

AutoCell

Generated by Doxygen 1.8.14



# Contents

<b>1</b>	<b>Presentation</b>	<b>1</b>
<b>2</b>	<b>Game of life</b>	<b>3</b>
<b>3</b>	<b>Hierarchical Index</b>	<b>5</b>
3.1	Class Hierarchy . . . . .	5
<b>4</b>	<b>Class Index</b>	<b>7</b>
4.1	Class List . . . . .	7
<b>5</b>	<b>File Index</b>	<b>9</b>
5.1	File List . . . . .	9
<b>6</b>	<b>Class Documentation</b>	<b>11</b>
6.1	Automate Class Reference . . . . .	11
6.1.1	Detailed Description . . . . .	12
6.1.2	Constructor & Destructor Documentation . . . . .	12
6.1.2.1	Automate() [1/3] . . . . .	12
6.1.2.2	Automate() [2/3] . . . . .	12
6.1.2.3	Automate() [3/3] . . . . .	13
6.1.2.4	~Automate() . . . . .	13
6.1.3	Member Function Documentation . . . . .	13
6.1.3.1	addRule() . . . . .	14
6.1.3.2	addRuleFile() . . . . .	14
6.1.3.3	getCellHandler() . . . . .	14
6.1.3.4	getRules() . . . . .	14

6.1.3.5	<a href="#">loadRules()</a> . . . . .	14
6.1.3.6	<a href="#">run()</a> . . . . .	15
6.1.3.7	<a href="#">saveAll()</a> . . . . .	15
6.1.3.8	<a href="#">saveCells()</a> . . . . .	15
6.1.3.9	<a href="#">saveRules()</a> . . . . .	16
6.1.3.10	<a href="#">setRulePriority()</a> . . . . .	16
6.1.4	<a href="#">Friends And Related Function Documentation</a> . . . . .	16
6.1.4.1	<a href="#">AutomateHandler</a> . . . . .	16
6.1.5	<a href="#">Member Data Documentation</a> . . . . .	17
6.1.5.1	<a href="#">m_cellHandler</a> . . . . .	17
6.1.5.2	<a href="#">m_rules</a> . . . . .	17
6.2	<a href="#">AutomateHandler Class Reference</a> . . . . .	17
6.2.1	<a href="#">Detailed Description</a> . . . . .	18
6.2.2	<a href="#">Constructor &amp; Destructor Documentation</a> . . . . .	18
6.2.2.1	<a href="#">AutomateHandler()</a> [1/2] . . . . .	18
6.2.2.2	<a href="#">AutomateHandler()</a> [2/2] . . . . .	19
6.2.2.3	<a href="#">~AutomateHandler()</a> . . . . .	19
6.2.3	<a href="#">Member Function Documentation</a> . . . . .	19
6.2.3.1	<a href="#">addAutomate()</a> . . . . .	19
6.2.3.2	<a href="#">deleteAutomate()</a> . . . . .	19
6.2.3.3	<a href="#">deleteAutomateHandler()</a> . . . . .	20
6.2.3.4	<a href="#">getAutomate()</a> . . . . .	20
6.2.3.5	<a href="#">getAutomateHandler()</a> . . . . .	21
6.2.3.6	<a href="#">getNumberAutomates()</a> . . . . .	21
6.2.3.7	<a href="#">operator=()</a> . . . . .	21
6.2.4	<a href="#">Member Data Documentation</a> . . . . .	21
6.2.4.1	<a href="#">m_activeAutomateHandler</a> . . . . .	22
6.2.4.2	<a href="#">m_ActiveAutomates</a> . . . . .	22
6.3	<a href="#">Cell Class Reference</a> . . . . .	22
6.3.1	<a href="#">Detailed Description</a> . . . . .	23

