Compact User Guide of the Crystalline Simulator

J. Soler and and V. Sacristán

June 25, 2013

Contents

1 Introduction **3**

2 Program flow **3**

3 Program inputs **4**
3.1 Definition of the initial setting 6
3.2 Definition of the rules 7
3.2.1 The precondition 7
3.2.2 The action 10
3.2.3 Include rules 11
3.2.4 State colors 12
3.3 Program options 13
3.3.1 Layout options 13
3.3.2 Running options 14
3.3.3 Behavior options 15
3.3.4 Limitations 16

4 Program window **16**
4.1 Universe 16
4.2 Agents and rules 18
4.2.1 Agents panel 18
4.2.2 Rules panel 20
4.3 actions.log 22
4.4 positions.log 23
4.5 error.log 23
4.6 Agents generator 24
4.6.1 Agents panel 24
4.6.2 Universe panel 24
4.7 Exit menu 27
4.8 Universe menu 27
4.9 Agents menu 27
4.10 Rules menu 27
4.11 Agents generator menu 27

4.12 Options menu 28
4.12.1 Layout options 28
4.12.2 Running options 28
4.12.3 Behavior options 28
4.12.4 Limitations 28
4.13 Help menu 28

5 Limitations **28**

6 Program outputs **29**

7 Requirements **31**

8 Installation **31**

1 Introduction

This document describes the functionalities of the simulation program of distributed algorithms for Crystalline and Telecube robots. If refers to the CrystalSimulator version 1.0 program. It is a modification of an existing simulation program for modular robotic systems in square lattices named AgentSystem, therefore some of its functionalities are the same. Thanks are given to Birgit Vogtenhuber, Reinhard Wallner and Óscar Rodriguez for the square lattice Agent System simulator as well as for the structure of this user guide.

2 Program flow

This section is dedicated to explain how the system evaluates the precondition and the priority and how it executes the action of each module. Since the system supports priorities of the rules, the evaluation model is more complex than it would be without the usage of priorities. But priorities provide a helpful extension of the rules to enable advanced rule sets with simple rules.

The program starts by reading the initial set of modules as well as the set of rules. In every calculation step, the following operations are performed (in the order listed here; alternatively sending messages can be done before performing actions).

1. Check and get valid rules (each step for all modules)
   (a) check all rules whether they would apply (ignoring priorities) and store valid rules sorted by priority in descending order
   (b) store the highest priority of valid rules as current priority (if > 0) and set the modules priority to `PRIORITY_OPEN`
   (c) for all modules sorted by priority in descending order (if priority is `PRIORITY_OPEN`):
   do while not all modules have `PRIORITY_FIXED` :
      - fetch current rules to current priority
      - for all rules: check priority-conditions
        - if they are fulfilled, set priority to `PRIORITY_FIXED`
        - if one of them is not fulfilled, remove this rule from the specified module and if the modules rule list is empty, reduce the current priority to the highest priority of the remaining rules
        - if a circular dependency between rules on different modules is detected, remove all related rules

   (d) remove all rules with priority lower than the priority of the module
2. For all modules, for all previously stored rules for the module: perform from the actions (if they are part of the action):
   (a) detach
   (b) compute the attachment component

(c) move the attachment component (including collision detection test)

(d) update attachments

(e) update the state

(f) swap modules

(g) zip modules

(h) unzip modules

(i) swap-zip modules

3. Send and deliver messages

   (a) for all modules, for all previously stored rules for the module:

   - do all calculations (in the order they are listed in the rule) and send numerical messages to the post-office
   - send all text messages to the post-office

   (b) for all messages at the post office: deliver them to their recipients.

3 Program inputs

The program `CrystalSimulator.jar` uses three text files as input.
The file `files/config.txt` is a mandatory file. If it doesn't exist or an error on reading this file occurs, an error display is shown instead of the main window. A similar message appears if any file in the directory `img` does not exist.

The files `files/agents.txt` and `files/rules.txt` are the standard input files for the initial setting. If one of these files doesn't exist or doesn't match the specifications respectively described in sections 3.1 and 3.2 an error dialog occurs at the beginning. The dialog can be closed and necessary files can be chosen (see Section 4.2).

Syntax to launch the program:

```
java [parameters] CrystalSimulator.jar
```

The program can be alternatively started with commandline parameters. If input files are chosen with commandline parameters (arguments `-a` or `-r`), these files must exist instead of the files listed above. Available parameters are:

- `-a <agents_file>` this file is used instead of `files/agents.txt`
- `-r <rules_file>` this file is used instead of `files/rules.txt`
- `-n <number_of_steps>` performs after start the given number of steps with delay (see sections 3.3.2 and 4.1).
- `-nx <number_of_steps>` same as option `n` but additionally the program is stopped after performing the given number of steps.

- `-j <number_of_steps>` performs after start the given number of steps in jump mode (see Section 4.1).
- `-jx <number_of_steps>` same as option `j` but additionally the program is stopped after jumping the given number of steps.
- `-e` performs after start an endless run with delay (see sections 3.3.2 and 4.1).
- `-ex` same as option `e` but additionally the program is stopped if no further rule can be performed. Note that the endless run only stops if none of the preconditions of any rule is fulfilled. It doesn't stop if any rule tries to perform a step, but an error on performing occurs (see Section 4.1).
- `-s <agentsfile>` stores automatically the resulting modules configuration after performing multiple steps. Hence this argument is just allowed in combination with the `-nx, -jx` or `-ex` argument.
- `-h` help to commandline arguments

The file `files/config.txt` contains the options of the program (see Section 3.3). The file `files/agents.txt` contains the definition of the initial setting, meaning the initial positions, states, attachments, and counters of all modules (see Section 3.1). The definition of what an (arbitrary) module will do is contained in the file `files/rules.txt` (see Section 3.2). In all files, one-line-comments can be written by pre-pending the percentage-symbol (`%`) to the comment (meaning everything in the line after `%` is ignored). White spaces as well as empty lines, are ignored. Thus these two symbols may not be used as text. Further, for any kind of pattern matching, the asterisk (`*`) matches every character.

The error handling when starting the program is done in different ways because the program on the one hand can be used by manually starting and navigating it, or on the other hand it can be used by batch processing. To enable both cases with a maximum of usability, the error cases are handled as follows:

1. If the arguments contain syntax errors or irregular combinations of arguments are detected, a command line help with the information about the usage of the program is shown. The GUI does not start in that case.
2. If the program is started without any argument, but there exist errors in the modules or the rules files, then a dialog box appears after starting the graphical user interface with an appropriate message.
3. If the program is started with the arguments `-a` and/or `-r`, but no other parameters, and errors are found in the modules or in the rules files, the same behavior as in Case 2 appears.
4. If the program is started with the arguments `-a` and/or `-r`, but no other parameters, and any of the input files is not found, the GUI starts and an error message about this fact is displayed in a dialog box.

5. If any of the command line arguments `-nx`, `-jx` or `-ex` is used and any of the modules or the rules files is not found, then an error message is shown in the command line to ensure that a batch routine can proceed its work without interruption.

