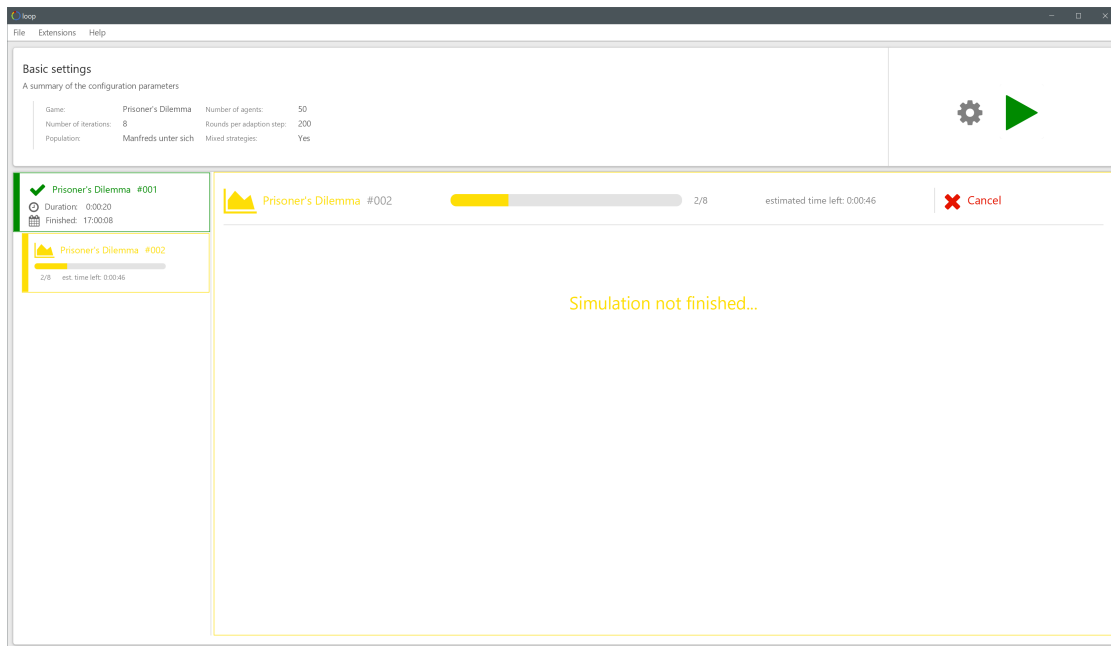


Figure 4: The home window with a running simulation selected.

## 2.2 Start a simulation

Pressing the Play-button in the home window will start a simulation with the currently active configuration. It will then appear in the list in the left-hand side area of the home window. The list entry displays an estimate of the time left until the simulation finishes as well as how many iterations have already been executed. If the running simulation is selected, the output view will contain the same information as the list entry as well as a button labeled with an . If pressed, the simulation is cancelled.

## 2.3 View the simulation results

As soon as a simulation is finished, its list entry turns green and displays time and date of the moment it finished as well as the duration of its execution:



If selected, the output view will display detailed information about the simulations results. You can also open the output in a new window by pressing . The output is divided