6.3.2	Constructor & Destructor Documentation . . . . .	23
6.3.2.1	Cell() . . . . .	23
6.3.3	Member Function Documentation . . . . .	23
6.3.3.1	addNeighbour() . . . . .	24
6.3.3.2	back() . . . . .	24
6.3.3.3	countNeighbours() [1/2] . . . . .	24
6.3.3.4	countNeighbours() [2/2] . . . . .	25
6.3.3.5	forceState() . . . . .	25
6.3.3.6	getNeighbour() . . . . .	25
6.3.3.7	getNeighbours() . . . . .	26
6.3.3.8	getRelativePosition() . . . . .	26
6.3.3.9	getState() . . . . .	26
6.3.3.10	reset() . . . . .	27
6.3.3.11	setState() . . . . .	27
6.3.3.12	validState() . . . . .	27
6.3.4	Member Data Documentation . . . . .	27
6.3.4.1	m_neighbours . . . . .	28
6.3.4.2	m_nextState . . . . .	28
6.3.4.3	m_states . . . . .	28
6.4	CellHandler Class Reference . . . . .	28
6.4.1	Detailed Description . . . . .	30
6.4.2	Member Typedef Documentation . . . . .	30
6.4.2.1	const_iterator . . . . .	30
6.4.2.2	iterator . . . . .	30
6.4.3	Member Enumeration Documentation . . . . .	30
6.4.3.1	generationTypes . . . . .	30
6.4.4	Constructor & Destructor Documentation . . . . .	31
6.4.4.1	CellHandler() [1/3] . . . . .	31
6.4.4.2	CellHandler() [2/3] . . . . .	31
6.4.4.3	CellHandler() [3/3] . . . . .	32

6.4.4.4	<code>~CellHandler()</code>	33
6.4.5	Member Function Documentation	33
6.4.5.1	<code>begin()</code> [1/2]	33
6.4.5.2	<code>begin()</code> [2/2]	33
6.4.5.3	<code>end()</code>	33
6.4.5.4	<code>foundNeighbours()</code>	34
6.4.5.5	<code>generate()</code>	34
6.4.5.6	<code>getCell()</code>	34
6.4.5.7	<code>getDimensions()</code>	35
6.4.5.8	<code>getListNeighboursPositions()</code>	35
6.4.5.9	<code>getListNeighboursPositionsRecursive()</code>	35
6.4.5.10	<code>getMaxState()</code>	36
6.4.5.11	<code>load()</code>	37
6.4.5.12	<code>nextStates()</code>	37
6.4.5.13	<code>positionIncrement()</code>	38
6.4.5.14	<code>previousStates()</code>	38
6.4.5.15	<code>print()</code>	38
6.4.5.16	<code>reset()</code>	39
6.4.5.17	<code>save()</code>	39
6.4.6	Member Data Documentation	39
6.4.6.1	<code>m_cells</code>	40
6.4.6.2	<code>m_dimensions</code>	40
6.5	CreationDialog Class Reference	40
6.5.1	Detailed Description	41
6.5.2	Constructor & Destructor Documentation	41
6.5.2.1	<code>CreationDialog()</code>	41
6.5.3	Member Function Documentation	41
6.5.3.1	<code>createGenButtons()</code>	42
6.5.3.2	<code>processSettings</code>	42
6.5.3.3	<code>settingsFilled</code>	42