6. If the same arguments as in Case 5 are used and all files are found, but an error is detected in the modules or the rules file, then the program starts the GUI and shows the error to inform the user.

Moreover note that the usage of one of the arguments `-nx, -jx, -ex` has a side effect. When starting the program with one of those parameters, the `STOP` button is enabled to give the user the possibility to stop the process. But if the `STOP` button is pressed, then the program does not stop processing until the given number of steps are performed!

3.1 Definition of the initial setting

The initial setting is stored in the file `files/agents.txt`. Each line of the file defines one module by its initial (global) coordinates (mandatory) plus (optional) its state, its attachments and initializations for (some of) its counters. Please take into account that it is not possible to create a set of modules with compressions, i.e., in the file `files/agents.txt` each grid location can host at most one module. Optionally it is possible to change the size of the universe in the modules file with the parameter `U`.

Initial (global) position `x,y`
: The initial position is written as integer x-, y-coordinates, separated by a comma (see Section 3.3.4 for the range of coordinates).

State `S_____`
: The state of a module consists of exactly 5 characters, written with a leading S.
Default (if no argument given): `S*****` (no information).

Attachments `A____`
: The attachments of a module are written as A followed by 4 boolean values (0 for not attached, 1 for attached), in the order North, West, East, South.
Default (if no argument given): `A0000` (nothing attached)

Counters `C__ _____`
: Each module has 25 integer counters, `C00`,...,`C24`, which can be set to any 16-bit integer between -32767 and 32767 (-32768 is defined as no-number).
Default (if no argument given): any not explicitly initialized counter is set to zero.

Limitation of the universe size `U minX,maxX,minY,maxY`
: This parameter must be positioned at the beginning of the file. If this parameter is used the option `coordinates_range` of the file `files/config.txt` is overridden for this modules configuration (see Section 3.3.4)!

Small example:

```
U -5,20,20,0 % optional line
0,0 A0010 S1ST-- C0 -5 C1 17
1,0 A0100 S2ND--
3,0 SALONE
5,6
5,7
6,6 C0 1
7,3 C1 -234   C5 43   C7 999
```

3.2 Definition of the rules

The definition of what the modules will do is stored in the file `files/rules.txt`. Each rule definition consists of 4 lines:

1. the name of the rule
2. the priority of the rule
3. the precondition
4. the action

The name is a nonempty string. The priority is either a positive integer (at least 1 and at most 32767), where higher priority 'wins' lower priority, or it is written with the sign `$`. In the latter case the rule is a non-priority rule. The precondition defines whether or not a module may apply the rule. Finally, the action defines what has to be performed, if a module applies a rule.

3.2.1 The precondition

A module can apply a rule if it fulfills all the statements included in the precondition of the rule. The precondition may consist of the following parts, where each part is optional. In total, the precondition of a rule cannot be empty. Additionally each "sub-condition" of the precondition can be negated (see the end of this section).

Neighbors `N____`
: The definition of the direct neighbors (North, West, East, South). For each of them

  `0` denotes empty (no module),

  `1` denotes full (a module), and

  `*` denotes don't care.

  For example `N01*1` denotes that there must be an empty field to the North of the module, and the West and South fields need to be full.

Empty field `Edx,dy`, `EC__,dy`, `Edx,C__`
: An empty field requirement. Written as an E, followed by the relative coordinates of the field that needs to be empty, separated by a comma. Alternatively instead of each value `dx` or `dy` a name of any counter can be inserted, where a counter starts

with a `C`, followed by the two digit number of the counter. If a counter name is written, the value of the counter of the current module is inserted as this coordinate. Of course it is possible to mix numbers and counters.
Note that the usage of counter values for coordinates may lead to relative coordinates outside the universe. In that case a warning is shown but the rule is performed.

Filled field `Fdx,dy`
A full field requirement. The restrictions and the syntax are the same as in the "Empty field" condition.

Neighboring priorities `P____`
The priority of (the applied rule/s) of the direct neighboring positions (North, West, East, South). For each of them:

`<` denotes that the priority of the corresponding module needs to be strictly smaller,

`=` denotes smaller or equal, and

`*` denotes don't care.

Note that the priority of an empty field is always zero.

Remote smaller priority `Ldx,dy`
A strictly smaller priority field requirement. The L is followed by the relative coordinates of the modules that is required to have smaller priority, separated by a comma. The usage for counters is the same as in the "Empty field" condition.

Remote smaller or equal priority `Qdx,dy`
A less or equal priority field requirement. Syntax and usage is analogous to those of the "Remote smaller priority" condition.

Attachments `A____`
The attachment states to the direct neighbors (North, West, East, South), where

`0` denotes not attached,

`1` denotes attached, and

`*` denotes don't care.

For example `A0**1` denotes that the module must not be attached North and needs to be attached South (implying that a module to the South must exist).

State `S_____`
Every module has a state, consisting of exactly 5 characters. This state can be required to match a simple pattern, where an asterisk matches any character.
For example `SA***Z` denotes that the state of the module must start with `A` and end with `Z`.

State of a remote module `Tdx,dy,_____`

This option is a combination of the "Filled field" and the "State" preconditions. Written as a T, followed by the relative coordinates of the field that is required to be full, and ended by the state that the remote module must have. The usage of counters is the same as in the "Empty field" condition.

Incoming text messages

`M*_____`, `MN_____`, `MW_____`, `ME_____`, `MS_____`

Every module has four text messages from its direct neighbors (`*`=any, `N`=North, `W`=West, `E`=East, `S`=South), consisting of exactly 5 characters. Any of these messages can be required to match a pattern, where the asterisk matches any character but at most four asterisks are allowed.

Numeric comparisons on counters and incoming numeric messages

`<(-)____ (-)____`, `>(-)____ (-)____`, `=(-)____ (-)____`

Every module has 25 counters `C00`...`C24`, and additionally $4 * 8 = 32$ numeric messages from its direct neighbors:

- `#N01`, `#N02`, `#N03`, `#N04`, `#N05`, `#N06`, `#N07`, `#N08`, `#W01`, `#W02`, `#W03`, `#W04`, `#W05`, `#W06`, `#W07`, `#W08`, `#E01`, `#E02`, `#E03`, `#E04`, `#E05`, `#E06`, `#E07`, `#E08`, `#S01`, `#S02`, `#S03`, `#S04`, `#S05`, `#S06`, `#S07`, `#S08`
- Use `#*01`, `#*02`, `#*03`, `#*04`, `#*05`, `#*06`, `#*07`, `#*08` to check whether a numeric message is received from any direction at the specified channel.

Any of these numeric values can be required to fulfill a comparison with respect to any other such value or to any four digit number, where a negative number contains additionally a leading `-`.

Note that messages at different channels are independently from each other.

Remote numeric comparisons `Vdx,dy,C__ (-)____`, `Wdx,dy,C__ (-)____`

These options compare the first value with the second value:

`V`: the first value is strictly smaller than the second;

`W`: the first value is smaller or equal than the second.