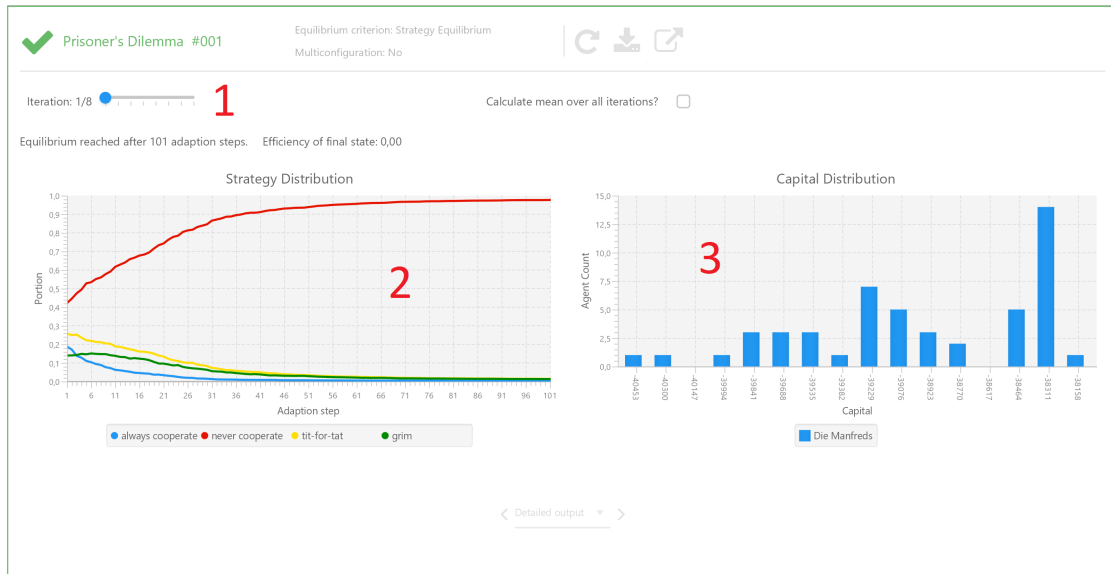


Figure 5: The detailed output.

into two subpages: the “Detailed output” and the “Abstracted output”. To switch between the two, the ◀ and ▶ buttons at the bottom of the output view can be used.

### 2.3.1 Detailed Output

In the detailed output (see Fig.5), information about strategy and capital distribution of the agents are displayed for any selected iteration. To select an iteration, use the correspondingly labeled slider at the top of the output view ①. The iterations are sorted by their final efficiency, i.e. in the case of Fig.5, iteration 1/8 is the least, iteration 8/8 the most efficient among the eight executed iterations.

**Strategy distribution:** The line chart labeled “Strategy distribution” ② displays the evolution of the mixture of strategies used by the agents over the time of the simulation. It contains one line for every used (pure) strategy. Each line indicates the relative frequency of the corresponding strategy being used by agents in every single adaption step. For example, consider the strategy distribution displayed in Fig.6. It tells us that in the beginning of the simulation, all strategies appeared with a similar frequency of 20% to 35%. Towards the end, “grim” prevailed with a frequency of about 67%, while “tit for tat” stayed at about 23% and “never cooperate” and “always cooperate” dropped below 10%.

**Capital distribution:** The bar chart labeled “Capital distribution” ③ displays a histogram of the final total capitals of all agents at the end of the simulation. The width of the bins



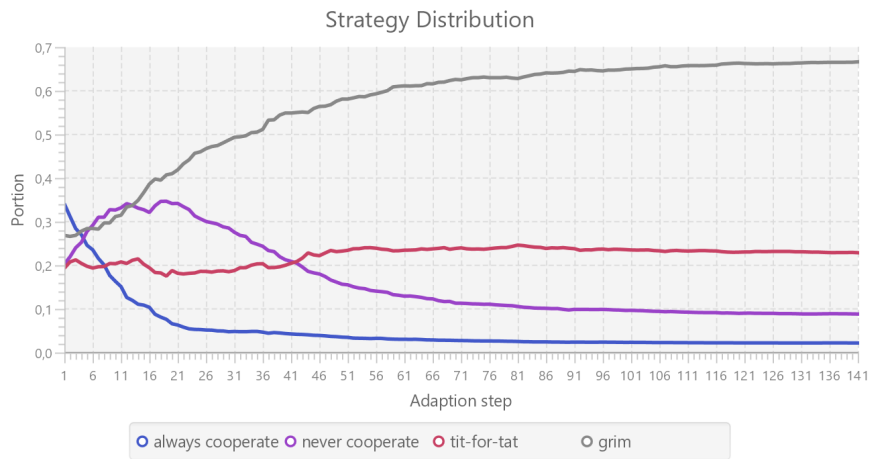


Figure 6: An exemplary strategy distribution.

of the histogram are chosen in a way such that there is always about 15 bins. Each bin is then labeled with the mean of the interval covered by it. Note that final capitals might be negative (as in Fig.7) if the payoffs of the used game are negative (such as in the prisoner's dilemma).