6.5.4	Member Data Documentation . . . . .	42
6.5.4.1	m_densityBox . . . . .	42
6.5.4.2	m_dimensionsEdit . . . . .	43
6.5.4.3	m_doneBt . . . . .	43
6.5.4.4	m_empGen . . . . .	43
6.5.4.5	m_groupBox . . . . .	43
6.5.4.6	m_randGen . . . . .	43
6.5.4.7	m_stateMaxBox . . . . .	44
6.5.4.8	m_symGen . . . . .	44
6.6	CellHandler::iteratorT< CellHandler_T, Cell_T > Class Template Reference . . . . .	44
6.6.1	Detailed Description . . . . .	45
6.6.2	Constructor & Destructor Documentation . . . . .	45
6.6.2.1	iteratorT() . . . . .	45
6.6.3	Member Function Documentation . . . . .	45
6.6.3.1	changedDimension() . . . . .	46
6.6.3.2	operator"!=( ) . . . . .	46
6.6.3.3	operator*( ) . . . . .	46
6.6.3.4	operator++( ) . . . . .	46
6.6.3.5	operator->( ) . . . . .	47
6.6.4	Friends And Related Function Documentation . . . . .	47
6.6.4.1	CellHandler . . . . .	47
6.6.5	Member Data Documentation . . . . .	47
6.6.5.1	m_changedDimension . . . . .	47
6.6.5.2	m_finished . . . . .	47
6.6.5.3	m_handler . . . . .	48
6.6.5.4	m_position . . . . .	48
6.6.5.5	m_zero . . . . .	48
6.7	MainWindow Class Reference . . . . .	48
6.7.1	Detailed Description . . . . .	51
6.7.2	Constructor & Destructor Documentation . . . . .	51

6.7.2.1	MainWindow()	51
6.7.2.2	~MainWindow()	51
6.7.3	Member Function Documentation	51
6.7.3.1	addAutomatonRuleFile	52
6.7.3.2	addAutomatonRules	52
6.7.3.3	addEmptyRow()	52
6.7.3.4	backward	53
6.7.3.5	cellPressed	53
6.7.3.6	changeCellValue	53
6.7.3.7	closeTab	54
6.7.3.8	createBoard()	54
6.7.3.9	createButtons()	54
6.7.3.10	createTab()	54
6.7.3.11	createTabs()	55
6.7.3.12	createToolBar()	55
6.7.3.13	forward	55
6.7.3.14	getBoard()	55
6.7.3.15	getColor()	56
6.7.3.16	handlePlayPause	56
6.7.3.17	handleTabChanged	56
6.7.3.18	hookMoreColor()	57
6.7.3.19	nextState()	57
6.7.3.20	openCreationWindow	57
6.7.3.21	openFile	57
6.7.3.22	receiveCellHandler	58
6.7.3.23	reset	58
6.7.3.24	runAutomaton	58
6.7.3.25	saveToFile	59
6.7.3.26	setSize	59
6.7.3.27	updateBoard()	60



6.7.4	Member Data Documentation . . . . .	60
6.7.4.1	m_boardHSize . . . . .	60
6.7.4.2	m_boardVSize . . . . .	60
6.7.4.3	m_cellSetter . . . . .	61
6.7.4.4	m_cellSize . . . . .	61
6.7.4.5	m_currentCellX . . . . .	61
6.7.4.6	m_currentCellY . . . . .	61
6.7.4.7	m_newAutomatonBt . . . . .	61
6.7.4.8	m_nextStateBt . . . . .	62
6.7.4.9	m_openAutomatonBt . . . . .	62
6.7.4.10	m_pauseIcon . . . . .	62
6.7.4.11	m_playIcon . . . . .	62
6.7.4.12	m_playPauseBt . . . . .	62
6.7.4.13	m_previousStateBt . . . . .	63
6.7.4.14	m_resetBt . . . . .	63
6.7.4.15	m_running . . . . .	63
6.7.4.16	m_saveAutomatonBt . . . . .	63
6.7.4.17	m_tabs . . . . .	63
6.7.4.18	m_timer . . . . .	64
6.7.4.19	m_timeStep . . . . .	64
6.7.4.20	m_toolBar . . . . .	64
6.7.4.21	m_zoom . . . . .	64
6.8	MatrixRule Class Reference . . . . .	65
6.8.1	Detailed Description . . . . .	65
6.8.2	Constructor & Destructor Documentation . . . . .	65
6.8.2.1	MatrixRule() . . . . .	65
6.8.3	Member Function Documentation . . . . .	66
6.8.3.1	addNeighbourState() [1/2] . . . . .	66
6.8.3.2	addNeighbourState() [2/2] . . . . .	66
6.8.3.3	matchCell() . . . . .	66