The first numeric value is a counter from another module at relative coordinates `dx`, `dy`. It requires the module to exist. The second numeric value can be a counter, a numeric message from a neighbor or any four digit number, where a negative number contains additionally a leading `-`. Recall that every module has 25 counters and $4 * 8 = 32$ numeric messages from its direct neighbors (enumerated in the previous "Numeric comparison" condition).

Negation `!`

A negation can be applied to any part of the precondition. The negation means that the result of a "sub-condition" will be negated.

Parenthesis `( )`

To enable grouping of conditions as well as offering a disjunction between different conditions, parenthesis can be set around a group of conditions. It is also possible to interleave parenthesis (example: `!( N**1* !(A101* L-3,2) )`).

3.2.2 The action

The action of a rule defines what happens when a module applies the rule. It may consist of the following parts, were each part is optional (but in total the action cannot be empty).

Position change `Pdx,dy`
: Written as a P, followed by the relative coordinates of the field to which the module moves (separated by a comma). The usage for counters is the same as in the "Empty field" condition.

Attachments `A____`
: Attachment behavior for the four directions North, West, East, South (in this order), with the following possibilities for each direction:

  - `0` for detaching (if was attached) before moving, and staying detached afterwards
  - `1` for detaching (if was attached) before moving, and attaching afterwards (if possible)
  - `*` for detaching (if was attached) before moving, and attaching afterwards if was attached before (and possible)
  - `+` (only allowed if the module is attached at the beginning) staying attached all the time. If the module moves, every (directly or indirectly) attached module is moved in the same way.

  Default (if no argument given): `A****` (try attaching as before in all directions).

State `S_____`
: New state of the module, consisting of exactly 5 characters. An asterisk denotes that the according character remains unchanged.
  Default (if no argument given): `S*****` (no change).

Outgoing text messages
: `M*_____`, `MN_____`, `MW_____`, `ME_____`, `MS_____`
  Messages that are sent to neighbors (`*`=all, `N`=North, `W`=West, `E`=East, `S`=South), consisting of exactly 5 characters. At most four asterisks are allowed (for example `MSRUN**` is allowed, but not `MS*****`).

Assign calculation results to counters and outgoing numerical messages
: `C__ _ ____ ____`, `#___ _ ____ ____`
  `C__ _ dx,dy,C___ ____`, `#___ _ dx,dy,C__ ____`
  `C__ _ C__,C__,C___ ____`, `#___ _ C__,C__,C__ ____`
  Every calculation action starts with a position to write the result to (counter or outgoing message), followed by the operation to be performed, and two (readable) values on which the operation is performed.
  Possible operations are `+` (add), `-` (subtract), `*` (multiply), `/` (divide), `M` (modulo), `A` (maximum), and `I` (minimum).
  As values for an operation, either four-digit-numbers, or internal counters, or one remote counter, or incoming numerical messages can be used. Every module has 25 counters `C00` ... `C24` which can be read and written. Further, it has 32 (incoming)

numeric messages from its direct neighbors, which it can only read, and 32 numeric (outgoing) messages to its direct neighbors, which it can only write.
The remote counter is defined by first indicating the coordinates (`dx,dy` or `C__,C__`) of the module and then the counter to be used. Note that the coordinates are restricted to three digits (`-999` to `999`), and that it is required that there exists a module at the indicated location.
Available message senders/recievers:
`#N01`, `#N02`, `#N03`, `#N04`, `#N05`, `#N06`, `#N07`, `#N08`,
`#W01`, `#W02`, `#W03`, `#W04`, `#W05`, `#W06`, `#W07`, `#W08`,
`#E01`, `#E02`, `#E03`, `#E04`, `#E05`, `#E06`, `#E07`, `#E08`,
`#S01`, `#S02`, `#S03`, `#S04`, `#S05`, `#S06`, `#S07`, `#S08`
Use `#*01`, `#*02`, `#*03`, `#*04`, `#*05`, `#*06`, `#*07`, `#*08` to send a numeric message to all neighbors at the specified channel. Note that a calculation with undefined result (for example division or modulo by `0`) leads to an error. If such an error occurs no value will be sent to the receiver of the result.

Swap `XN`, `XW`, `XE`, `XS`
Written as a `X`, followed by the desired swap neighbor.

Zip `ZN`, `ZW`, `ZE`, `ZS`
Written as a `Z`, followed by the desired neighbor direction to zip the module into. Both the module applying the rule (which then becomes a compressed guest) and the module acting as host, must be uncompressed.

Unzip `zNS_____`, `zWS_____`, `zES_____`, `zSS_____`
Applies to a module compressed and hosting another compressed guest. Written as a `z`, followed by the desired neighbor direction to unzip the guest module to, together with the new state (with the same format as `state`) of the guest module, once uncompressed.

Swap-zip `xN`, `xW`, `xE`, `xS`
Applies to a module compressed and hosting a compressed guest. Written as a `x`, followed by the desired neighbor direction to send the compressed guest to. The module receiving the guest must be uncompressed.

3.2.3 Include rules

To enable combinations of different rule sets for one module configuration, it is possible to include other rule files. The syntax for an include is

`include:<file_path>:<Priority_Offset>`

where include statements must be added at the beginning of the file. If just a file name is given the file will be searched in the `files` directory of the program. If a relative path definition is given in the include statement, the path separator sign must be chosen from the user according to the operating system. Note that spaces are not allowed in the path according to Section 3. It is possible to insert as many include statements as desired and it is also possible to cascade include statements. If include statements are cascaded priority offsets will be accumulated.

If an error in any of the included files occurs, an error message is shown in the specified line of the main rules file. Additionally an error message occurs if the include statement is behind the init block (see Section 3.2.4) or any defined rule (see Section 4.2.2).

For example `include:foreign_rules.txt:100` means that all rules from the file `foreign_rules.txt` are included in the ruleset, where the priority of each rule defined in `foreign_rules.txt` will be incremented by `100`.

Note that the usage of include statements can simplify the definition of complex rulesets. But this can lead to some problems, for example the usage of the same state name in different files or the usage of same counters in different files, what definitively would lead to undesired behavior.

3.2.4 State colors

A helpful function for visualization is the color of modules. The standard color for attached and detached modules can be defined in the file `files/config.txt` file or alternatively in the options menu. But in the visualization it is also essential which state a module at a specified step has. Of course it is possible to check the state in the tooltip function (see Section 3.3.1). But to get an overview about the state of all modules this would need a lot of time. Therefore a color can be assigned to each state of the modules.

Assigning colors to states can be made in the current rules file. In order to do so an init block is inserted in the rules file. The init block needs to be located after the include lines and before the rules. In the init block each line assigns a color to a state. Assigning a color to a state can be made in two different ways. The first possiblity is to write the name of the state followed by the name of the color, where possible colors are `BLACK`, `BLUE`, `CYAN`, `GRAY`, `MAGENTA`, `ORANGE`, `PINK`, `RED`, `YELLOW`. The names are listed additionally in the help menu (described in Section 4.13). An alternative option is to write the specified state followed by a `#` and the hexadecimal `RGB` color code.