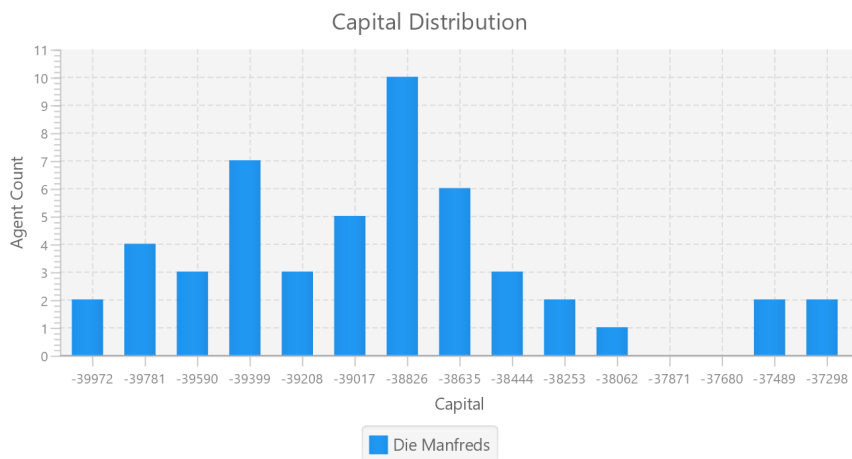


Figure 7: An exemplary capital distribution.

### 2.3.2 Abstracted Output

The abstracted output (see Fig.8) contains information abstracted from all executed iterations. At the top ① the “equilibrium frequency”, i.e. the portion of all iterations in which



Figure 8: The abstracted output.

an equilibrium was reached is displayed. Below reside two histograms, the “Efficiency distribution” and the “Distribution of executed adaption steps”.

**Efficiency distribution:** The efficiency distribution (2) is a histogram of the final efficiencies of all executed iterations. Consider Fig.9. In the corresponding simulation, five iterations finished with an efficiency above 0.8, one with about 0.54 and two with almost 0.0. Again, bin width is chosen such that there is around 15 bins in total and the labels display the mean of the interval covered by the bins. Below the strategy distribution, the mean efficiency of all iterations is displayed, in this case 0.614 (see Fig.8).

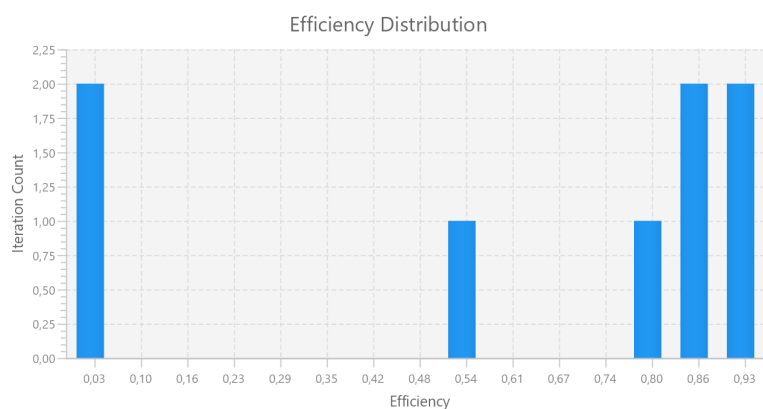


Figure 9: An exemplary efficiency distribution.

**Distribution of executed adaption steps:** This chart (3) is a histogram of the amount of executed adaption steps of all iterations. Consider Fig.10; in the corresponding simulation, all iterations took between 101 and 160 adaption steps to finish, quite equally distributed among that range. Below the distribution, the mean amount of executed adaption steps is displayed, in this case 129 (see Fig.8).

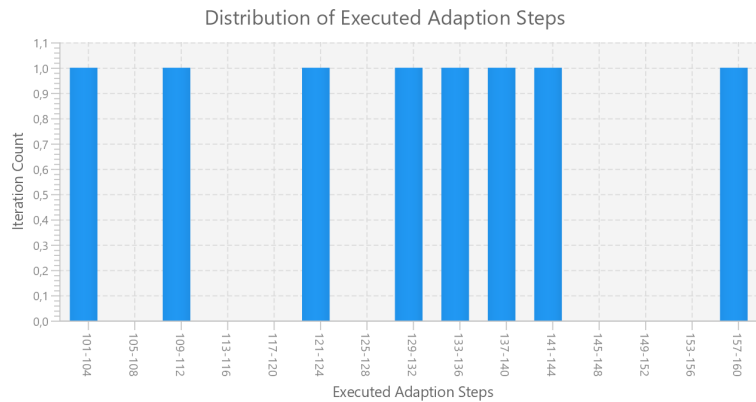




Figure 10: An exemplary distribution of executed adaption steps.