6.8.3.4	<a href="#">toJson()</a>	67
6.8.4	<a href="#">Member Data Documentation</a>	67
6.8.4.1	<a href="#">m_matrix</a>	67
6.9	<a href="#">NeighbourRule Class Reference</a>	68
6.9.1	<a href="#">Detailed Description</a>	68
6.9.2	<a href="#">Constructor &amp; Destructor Documentation</a>	68
6.9.2.1	<a href="#">NeighbourRule()</a>	69
6.9.2.2	<a href="#">~NeighbourRule()</a>	69
6.9.3	<a href="#">Member Function Documentation</a>	69
6.9.3.1	<a href="#">inInterval()</a>	69
6.9.3.2	<a href="#">matchCell()</a>	70
6.9.3.3	<a href="#">toJson()</a>	70
6.9.4	<a href="#">Member Data Documentation</a>	70
6.9.4.1	<a href="#">m_neighbourInterval</a>	70
6.9.4.2	<a href="#">m_neighbourPossibleValues</a>	71
6.10	<a href="#">Rule Class Reference</a>	71
6.10.1	<a href="#">Detailed Description</a>	72
6.10.2	<a href="#">Constructor &amp; Destructor Documentation</a>	72
6.10.2.1	<a href="#">Rule()</a>	72
6.10.2.2	<a href="#">~Rule()</a>	72
6.10.3	<a href="#">Member Function Documentation</a>	72
6.10.3.1	<a href="#">getCellOutputState()</a>	72
6.10.3.2	<a href="#">matchCell()</a>	73
6.10.3.3	<a href="#">toJson()</a>	73
6.10.4	<a href="#">Member Data Documentation</a>	73
6.10.4.1	<a href="#">m_cellOutputState</a>	73
6.10.4.2	<a href="#">m_currentCellPossibleValues</a>	74
6.11	<a href="#">RuleEditor Class Reference</a>	74
6.11.1	<a href="#">Detailed Description</a>	75
6.11.2	<a href="#">Constructor &amp; Destructor Documentation</a>	75

6.11.2.1	RuleEditor()	75
6.11.3	Member Function Documentation	75
6.11.3.1	addRule	76
6.11.3.2	fileImported	76
6.11.3.3	importFile	76
6.11.3.4	removeRule	76
6.11.3.5	rulesFilled	77
6.11.3.6	sendRules	77
6.11.4	Member Data Documentation	77
6.11.4.1	m_addBt	77
6.11.4.2	m_automatonNumber	77
6.11.4.3	m_currentStatesEdit	78
6.11.4.4	m_dimensions	78
6.11.4.5	m_doneBt	78
6.11.4.6	m_importBt	78
6.11.4.7	m_lowerNeighbourBox	78
6.11.4.8	m_neighbourStatesEdit	79
6.11.4.9	m_outputStateBox	79
6.11.4.10	m_removeBt	79
6.11.4.11	m_rules	79
6.11.4.12	m_rulesListWidget	79
6.11.4.13	m_rulesTable	80
6.11.4.14	m_selectedRule	80
6.11.4.15	m_upperNeighbourBox	80

<b>7 File Documentation</b>	<b>81</b>
7.1 automate.cpp File Reference	81
7.1.1 Function Documentation	81
7.1.1.1 generate1DRules()	81
7.1.1.2 getRuleFromNumber()	82
7.2 automate.cpp	82
7.3 automate.h File Reference	86
7.3.1 Function Documentation	87
7.3.1.1 generate1DRules()	87
7.3.1.2 getRuleFromNumber()	87
7.4 automate.h	88
7.5 automatehandler.cpp File Reference	88
7.6 automatehandler.cpp	89
7.7 automatehandler.h File Reference	89
7.8 automatehandler.h	90
7.9 cell.cpp File Reference	90
7.10 cell.cpp	90
7.11 cell.h File Reference	91
7.12 cell.h	92
7.13 cellhandler.cpp File Reference	92
7.14 cellhandler.cpp	92
7.15 cellhandler.h File Reference	97
7.16 cellhandler.h	98
7.17 creationdialog.cpp File Reference	99
7.18 creationdialog.cpp	99
7.19 creationdialog.h File Reference	100
7.20 creationdialog.h	101
7.21 main.cpp File Reference	101
7.21.1 Function Documentation	101
7.21.1.1 main()	102