The following is an example of a rules file:

```
include:foreign_rules.txt:50

_init_start_
SRIGHT CYAN
SDOWN_ #ff0000 %means the same as SDOWN_ RED
_init_end_

Go Down
100
SDOWN_ >C000000
P0,-1 C00-C000001 A++++
```

Note that color names are written with capital letters and that the hexadecimal code uses lower case letters. Further in the hexadecimal code the first two digits stand for the red color, the middle two digits for the green color and the last two digits for the yellow color. If a module gets a state where no color definition is made in the rules file, the module gets the standard color as defined in the options (see Section 3.3.1). If any error on reading the state colors is found, an error message is written in the specified line (see Section 4.2.2).

3.3 Program options

The definition of available options is stored in the file `files/config.txt`. All options that are available can be set in the file or in the Options menu(see Section 4.12). The syntax of options is always the same: `[OPTION_NAME]=[OPTION_VALUE]`.

3.3.1 Layout options

Available layout options are color options, font options [and the grid option].

1. Color options

   The color option value has the syntax: `[R],[G],[B]` whereas `R` means the red color value, `G` means the green color value and `B` means the blue color value. The values for `R`, `G` and `B` can be chosen in the range 0-255.

   Example:

   ```
   color_bg=255,255,255
   ```

   The following colors can be modified:

   Background color Color of the universe background.

   - Option name: `color_bg`
   - Standard value: `[255,255,255]`: (White)

   Grid color Color or the universe grid.

   - Option name: `color_grid`
   - Standard value: `[0,0,0]`: (Black)

   Color of not attached agent Color of modules who are nowhere attached, and for attachment arms which aren't attached.

   - Option name: `color_agent_not_att`
   - Standard value: `[0,255,0]`: (Green)

   Color of attached Agent Color of modules who are anywhere attached.

   - Option name: `color_agent_att`
   - Standard value: `[0,124,0]`: (Dark Green)

   Color of attached arm Color of attachment arms which are attached.

   - Option name: `color_attached_arm`
   - Standard value: `[255,0,0]`: (Red)

   Standard text color Color of the iteration number in the universe.

   - Option name: `color_text`
   - Standard value: `[0,0,0]`: (Black)

   Error text color Color of the error message text in the universe.

   - Option name: `color_error_text`

- Standard value: `[255,0,0]`: (Red)

Color of tooltip background Color of the tooltips background.

- Option name: `color_tooltip`
- Standard value: `[255,255,0]`: (Yellow)

Color of tooltip text Color of the tooltips text.

- Option name: `color_tooltip_text`
- Standard value: `[0,0,0]`: (Black)

2. Font options

   Font options are font type and font size for the modules, rules, actions, positions, and error panels.

   Font type Type of the font in the modules, rules, actions, positions and error panel.

   - Option name: `current_font`
   - Standard value: `[SansSerif]`.

   Font size Size of the font. Available range is from 6 to 32 dots per inch.

   - Option name: `current_font_size`
   - Standard value: `[12]`.

3.3.2 Running options

These options are used for better usability and performance.

1. History options

   The program has a history option to provide reverse steps.

   - Option name: `history`
   - Available values:
     - `all`: All iterations are stored in the history. Take into account, though, that the system is limited to a maximal stored number of history entries (see Section 5).
     - `last20`: The last 20 iterations are stored in the history (at most 20 back steps are possible).
     - `off`: No history entries are stored. Thus no back steps are possible.
   - Standard value: `all`

2. Logging options

   For details to the log files see Section 6.

   - Option name: `logging`
   - Available values:

- `all`: All log files are written.
- `error`: Just the error.log file is written.
- `off`: No log files are written.

- Standard value: `all`

3. Delay options

   Delay between multiple steps (10 Steps Back, Next 10 Steps, Next n steps, Run forever; see Section 4).

   - Option name: `timeout`
   - Available values:
     - The timeout in milliseconds.
   - Standard value: `500`

4. Rules warning option

   This option enables or disables the warning for duplicate rule names (see Section 4.2.2).

   - Option name: `rules_warning`
   - Available values:
     - `on`: Rules warnings are shown in rules panel.
     - `off`: Rules warnings are not shown in rules panel.
   - Standard value: `on`

3.3.3 Behavior options

These options are concerning the actions in each iteration.

1. Agent options

   This option takes effect on applying rules on each module. In the standard way all rules which can be applied to one module are performed in the order they are listed in the rules file `files/rules.txt`. This option allows to change this in the following way: rules which don't have an action including a position change can be applied before rules which imply a position change. This option has an effect on each module, but does not influence the global order of the rules performance: rules are always applied to the modules in the order modules are listed in `files/agents.txt`.

   - Option name: `position_changes_last`
   - Available values:
     - `on`: All stored rules of a module implying a position change are performed after all rules without position changes.
     - `off`: All rules are applied in the order listed in the rules file.
   - Standard value: `off`

2. Message delivering option

   In the standard way messages are sent and delivered after performing all other actions. With this option this can be changed so that after checking all rules all messages are sent and delivered and after that all remaining actions are performed.

   - Option name: `perform_messages_first_move_last`
   - Available values:
     - `true`: All messages are sent before performing all actions to all modules.
     - `false`: All messages are sent after performing all actions to all modules.
   - Standard value: `false`

3.3.4 Limitations

Limitations to the universe range can be set. Attention: This option does not have any effect if the parameter `U` is used in the modules file (see Section 3.1).

1. Coordinates range

   This option has the syntax: `min_X, max_X, min_Y, max_Y`, where the values must be integers.

   - Option name: `coordinates_range`
   - Example:
     - `coordinates_range=-5,70,-5,27`
   - Standard value: `-5,70,-5,27`

4 Program window

The main window consists of a menu and a tabbed panel with five tabs. The
Short key: `'Ctrl'+'T'` allows to navigate to the next tab, and the
Short key: `'Ctrl'+'Shift'+'T'` sends to the previous tab.

4.1 Universe

This panel visualizes all read modules in the universe. In the upper side of the universe panel the iteration is shown and if an error occurs a message is shown, too. If more than one error appears in one iteration a `+` sign is added in front of the error field. Clicking on the `+` sign opens a window with all error messages appeared in this iteration, where the first error is on the first position in the window. If more than 10 errors appears, an additional line

`< < < < Further errors occurred! For details see error.log! > > > >`

is shown in the last line of this window. Furthermore, the number of `TotalErrors` and `TotalWarnings` is shown in the upper side of the universe panel.

Several navigation buttons are available (back steps are only available if the history function is on, see Section 3.3.3). For multiple steps buttons (10 Steps Back, Next 10

Steps, Next n steps, Run forever) a timeout between the steps is passed (see Section 3.3). Alternatively for each button at least one short key is available.

Universe panel functions:

Start Before iterations can be performed this button has to be clicked.
Short keys: `'s'`, `'S'`.

Reset A click on this button resets the configuration to its initial values. This means that modules and rules are read once again and all history entries are removed.
Short key: `'F2'`.

Back to start A click on this button resets the universe to its initial position (Iteration0)
Short key: `'Home'`.