Below the charts resides a dropdown menu labeled “Considered iterations” (4). It can be used to determine whether all iterations or just the ones where an equilibrium (or no equilibrium) was reached shall be considered in the histograms above.

## 2.4 Save the result

If you wish to save the result of a simulation so you can take a look at it again later, you may do so by clicking the  button at the top of the output view. This will open a file dialog where you can choose a place to save your result. To load a result that has been exported in this way, press “File” and then “Load simulation result” in the menu of the home window.

You can also set the configuration that was used for a selected simulation as the active configuration by pressing the  button.

### 3 Multiconfigurations

A multiconfiguration can be regarded as a set of (normal) configurations (the “elementary configurations”), where all parameters but one (the “multiconfiguration parameter”, short: multi-parameter) are fix and the multi-parameter varies through a certain set of values. For example, one could specify a multiconfiguration in which all parameters are as in the preconfigured configuration but the round count (the amount of rounds per adaption steps) varies from 100 to 200 in steps of 10 (see Fig.11).

To create a multiconfiguration, open the configuration window and choose the desired multi-parameter in the dropdown menu at the bottom right. Possible multi-parameters are:

- round count
- maximum number of adaption steps
- configurable numerical parameters in the success quantification, the pair building algorithm, the strategy adaption mechanism and the equilibrium criterion
- the size of a group in the selected population
- the size of the segments in one of the groups of the selected population<sup>2</sup>

Below the dropdown menu, start value, end value and step size of the multi-parameter can be entered.

The output changes slightly for multiconfigurations. In the output pages “Detailed output” and “Abstracted output”, a new slider labeled “Configuration” appears, with which the considered (elementary) configuration may be selected. Next to the slider, the name of the mutli-parameter and its value in the currently selected configuration are displayed. On top of that, a third output page is available, the “Multiconfiguration output” (see 12). It contains two line graphs:

**Efficiency and equilibrium frequency:** The first chart plots equilibrium frequency<sup>3</sup> and mean final efficiency against the value of the multi-parameter.

**Mean amount of executed adaption steps:** The second chart plots the mean amount of executed adaption steps against the value of the multi-parameter.

---

<sup>2</sup>Works only for groups that consist of exactly two segments. If  $X \in [0, 1]$  is the value of the multi-parameter, then the first segment has relative size  $X$  and the second segment has relative size  $1 - X$ .

<sup>3</sup>The portion of iterations in which an equilibrium was reached

Simulation settings

Prisoner's Dilemma

400

8

Allow mixed strategies: ☒

Manfreds unter sich

Random Pair Builder

Payoff in the Last Adaption Step

Replicator Dynamic

comparing probability: 0.5

adjusting probability: 0.5

Strategy Equilibrium

strictness: 0.005

consecutive steps: 50

500

Maximum number of adaption ste...

Selected game

Number of rounds per adaption s...

Number of iterations

Selected population

Pairing algorithm

Success quantification

Strategy adaption mechanism

Equilibrium criterion

	SP 2 coop.	SP 2 def.
SP 1 coop.	-1/-1	-3/0
SP 1 def.	0/-3	-2/-2

Two members of a criminal gang are arrested and imprisoned. Each prisoner is in solitary confinement with no means of communicating with the other. The prosecutors lack sufficient evidence to convict the pair on the principal charge, but they have enough to convict both on a lesser charge. Simultaneously, the prosecutors offer each prisoner a bargain. Each prisoner is given the opportunity either to betray the other by testifying that the other committed the crime, or to cooperate with the other by remaining silent. The offer is: If A and B each betray the other, each of them serves two years in prison. If A betrays B but B remains silent, A will be set free and B will serve three years in prison (and vice versa). If A and B both remain silent, both of them will only serve one year in prison (on the lesser charge).

round count

Start value: 100

End value: 200

Step size: 10

↺ ✓

Figure 11: The configuration window with “round count” as multi-parameter.