7.22	<a href="#">main.cpp</a>	102
7.23	<a href="#">mainwindow.cpp File Reference</a>	102
7.24	<a href="#">mainwindow.cpp</a>	102
7.25	<a href="#">mainwindow.h File Reference</a>	110
7.26	<a href="#">mainwindow.h</a>	110
7.27	<a href="#">matrixrule.cpp File Reference</a>	112
7.27.1	Function Documentation	112
7.27.1.1	<a href="#">fillInterval()</a>	112
7.28	<a href="#">matrixrule.cpp</a>	112
7.29	<a href="#">matrixrule.h File Reference</a>	114
7.29.1	Function Documentation	114
7.29.1.1	<a href="#">fillInterval()</a>	114
7.30	<a href="#">matrixrule.h</a>	114
7.31	<a href="#">neighbourrule.cpp File Reference</a>	115
7.32	<a href="#">neighbourrule.cpp</a>	115
7.33	<a href="#">neighbourrule.h File Reference</a>	116
7.34	<a href="#">neighbourrule.h</a>	116
7.35	<a href="#">presentation.md File Reference</a>	117
7.36	<a href="#">presentation.md</a>	117
7.37	<a href="#">rule.cpp File Reference</a>	117
7.38	<a href="#">rule.cpp</a>	117
7.39	<a href="#">rule.h File Reference</a>	118
7.40	<a href="#">rule.h</a>	118
7.41	<a href="#">ruleeditor.cpp File Reference</a>	118
7.42	<a href="#">ruleeditor.cpp</a>	119
7.43	<a href="#">ruleeditor.h File Reference</a>	120
7.44	<a href="#">ruleeditor.h</a>	121



# Chapter 1

## Presentation

### Le projet

Dans le cadre de l'UV LO21, nous avons réalisé le projet AutoCell. Ce projet vise à nous faire concevoir un programme informatique à partir d'un cahier des charges. Le projet a été réalisé en C++ avec le framework Qt. Les membres de l'équipe sont Alexandre Bertolotto, Elias Hariz et Maxime Escourrou, tous les 3 en GI02.

### Qu'est-ce qu'un automate cellulaire ?

D'après Wikipédia :

"Un automate cellulaire consiste en une grille régulière de « cellules » contenant chacune un « état » choisi parmi un ensemble fini et qui peut évoluer au cours du temps. L'état d'une cellule au temps  $t+1$  est fonction de l'état au temps  $t$  d'un nombre fini de cellules appelé son « voisinage ». À chaque nouvelle unité de temps, les mêmes règles sont appliquées simultanément à toutes les cellules de la grille, produisant une nouvelle « génération » de cellules dépendant entièrement de la génération précédente."

Le concept des automates cellulaires a été découvert dans les années 1940 par Stanislaw Ulam et John von Neumann au laboratoire de Los Alamos (USA). C'est dans les années 1970 que le concept s'est étendu, avec le jeu de la vie de la vie du mathématicien britannique John Horton Conway. Le jeu de la vie sera présenté plus en détail dans la partie suivante.





## Chapter 2

# Game of life

According to the requirements : a and b values are chosen by the user. No matter its current state, if the cell has between a and b neighbours living, it lives, else it dies/or stays dead. So the "current cell possible values" vector contains all the possible cell values (0 and 1) and the 2 pair contains (a, b) with an output state set at 1. 2 other rules, respectively with an interval of (0,a-1) and (b+1, 8) and an output state of 0 are created.

The game of life by John Horton Conway according to wikipedia:

"At each step, the cell evolution is determined by the state of its 8 neighbours as following: A dead cell which has exactly 3 living neighbours starts to live. An alive cell which has 2 or 3 living neighbours stays alive, else it dies."