10 Steps Back With this button 10 reverse steps are performed.
Short keys: `'PgUp'`, `'CursorUp'`.

Back Just to go one step back.
Short keys: `'CursorLeft'`, `'Backspace'`.

Next Step Go one step forward.
Short keys: `'CursorRight'`, `'Enter'`.

Next 10 Steps Go 10 steps forward. If no further rule can be performed a message is shown.
Short keys: `'PgDown'`, `'CursorDown'`.

Next n steps A click on the button opens a dialog where the number of steps to move can be introduced. The dialog has a text field to insert the desired number of steps, a button `Next` to perform the chosen number of steps and a button `Cancel` to close the dialog without performing steps. After clicking on `Next` the steps are performed. If no further rule can be performed a message is shown.

Jump n A click on this button performs a chosen number of steps without visualization and without timeout (see Section 3.3.2, item 3). After clicking on the button a dialog appears. The dialog has a text field to insert the desired number of steps, a button `Jump` to perform the chosen number of steps and a button `Cancel` to close the dialog without performing steps. After clicking on `Jump` the steps are performed. To update the user about the remaining steps a message `Wait n steps` is shown on the top of the universe panel (where n indicates the remaining number of steps). If no further rule can be performed a message is shown.
Short keys: `'j'`, `'J'`.

Run forever Performs endless forward steps. If no further rule can be performed a message is shown.
Short key: `'End'`.

Stop Stops executing steps (10 Steps Back, Next 10 Steps, Next n Steps, Run forever).
Short keys: `'Esc'`, `'Pause'`.

Tooltip function A helpful function to check the current values of each module is the Tooltip function. A click on a module shows a tooltip window.

The information displayed on this windows is:

always shown: (on each module)

Id
Global position
Attachments
Counter values
Current priority

optionally shown (if information is stored)

State
Text messages
Numeric messages
Id zip
State zip
Zip counter values

When clicking on a grid position containing two compressed modules, the tooltip window shows the information for both the host and the guest modules, in this order.

Speed This function is only available with the
Short key: `'Ctrl'`.
If multiple steps are performed a timeout between steps is passed. To avoid this, the timeout can be disabled. But this can be done in another way, too: while multiple steps are performed the Short key `'Ctrl'` can be hold and the timeout while holding the key is 0.

4.2 Agents and rules

The tab consists of two text panels. On the left side the current modules file is shown and on the right side the current rules file is shown. To enable changing focus between the two text panels if editable, the mouse or alternatively the short key: `'Ctrl'+'P'` can be used. If just one text panel is editable the focus grabs to that panel.

The font type and size of the modules and rules panels as well as of the actions, positions and error panel can be modified in the options menu (see Section 3.3.1).
Short key: `'Ctrl'+'+'` increments the font size.
Short key: `'Ctrl'+'-'` decrements the font size.

4.2.1 Agents panel

The panel has on the upper side buttons to create, open and modify modules files. The text panel contains the current modules file. Lines with successfully read modules (valid initial setting) are shown in black, comment lines are shown in gray and lines with errors

are shown in red. A module definition is valid if and only if all settings in the specified lines fulfill the specification. For further information see Section 3.1.

Possible error messages:

- `"% AGENT already exists! Ignored!` occurs if a module on this position is already stored.
- `"% Position OUT_OF_RANGE! Ignored!"` is shown if a position value of a module isn't in the coordinate range of the universe (see also Sections 5 and 3.1).
- `"% % Counter OUT_OF_RANGE! Ignored!"` occurs if any counter value inserted is out of the specified range (see Section 3.1).
- `"% ATTACHMENTS ERROR: not symmetrical! Attachment ignored!"` is displayed if an attachment definition between two modules exists that is not symmetrical.
- The message `"% ATTACHMENTS ERROR: to attach agent doesn't exist! Attachment ignored!"` shows that an attachment definition refers to a non existent module.
- `"% SYNTAX ERROR OR IRREGULAR LIMITATIONS! Ignored!"` occurs if a syntax error on a module definition or the universe limitation is found, or if the universe limitation is not at the beginning of the file.

New To create a new modules file. After clicking on the button the text panel will be cleared. Modules can be defined and saved. To interrupt the process click on Reset. To store the new setting click on Save.
Short key: `'Ctrl'+'N'`.

Open Opens a modules setting file. A dialog to save all log files appears before opening the file (see button Save) because when opening the modules file the log files are overwritten.
Short key: `'Ctrl'+'O'`.

Modify With this button the current modules setting can be modified.
Short key: `'Ctrl'+'M'`.

Reset If a new modules setting is created or an old one is modified, this button discards the new setting (the last read modules become active).
Short key: `'Ctrl'+'R'`.

Save To save an existing modules setting. If the button is clicked all log files are overwritten and the new modules setting is read immediately. No dialog appears to store existing log files!
Short key: `'Ctrl'+'S'`.

Save as To save a new modules setting or a modified one. If this button is clicked all log files are overwritten and the new modules setting is read immediately. A dialog to save all log files appears before loading the new setting.
Short key: `'Ctrl'+'W'`.

Undo If the modules file is modified in the agents panel this function undoes the last modification. If modifications in the rules panel are done too, the current focus decides on which panel the function is performed. This can be done via the menu or the
Short key: `'Ctrl'+'Z'`.

Redo If the modules file is modified in the agents panel and undo steps are performed this function redoes the last undone operation. If modifications in the rules panel are done too, the current focus decides on which panel the function is performed. Redo can done via the menu or the
Short key: `'Ctrl'+'Y'`.

Find If the modules file is editable in the agents panel a find dialog is opened with the
Short key: `'Ctrl'+'F'`.
Available options in this dialog are search direction, search scope, wrap search and case sensitive search. After clicking `Find` the dialog will be closed and eventually the found text will be marked. The opened dialog can be canceled with the short key `'Esc'` and the find process can be started with the short key `'Enter'`, too. If the rules file is also editable, the current focus determines on which panel the find process takes effect.

Find next The next occurrence of an already searched text can be found with the
Short key `'F3'`.
Search options are the same as for "Find".

Replace A replace dialog for the agents panel can be opened with the
Short key `'Ctrl'+'H'`.
Finding options are the same as in the "Find" dialog. The dialog is opened until the `Close` button or the short key `'Esc'` is pressed. The button `Find` or the short key `'Enter'` searches the next occurrence in the text; the button `Replace` replaces the first occurrence of found text after the current cursor position; the button `Replace and Find` replaces the first occurrence and searches the next occurrence of the searched text. Finally the button `Replace All` replaces all occurrences of the searched text. If the rules file is also editable, the current focus determines on which panel the find and replace process takes effect.

4.2.2 Rules panel

The panel has on the upper side buttons to create, open and modify rules files. The text panel contains the current rules. Rules with successfully read priority, preconditions and actions are shown in black, comment lines are shown in gray and rules with errors are shown in red. A rule definition is valid if and only if all settings in the specified lines fulfill the specifications. For further information see Section 3.2.