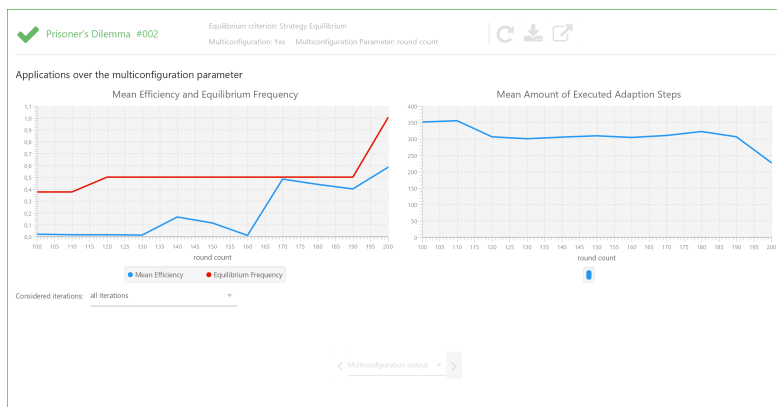


Figure 12: The multiconfiguration output.

Below the charts resides a dropdown menu labeled “Considered iterations” (4). It can be used to determine whether all iterations or just the ones where an equilibrium (or no equilibrium) was reached shall be considered in the plots above.

## 4 Extensions

To further customise configurations, the user can create own strategies, games, groups and populations and use them in simulations.

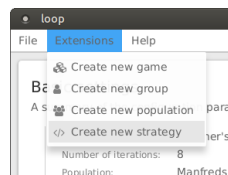
### 4.1 Create new strategies

The strategy creation lets you build strategies of the form

$$\text{If } X \text{ then } A, \text{ otherwise } B, \quad (1)$$

where  $A$  and  $B$  are already existing strategies and  $X$  is a configurable expression, such as “The opponent is in the same group as me” or “The opponent cooperated in his last game against me”.

The strategy creation window can be opened by clicking the “Create new strategy” entry in the “Extensions” menu in the home window:



It is divided into three vertically aligned sections. The topmost section ① contains two text fields in which name and description of the new strategy must be entered.

Section ② contains a set of predicates and operators, where most of the predicates can be configured using dropdown menus. They can be used to compound the expression  $X$  in (4.1). Clicking the **+** button adds the corresponding operator or predicate to the expression, which will be displayed in section ③.

When you add an operator to the expression, an empty box will be generated for every required operand (see Fig.14). To insert a predicate or operator as one of the operands, select the corresponding box (as in ④) and then add the desired predicate by clicking the **+** button.

To remove a predicate or an operator from the expression, click the **X** button next to it. A removed predicate will be replaced by an empty box. When you remove an operator, its operands will be removed as well.

Figure 13: The strategy creation window.

Figure 14: The generated expression.


Below the displayed expression reside two drop down menus (5) and (6), in which the strategies  $A$  and  $B$  from (4.1) can be selected.



To reset all settings, press the button in section (1).


To export the created strategy as a file, press the button. To add the strategy to the local repository and close the window, press (Further details and important information on exporting and saving see section 4.5).

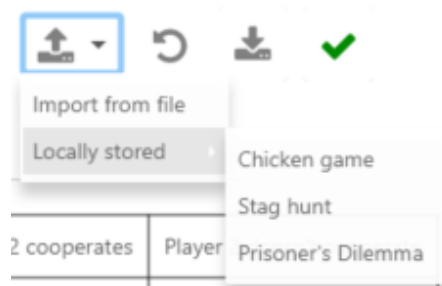
## 4.2 Create new games

When the “Create new game” entry in the “Extensions” menu is pressed, the game creation window opens (see Fig.15). Here, a new game can be created. To do so, a name, a description and the players payoffs must be entered in the corresponding text fields (① and ②). For example, an entry of the form “1/2” in the bottom left part of the payoff table means: “If player 1 doesn’t but player 2 does cooperate, player 1 will receive the payoff 1 and player 2 will receive the payoff 2”.

To reset all settings, press the  button.

To export the created game as a file, press the  button. To add the game to the local repository and close the window, press  (Further details and important information on exporting and saving see section 4.5).

To load an existing game press the  button. The following dropdown menu will pop up:



You can either import a game from a previously exported game file or open one of the locally stored games, i.e. the preconfigured games and the ones added to the repository by the user.

## 4.3 Create new groups

When the “Create new group” entry in the “Extensions” menu is pressed, the group creation window opens (see Fig.16). To create a new group, name and description must be entered in the corresponding text fields (①) and at least one segment must be configured. Segments can be added by pressing the “Add segment” button and removed by pressing the “X” button at the head of the corresponding tab.