1 : cell is alive 0 : cell is dead

```
// Rule 1: dead cell (state 0) starts living (state 1) if it has exactly 3 living neighbours (in state 1)

unsigned int rule1OutputState = 1; // output state is alive state

QVector<unsigned int> rule1CurrentCellValues;
rule1CurrentCellValues.insert(0); //current cell is dead

QPair<unsigned int, unsigned int> rule1IntervalNbrNeighbours;
rule1IntervalNbrNeighbours.first = 3;
rule1IntervalNbrNeighbours.second = 3;

QSet<unsigned int> rule1NeighbourPossibleValues;
rule1NeighbourPossibleValues<<1; //living neighbours

NeighbourRule rule1 = NeighbourRule(rule1OutputState, rule1CurrentCellValues,
    rule1IntervalNbrNeighbours, rule1NeighbourPossibleValues);

// Rule 2: alive cell (state 1) dies (goes to state 0) if it has 0 to 1 living neighbours (in state 1)

unsigned int rule2OutputState = 0; // output state is dead state

QVector<unsigned int> rule2CurrentCellValues;
rule2CurrentCellValues.insert(1); //current cell is alive

QPair<unsigned int, unsigned int> rule2IntervalNbrNeighbours;
rule2IntervalNbrNeighbours.first = 0;
rule2IntervalNbrNeighbours.second = 1;

QSet<unsigned int> rule2NeighbourPossibleValues;
rule2NeighbourPossibleValues<<1; //living neighbours

NeighbourRule rule2 = NeighbourRule(rule2OutputState, rule2CurrentCellValues,
    rule2IntervalNbrNeighbours, rule2NeighbourPossibleValues);

// Rule 3: alive cell (state 1) dies (goes to state 0) if it has 4 to 8 living neighbours (in state 1)

unsigned int rule3OutputState = 0; // output state is dead state
```

```
QVector<unsigned int> rule3CurrentCellValues;
rule2CurrentCellValues.insert(1); //current cell is alive

QPair<unsigned int, unsigned int> rule3IntervalNbrNeighbours;
rule3IntervalNbrNeighbours.first = 4;
rule3IntervalNbrNeighbours.second = 8;

QSet<unsigned int> rule3NeighbourPossibleValues;
rule3NeighbourPossibleValues<<1; //living neighbours

NeighbourRule rule3 = NeighbourRule(rule3OutputState, rule3CurrentCellValues,
    rule3IntervalNbrNeighbours, rule3NeighbourPossibleValues);
```

## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Automate . . . . .	11
AutomateHandler . . . . .	17
Cell . . . . .	22
CellHandler . . . . .	28
CellHandler::iteratorT< CellHandler_T, Cell_T > . . . . .	44
QDialog	
CreationDialog . . . . .	40
RuleEditor . . . . .	74
QMainWindow	
MainWindow . . . . .	48
Rule . . . . .	71
MatrixRule . . . . .	65
NeighbourRule . . . . .	68



## Chapter 4

# Class Index

### 4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">Automate</a>	Manage the application of rules on the cells . . . . .	11
<a href="#">AutomateHandler</a>	Implementation of singleton design pattern to manage the Automates . . . . .	17
<a href="#">Cell</a>	Contains the state, the next state and the neighbours . . . . .	22
<a href="#">CellHandler</a>	<a href="#">Cell</a> container and cell generator . . . . .	28
<a href="#">CreationDialog</a>	Automaton creation dialog box . . . . .	40
<a href="#">CellHandler::iteratorT&lt; CellHandler_T, Cell_T &gt;</a>	Implementation of iterator design pattern with a template to generate iterator and const_iterator at the same time . . . . .	44
<a href="#">MainWindow</a>	Simulation window . . . . .	48
<a href="#">MatrixRule</a>	Manage specific rules, about specific values of specific neighbour . . . . .	65
<a href="#">NeighbourRule</a>	Contains the rule condition and the output state if that condition is satisfied The rule modifies a cell depending on the number of its neighbours belonging to a range . . . . .	68
<a href="#">Rule</a>	. . . . .	71
<a href="#">RuleEditor</a>	Dialog for editing the rules . . . . .	74