Possible error messages:

- If an error on reading an include file is found the message `"% Error: Specified file not found or error in nested file! Please check it!"` is displayed. Possible causes are: the file doesn't exist; the path to the file contains spaces,

which is not allowed; some rule in some include file (they can be nested) contains irregular expressions or some file contains an irregular include statement. It would be possible that a negative priority offset causes problems, too.

- If an include statement is found at an irregular position the message `"% Error: Include statement only allowed at the beginning of the file!"` is shown.
- If an irregular syntax in an include line or in any of the module colors is found the message `"% Syntax error!"` is shown.
- If the priority isn't an integer > 0 the message `"% Wrong syntax! Rule ignored!"` is displayed in the second line of the rule.
- `"% Irregular Precondition! Rule ignored!"` shows that the precondition contains either a syntax error or that an `Empty Field`, `Filled Field`, `Smaller Priority`, `Smaller or Equal Priority` or `State of a remote agent` refers to a position outside the universe.
- `"% Irregular Precondition! Priority conditions and NonPriority conditions are mixed! Rule ignored!"` is shown if a NOT condition (`!`) contains priority and non priority conditions (see Section 3.2.1).
- `"% Irregular Action! Rule ignored!"` occurs if a syntax error in the action line is found or if a `Position change` would lead to move outside the universe.

Possible warnings (can be disabled, see Section 3.3.2):

- `"% WARNING: DUPLICATE RULE NAME!"`
  shows that a rule with the same name has already been defined.

In the rules panel, the following options can also be found:

New To create a new rules file. After clicking on the button the text panel will be cleared. Rules can be defined and saved. To interrupt the process click on Reset. To store the new definitions click on Save.
Short key: `'Ctrl'+'Shift'+'N'`.

Open Opens a rules file. A dialog to save all log files appears before opening the new definition (see button Save) because when the rules file is opened, the log files are overwritten.
Short key: `'Ctrl'+'Shift'+'O'`.

Modify With this button the current rules file can be modified.
Short key: `'Ctrl'+'Shift'+'M'`.

Reset If a new rules definition is created or an old one is modified this option discards the new definition, and the last read rules become active.
Short key: `'Ctrl'+'Shift'+'R'`.

Save To save an existing rules definition or a modified one. If this button is clicked all log files are overwritten and the new rules file is read immediately. No dialog appears to store existing log files!
Short key: `'Ctrl'+'Shift'+'S'`.

Save as To save a new rules definition or a modified one. If this button is clicked all log files are overwritten and the new rules file is read immediately. A dialog to save all log files appears before loading the new definition.
Short key: `'Ctrl'+'Shift'+'W'`.

Undo If the rules file is modified in the rules panel this function undoes the last modification. If modifications in the agents panel are done, too, the current focus decides on which panel the function is performed. Undo is possible via the menu or the
Short key: `'Ctrl'+'Z'`.

Redo If the rules file is modified in the rules panel and undo steps are performed, this function redoes the last undone operation. If modifications in the agents panel are done too, the current focus decides on which panel the function is performed. Redo can be done via the menu or the
Short key: `'Ctrl'+'Y'`.

Find If the rules file is editable in the rules panel a find dialog is opened with the
Short key: `'Ctrl'+'F'`.
Available options in this dialog are search direction, search scope, wrap search and case sensitive search. After clicking `Find` the dialog will be closed and eventually the found text will be marked. The opened dialog can be canceled with the short key `'Esc'` and the finding process can be started with the short key `'Enter'`, too. If the modules file is also editable, the current focus determines on which panel the find process takes effect.

Find next The next occurrence of an already searched text can be found with the
Short key `'F3'`.
Search options are the same as for "Find".

Replace A replace dialog for the rules panel can be opened with the
Short key `'Ctrl'+'H'`.
Options are the same as for "Find". The dialog is opened until the `Close` button or the short key `'Esc'` is pressed. The button `Find` or the short key `'Enter'` search for the next occurrence in the text; the button `Replace` replaces the first occurrence of found text after the current cursor position; the button `Replace and Find` replaces the first occurrence and searches for the next occurrence of the search text. Finally the button `Replace All` replaces all occurrences of searched text. If the modules file is also editable, the current focus determines on which panel the find and replace process takes effect.

4.3 actions.log

This panel shows the log file `files/actions.log` (see Section 6). The panel contains in its upper side a combo box to select the iterations to be visualized. If an iteration range

is performed more than once a message `later iterations:` is inserted to the text panel.

Let us examine an example. First the iterations 1-48 are performed. Then 20 back steps are done, and finally the iterations 29-42 are performed. If now on the actions panel the set `Iteration 40-49` is selected, the following lines are shown:

```
% Iteration 40
  Agent 1 (28/9): DOWN
% Iteration 41
  Agent 1 (28/8): RIGHT
% Iteration 42
  Agent 1 (29/8): RIGHT
% Iteration 43
  Agent 1 (30/8): RIGHT
% Iteration 44
  Agent 1 (31/8): RIGHT
% Iteration 45
% Iteration 46
  Agent 1 (32/8): RIGHT
% Iteration 47
  Agent 1 (33/8): RIGHT
% Iteration 48
  Agent 1 (34/8): RIGHT
-------------------------------------------------------
 later iterations:

% Iteration 40
  Agent 1 (28/9): DOWN
% Iteration 41
  Agent 1 (28/8): RIGHT
% Iteration 42
  Agent 1 (29/8): RIGHT
```

Combo box iteration n-m Selects the shown iteration range.

Save as Button to store the `files/actions.log` file to another file.
Short key: `'Ctrl'+'S'`.

4.4 positions.log

This panel shows the log file `files/positions.log`, which stores the situation of all modules at each iteration (for more details on the content of this file, see Section 6). The handling is analogous to the one described in Section 4.3.

4.5 error.log

This panel shows the log file `files/error.log`, which stores all warning and error messages (see Section 6 for more details on the content of this file). The handling is analogous to the one described in Section 4.3.

4.6 Agents generator

The agents generator is dedicated to create and modify sets of modules. This tab shows on the left side a text panel, which shows the modules file, similar to the text panel for the agents tab (compare with Section 4.2.1). On the right side of the tab a graphical representation of the loaded set of modules is shown. When the program starts, the modules generator loads the same file as the universe. Nevertheless, the modules generator is independent from the current universe. This means that the modification of module sets does not influence the current setting on the "Universe" tab. Furthermore the text in the left panel and the graphical representation to its right are equivalent, i.e. a text/graphical representations of the same file. An exception to this fact happens if the content of the text panel is modified by hand; for details see the "Refresh" option of the Agent panel in Section 4.6.1.

4.6.1 Agents panel

The left part of the agents generator contains the text panel, which has the same functionality as the agents panel on the "Agents and rules" tab. In this case, though, the modules are shown on the universe of the creator panel instead of the current universe. Moreover the dialog to save log files does not appear in the agents panel of the modules creator if, for example, modules are saved. The "Undo" and "Redo" functions work a little bit differently from the functions described in Section 4.2.1 (see more details in Section 4.6.2). Additionally the following two options are included on the agents panel of the modules creator:

Save + init On clicking this button the current modules setting will be stored and shown in the creator universe.
Short key: `'Ctrl'+'I'`.