For each segment, a capital distribution must be chosen and parametrised (②); it will be



Create a new game

Name of the game

Description

	Player 2 cooperates	Player 2 doesn't cooperate
Player 1 cooperates	- / -	- / -
Player 1 doesn't cooperate	- / -	- / -

Figure 15: The game creation window.

used to initialise agents of that segment with capital at the beginning of simulations. Secondly, a set of strategies must be selected out of all locally stored strategies (3). Those include the preconfigured ones as well as strategies created by the user (if existent).

The multislider (4) can be used to configure the relative sizes of the segments. For example, if 100 agents of the group in Fig.16 were to be initialised, 25 of them would belong to segment 2, thus receiving an initial capital drawn out of a poisson distribution with mean 15 as well as “always cooperate” or “tit for tat” as initial strategy.

By deselecting the checkbox labeled “Group is cohesive”, the group becomes “uncohesive”, i.e. agents of this group do not consider each other as members of the same group anymore. This becomes relevant when strategies are used that refer to other agents’ group affiliation. This can be used to model agents that “aren’t part of any group”, which is equivalent to being part of an uncohesive group.

To reset all settings, press the ↺ button.

To export the created group as a file, press the ⬇️ button. To add the group to the local repository and close the window, press ✅ (Further details and important information on exporting and saving see section 4.5).

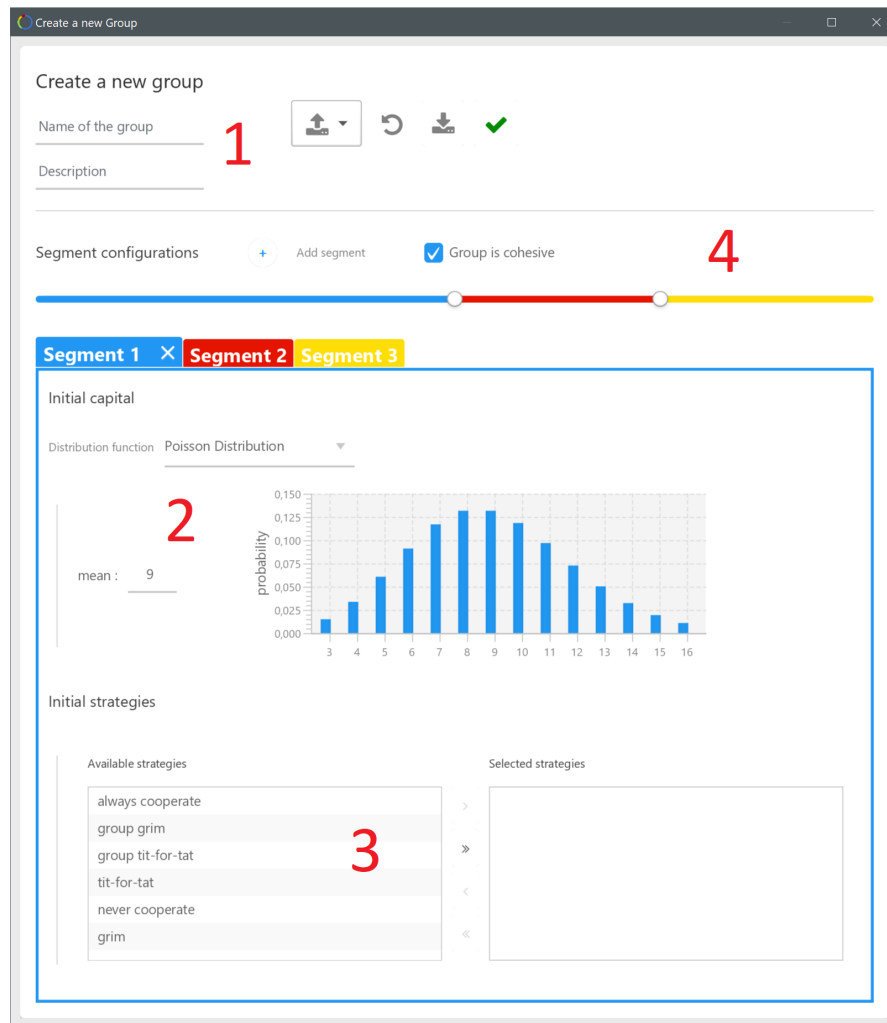



Figure 16: The group creation window.