Refresh If the agents panel is modified, the graphical presentation needs to be updated through this refresh option. On refreshing the text panel, syntax errors are checked, too.
Short key `'F5'`.

4.6.2 Universe panel

The universe panel can be used to create and modify sets of modules in a visual way, by using the mouse of the computer. Note that all functions to change the modules set in the universe panel are only available if the modules file in the agents panel is editable. All modifications on the modules are refreshed immediately in the text panel to the left of the universe panel. Contrary to this, any modification in the agents panel is only updated in the universe panel when using the "Refresh" function.

Available functions on the universe panel are "Add agents", "Remove agents", "Select agents and set their state" and "Change the values of counters on selected agents" as well as the "Undo" and the "Redo" function. Note that most of the actions are done by using the left mouse button (if the user is right handed, else the right mouse button)! For the "Add", "Remove" and "Select" functions the following modes are possible:

MODE: Single agent

If this option is selected, all actions are applied on just one single module (for example, a module is added to the universe).

MODE: Line of agents 

If the "Line mode" is chosen, an "Add", "Remove" or "Select" action is applied on all modules which are within the drawn line. If, for example, the "Add" action and the "Line mode" are selected, then a mouse click starts creating a line of modules. When the mouse is dragged (by holding the left mouse button), a line of modules is created, from the middle of the starting point to the current position of the mouse pointer. When the left mouse button is released, all modules are added to the universe.

MODE: Rectangle of agents 

The "Rectangle mode" works similar to the "Line mode": on dragging the mouse left button, a blue rectangle is drawn. On releasing the left mouse button, all cells enclosed in the drawn rectangle contain a module which is added to the universe.

In the following the three actions "Add", "Remove" and "Select" modules are described in detail:

ACTION: Add agent 

If this option is selected, modules are added to the current universe. To show which mode is selected when adding modules, a blue line in the line mode and a blue rectangle in the rectangle mode is drawn when dragging the mouse. Note that adding modules does not change any state or counter value of any previously added module! After adding modules, the cursor in the text panel is set to the module on the end point of the line or rectangle. Furthermore, several options for attachments on adding modules are available (see the options below).

ACTION: Remove agents 

This is the reverse of the adding action, and works similarly. Either a single module, a line of modules, or a rectangle of modules can be removed from the universe. The line and the rectangle on dragging the mouse is drawn in red color.

ACTION: Select agents 

This option allows to choose a set of modules and further to modify their state and the value of their counters(see the actions "Change state" and "Change counters"). When dragging the mouse, a gray line is drawn if the line mode is chosen, and a gray rectangle if the rectangle mode is selected. Modules are selected by using the left mouse button and unselected by using the right mouse button (if the user is right handed, else the inverse). Selected attached modules are drawn in red color, selected detached modules in orange. Furthermore every selected module is surrounded by a gray rectangle. Additionally it is possible to select groups of modules by holding

the Short key `'Ctrl'`!. If the `'Ctrl'` key is hold, and new modules are selected, the old selection is extended by the new selection, otherwise the old selection is replaced by the new selection. Note that is also possible to change the "mode" of selection without losing the current selection, because the selection is only removed if a new action is performed or the modules are stored or reset.

When adding modules to the universe, the new modules can be either detached from all their neighbors, or they can be attached only to the currently added modules. Finally, they can also be attached to all existing neighbors. Therefore, adding modules can overwrite attachments of existing modules!

OPTION: Detach all when adding 

If the "Add" action and the "Detach" option are chosen, all new modules are detached from all their neighbors, meaning that attachments of previously existing modules are removed.

OPTION: Attach along cursor when adding 

This option attaches all currently added modules to each other, but not with previously existent modules. Recall, though, that if new modules are added in the same positions where previously added modules were already present, no attachments are removed, but new attachments can be added.

OPTION: Attach all when adding 

All currently added modules are attached to all existing neighbors, including to previously added ones. Note that attachments of previously added modules are overwritten, if new modules are added on the same positions by using this option.

ACTION: Change state 

The state of all selected modules can be modified by using this action. A dialog box to insert the new state is shown. The Short key: `'Enter'` sets the new state if it is possible (equivalent to the `OK` button) and closes the dialog, and the Short key: `'Esc'` closes the dialog without changing the state of any module.

ACTION: Change counters 

The values of any counters of all selected modules can be changed by this action. A click on this button opens a dialog. Either all counters or a selection of counters can be selected, and their new values can be set. The new counter value can be inserted in a text field. The correctness of the inserted values is checked before changing the value. The Short key: `'Enter'` sets the new counter values if it is possible (equivalent to the `OK` button), and the Short key: `'Esc'` closes the dialog without changing any value.

ACTION: Undo 

This option works for all actions which are performed on the creator universe ("Add"

or “Remove” modules, “Change state” or “Change counters”). The “Undo” operation of the text panel to the left of the universe is only stored if the “Refresh” function of the text panel is applied. For example, if a set of modules is loaded, then a module is removed from in the text panel and finally the key `'F5'` is pressed, then the “Undo” operation works as desired. If changes are performed in the text panel without “Refreshing” the universe, and then an action is done using the mouse, then all modifications done in the text panel (previous to the mouse action) are lost!
Short key: `'Ctrl'+'Z'`

ACTION: Redo 
The “Redo” operation works if any “Undo” operation is performed prior. Short key: `'Ctrl'+'Y'`

The colors used to display modules are the same as the ones used in the original universe. This means that the “standard” colors for modules are green, and dark green for attached modules (if the colors of the modules are not changed in the options of the program). If a rules file is loaded containing color definitions for states, then the colors defined in the rules file are used. An exception to this is the selection of modules, which always uses the colors orange (for unattached modules) and red (for attached modules).

4.7 Exit menu

To end the program.

4.8 Universe menu

This menu contains all functions of the universe panel. For details see Section 4.1.

4.9 Agents menu

This menu contains all module functions of the panel “Agents and Rules”. More details can be found in Section 4.2.1.

4.10 Rules menu

This menu contains all rules functions of the panel “Agents and Rules”. More details can be found in Section 4.2.2.

4.11 Agents generator menu

See Section 4.6.

4.12 Options menu

After clicking on this menu a dialog with option values is opened. The values which can be set are described in Section 3.3. In the dialog the options can be modified and stored. Modifications can be discarded and options can also be reset to their standard values.

4.12.1 Layout options

Available layout options, color options [and the grid option]. In contrast to the color options in the file, in the panel the colors can be chosen from a color chooser dialog without knowledge about RGB-values. See Section 3.3.1.

4.12.2 Running options

Described in Section 3.3.2.

4.12.3 Behavior options

Described in Section 3.3.3.

4.12.4 Limitations

Limitations to the universe range are available. See Section 3.3.4 for details.
Attention:

- If new limitations are stored, the current universe state is invalid. Thus all modules and rules must be read once again and all log files are overwritten. To avoid data loss a dialog window to save all log files is shown.
- This option does not have any effect if the parameter `U` is used in the modules definition (see Section 3.1).

4.13 Help menu

This menu contains an entry `Shortkeys` which gives an overview of available short keys. "Color names for states" lists all available names, which can be used in the rules file (refer to Section 3.2.4). Finally it contains an entry `About`.

5 Limitations

The system has the following restrictions.

- The program needs a minimal display resolution of 800×600 pixels. If the display width is less than or equal to 1024 the buttons of the universe panel are arranged in two raws. If the width is greater than 1024 the buttons are arranged in one row.

- The duration of one step in the program differs on different modes. The fastest mode is the jump-mode because only the last step is shown. Multiple steps forward (Next 10 steps, Next n steps, Run forever) in speed-mode (see Section 4.1) are almost as fast as jump steps. Single steps forward are a little bit slower than multiple steps. Backwards steps are the slowest steps, because all modules are displayed at each step.

- If the option History (Section 3.3.2) is set to `all` the application is limited because of limited memory resources. If resources are exhausted an `OutOfMemory` error occurs. In this case an error dialog is shown. On the dialog the button `End Program` exists to end the program.

  On Windows XP with a standard stack size of 64MB this limit is reached with approximately 35000 modules stored in the history. This can be reached for instance with 200 modules in 175 iterations if all modules act at each step.

  The size of the stack can be set manually with a JVM-Flag `-Xmx[n]m` whereas the value for n is the stack size in MB.

6 Program outputs

In addition to the graphical representation, the program generates three output files:

- `files/positions.log`
- `files/actions.log`
- `files/error.log`.

For better readability of these files, the modules are given numerical id's according to the order they appeardefinition in the file `files/agents.txt`, starting with number 1.

The file `files/positions.log` contains a complete state history of the modules. Starting from the initial setting, and then after every (global) calculation step, all modules are listed in the order of their numerical id's. For each module one line is printed, containing its numerical id, its current global position, its attachments, its state, the values of all its counters, and all current incoming messages (numerical and text).

Example with numerical messages:

```
% Iteration 4
  1: 0/5 A0010 SLSent C0 0 C1 0 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
  2: 1/6 A0011 SLSent C0 0 C1 0 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
  3: 2/6 A0101 SFORWD C0 0 C1 2 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
  4: 1/5 A1110 SFORWD C0 1 C1 2 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
  5: 2/5 A1110 SUNDEF C0 0 C1 0 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0 #N2 #W2
  6: 3/5 A0101 SUNDEF C0 0 C1 0 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
  7: 3/4 A1010 SUNDEF C0 0 C1 0 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
  8: 4/4 A0100 SUNDEF C0 0 C1 0 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
```

Example with text messages:

```
% Iteration 16
  1: 0/5 A0010 SLSent C0 0 C1 0 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
  2: 1/6 A0011 SLSent C0 0 C1 0 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
  3: 2/6 A0101 SFORWD C0 0 C1 2 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0 MSWRONG
  4: 1/5 A1110 SFORWD C0 1 C1 2 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0 MEOK___
  5: 2/5 A1110 SFINIS C0 2 C1 3 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
  6: 3/5 A0101 SFINIS C0 0 C1 4 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
  7: 3/4 A1010 SFINIS C0 4 C1 0 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
  8: 4/4 A0100 SFINIS C0 0 C1 5 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
% Iteration 17
  1: 0/5 A0010 SLSent C0 0 C1 0 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0 MEOK___
  2: 1/6 A0011 SLSent C0 0 C1 0 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0 MEWRONG MSWRONG
  3: 2/6 A0101 SFINIS C0 0 C1 2 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
  4: 1/5 A1110 SFINIS C0 1 C1 2 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
  5: 2/5 A1110 SFINIS C0 2 C1 3 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
  6: 3/5 A0101 SFINIS C0 0 C1 4 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
  7: 3/4 A1010 SFINIS C0 4 C1 0 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
  8: 4/4 A0100 SFINIS C0 0 C1 5 C2 0 C3 0 C4 0 C5 0 C6 0 ... C24 0
```

The file `files/actions.log` contains a complete history of which module applied which rules, and when. In every global calculation step, the printing is done after the decision is taken of which modules will perform which rules, but before the rules are applied. For every module and every applied rule one line is printed, containing the numerical id of the module, its current position, and the name of the rule to be applied.

The following example shows an `actions.log` file. In iteration 3 module 4 applies two rules. The possibility of performing multiple rules on one module is a very powerful tool, which allows simplifying the rules set. It must be handled very carefully!

```
% Iteration 1
  Agent 1 (0/5): MAKE_LMOST_TRY
  Agent 2 (1/6): MAKE_LMOST_TRY
% Iteration 2
  Agent 1 (0/5): SEND_LMOST
  Agent 2 (1/6): SEND_LMOST
% Iteration 3
  Agent 3 (2/6): STORE_FW_MSG_FROM_WEST
  Agent 4 (1/5): STORE_FW_MSG_FROM_NORTH
  Agent 4 (1/5): STORE_FW_MSG_FROM_WEST
% Iteration 4
  Agent 3 (2/6): FW_MSG_IF_W
  Agent 4 (1/5): FW_MSG_IF_N_AND_W_C1__
% Iteration 5
  Agent 5 (2/5): STORE_FW_MSG_FROM_NORTH
  Agent 5 (2/5): STORE_FW_MSG_FROM_WEST
% Iteration 6
  Agent 5 (2/5): FW_MSG_IF_N_AND_W_C1__
% Iteration 7
  Agent 6 (3/5): STORE_FW_MSG_FROM_WEST
% Iteration 8
  Agent 6 (3/5): FW_MSG_IF_W
```

```
% Iteration 9
  Agent 7 (3/4): STORE_FW_MSG_FROM_NORTH
% Iteration 10
  Agent 7 (3/4): FW_MSG_IF_N
```

The file `files/error.log` contains a complete history of occurred errors. In the following example warning and error messages are shown. Each warning starts with `warning(` and ends with a `)`; error messages start with the string `error(` and end with a `)`.

```
% Iteration 122
warning(Perform Rule: COLLISION IN UNIVERSE)
  error(Consistency Error B:
    Universe[32,15] = 16
    Agent[13] = 32,15)
% Iteration 123
% Iteration 124
warning(Internal Error! Agent on 32,15 mark Attached to agent 10, but isn't there!)
warning(Internal Error! Agent on 32,15 mark Attached to agent 10, but isn't there!)
```

7 Requirements

The program is developed in Java 1.4.2.15. in order to run the application Java JRE 1.4 or higher must be installed.

8 Installation

To install the program, unzip the zip file in a folder of your choice. The program folder contains the program file `CrystalSimulator.jar` and a folder `files` with the files

- `config.txt`
- `agents.txt`
- `rules.txt`

The file `files/config.txt` is necessary to run the program.
The files `files/agents.txt` and `files/rules.txt` are used as the starting initial setting.
The files

- `files/actions.log`
- `files/positions.log`
- `files/error.log`.

are not initially necessary: they are created on starting the program.