To load an existing group press the  button. You can either import a group from a previously exported group file or open one of the locally stored groups, i.e. the preconfigured groups and the ones added to the repository by the user.

#### 4.4 Create new populations

When the “Create new population” entry in the “Extensions” menu is pressed, the population creation window opens (see Fig.17). To create a new population, name and description must be entered in the corresponding text fields (1) and at least one group must be

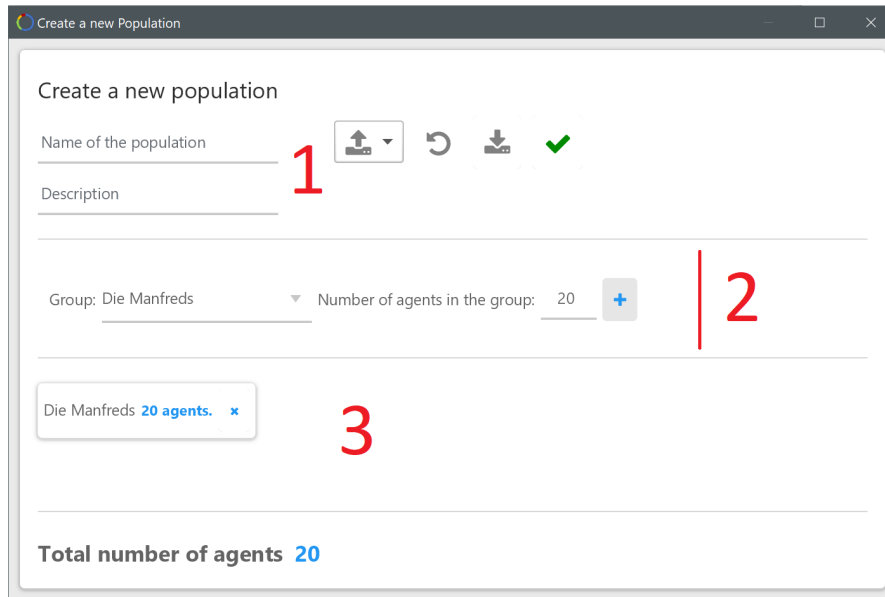








Figure 17: The population creation window.

added.



In order to add a group, choose the group in the dropdownmenu and enter the amount of agents of that group that shall be added (2). Then press the  button. To remove a group, press the corresponding  button. The set of all selected groups and their sizes is displayed in section (3).

To reset all settings, press the  button.

To export the created population as a file, press the  button. To add the population to the local repository and close the window, press  (Further details and important information on exporting and saving see section 4.5).

To load an existing population press the  button. You can either import a population from a previously exported population file or open one of the locally stored populations, i.e. the preconfigured populations and the ones added to the repository by the user.

## 4.5 Important Information on Saving and Exporting Extensions

There is an important difference between saving an extension (adding it to the local repository by pressing ) and exporting it as a file (by pressing ). If it is just saved, it will only be available until the program is closed, i.e. it is not stored persistently. If an extension shall be available persistently, it must be exported as a file and saved *inside the preconfigured folder*, which is `C:/Users/<Username>/loop/personallib/X`, where  $X \in \{\text{strategies, groups, populations, games}\}$ . All extensions that reside in those folders will be automatically loaded at program start.

There is another important note to make regarding the export of extensions. If an extension makes use of another extension (a population containing a group, a group containing certain strategies), then it will only store that extensions *name*. This means that when, for example, a previously exported population is loaded and it contains a group of name  $X$ , the local repository will be searched for a group of name  $X$ , and if none is found, the import will fail and an error message will be displayed. This also applies to configurations, i.e. an exported configuration only stores the *names* of the used game, population, etc. This has multiple consequences:

- Whenever you export an extension (or configuration) in order to store it persistently, make sure you also persistently store all referenced extensions (groups, strategies,...).
- When you export a configuration or an extension in order to send it to a colleague who also uses this program, make sure you send him all the referenced extensions as well.
- Since, for example, a population only stores the name of its contained groups, it can be modified by overwriting those groups. If a certain population contains a group with name  $X$ , and you were to delete that group and create a new one with the same name  $X$ , then the population would now contain that new group. This works in the same way for configurations with contained populations and games, and for groups with contained strategies.

## 5 The Plugin-System

The program features a plugin system which enables you to easily implement own versions of the following algorithms and mechanisms:

- the equilibrium criterion
- the success quantifier
- the strategy adjuster
- the pair builder
- the discrete distribution (used to initialise agents with capital)

In the following, we describe how to achieve this using eclipse. Start by creating a new Gradle-Project. Navigate to the projects root directory, create a new folder called “libs” and copy the “loop.jar” file into the newly created folder (you can find the “loop.jar” file in the “lib” folder inside the program’s installation folder). Next, edit the “build.gradle” file so it looks like in Fig. 18. Finally, do a gradle-refresh (Right click on the project in eclipse and then select “Gradle” → “Refresh Gradle Project”. Now the project setup is done and the implementation of the plugin can begin.



```
*build.gradle
1 plugins {
2     id 'java-library'
3 }
4
5 dependencies {
6     api 'org.apache.commons:commons-math3:3.6.1'
7
8     implementation 'com.google.guava:guava:23.0'
9
10    compile (name:'loop', ext:'jar') // ADD THIS LINE
11
12    testImplementation 'junit:junit:4.12'
13 }
14 repositories {
15     flatDir {
16         dirs 'libs' // ADD
17     } // THESE
18     // LINES
19     jcenter()
20 }
```

Figure 18: The population creation window.

Suppose we want to create a pair builder that greedily matches each agent to another agent that he would cooperate with. Therefore we create a new class called “GreedyPair-Builder” and let it extend the “PairBuilderPlugin” class. There are similar named classes

for all the other algorithms and mechanisms that can be implemented as plugins. Now we add the following code to the class (see Fig. 19).

```

14 public class GreedyPairBuilder extends PairBuilderPlugin{
15     @Override
16     public String getDescription() {
17         return "A greedy implementation of the cooperation considering pair builder";
18     }
19     @Override
20     public String getName() {
21         return "greedy pair builder";
22     }
23     @Override
24     public PairBuilder getNewInstance(List<Double> arg0) {
25         return new GreedyPB();
26     }
27     @Override
28     public List<Parameter> getParameters() {
29         return new ArrayList<Parameter>();
30     }
31     private class GreedyPB implements PairBuilder {
32         @Override
33         public List<AgentPair> buildPairs(List<Agent> agents, SimulationHistory hist) {
34             List<Agent> agentsMod = new ArrayList<Agent>(agents);
35             List<AgentPair> pairs = new ArrayList<AgentPair>(agents.size() / 2);
36             for (int i = 0; i < agentsMod.size() / 2; i++) {
37                 Agent firstAgent = agentsMod.get(i);
38                 int index = 1;
39                 for (int j = 1; j < agentsMod.size(); j++) {
40                     if (firstAgent.getStrategy().isCooperative(firstAgent, agentsMod.get(j), hist)) {
41                         index = j;
42                         break;
43                     }
44                 }
45                 pairs.add(new ConcreteAgentPair(firstAgent, agentsMod.get(index)));
46                 agentsMod.remove(index);
47                 agentsMod.remove(i);
48             }
49             return pairs;
50         }
51     }
52 }
53

```

Figure 19: The code for the greedy pair builder plugin.

After you finished your implementation of the plugin add the following folder structure to the project's root directory: "src/main/resources/META-INF/services". Inside the "services" folder add a new empty file named "loop.model.plugin.PairBuilderPlugin". Then add a single line to the file: The plugin's full class name including the package structure the class resides in. In our case the class is called "GreedyPairBuilder" and is placed in the "loop.extensions.plugins" package (see Fig. 20). So the line's content is "loop.extensions.plugins.GreedyPairB

```

1 package loop.extensions.plugins;
2
3 import loop.model.plugin.PairBuilderPlugin;
13
14 public class GreedyPairBuilder extends PairBuilderPlugin{
15

```

Figure 20: The package structure of the plugin.

Now you can run gradle build to generate the "GreedyPairBuilder.jar" file. You can find this file in the "build/libs" folder inside your project's root. Copy the "GreedyPairBuilder.jar" file into the plugins folder of the program's installation folder and your done. When you start the program the next time, your plugin will be loaded.