## Chapter 5

# File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

automate.cpp	81
automate.h	86
automatehandler.cpp	88
automatehandler.h	89
cell.cpp	90
cell.h	91
cellhandler.cpp	92
cellhandler.h	97
creationdialog.cpp	99
creationdialog.h	100
main.cpp	101
mainwindow.cpp	102
mainwindow.h	110
matrixrule.cpp	112
matrixrule.h	114
neighbourrule.cpp	115
neighbourrule.h	116
rule.cpp	117
rule.h	118
ruleeditor.cpp	118
ruleeditor.h	120





## Chapter 6

# Class Documentation

### 6.1 Automate Class Reference

Manage the application of rules on the cells.

```
#include <automate.h>
```

#### Public Member Functions

- [Automate](#) (QString filename)  
*Create an automate with only a cellHandler from file.*
- [Automate](#) (const QVector< unsigned int > dimensions, [CellHandler::generationTypes](#) type=[CellHandler::empty](#), unsigned int stateMax=1, unsigned int density=20)  
*Create an automate with only a cellHandler with parameters.*
- [Automate](#) (QString cellHandlerFilename, QString ruleFilename)  
*Create an automate from files.*
- virtual [~Automate](#) ()  
*Destructor : free the [CellHandler](#) and the rules !*
- bool [saveRules](#) (QString filename) const  
*Save automate's rules in the file.*
- bool [saveCells](#) (QString filename) const  
*Save cellHandler.*
- bool [saveAll](#) (QString cellHandlerFilename, QString rulesFilename) const  
*Save both rules and cellHandler in the differents files.*
- void [addRuleFile](#) (QString filename)
- void [addRule](#) (const [Rule](#) \*newRule)  
*Add a new rule to the [Automate](#). Careful, the rule will be destroyed with the [Automate](#).*
- void [setRulePriority](#) (const [Rule](#) \*rule, unsigned int newPlace)  
*Modify the place of the rule in the priority list.*
- const QList< const [Rule](#) \* > &[getRules](#) () const  
*Return all the rules.*
- bool [run](#) (unsigned int nbSteps=1)  
*Apply the rule on the cells grid nbSteps times.*
- const [CellHandler](#) &[getCellHandler](#) () const  
*Accessor of m\_cellHandler.*

## Private Member Functions

- bool [loadRules](#) (const QJsonArray &json)  
*Load the rules of the json given.*

## Private Attributes

- [CellHandler](#) \* [m\\_cellHandler](#) = nullptr  
*CellHandler to go through.*
- QList< const [Rule](#) \* > [m\\_rules](#)  
*Rules to use on the cells.*

## Friends

- class [AutomateHandler](#)

### 6.1.1 Detailed Description

Manage the application of rules on the cells.

Definition at line 15 of file [automate.h](#).

### 6.1.2 Constructor & Destructor Documentation

#### 6.1.2.1 Automate() [1/3]

```
Automate::Automate (
    QString cellHandlerFilename )
```

Create an automate with only a cellHandler from file.

#### Parameters

<a href="#">cellHandlerFilename</a>	File to load
-------------------------------------	--------------

Definition at line 120 of file [automate.cpp](#).

References [m\\_cellHandler](#).

#### 6.1.2.2 Automate() [2/3]

```
Automate::Automate (
    const QVector< unsigned int > dimensions,
```

```
CellHandler::generationTypes type = CellHandler::empty,
unsigned int stateMax = 1,
unsigned int density = 20 )
```

Create an automate with only a cellHandler with parameters.

#### Parameters

<i>dimensions</i>	Dimensions of the <a href="#">CellHandler</a>
<i>type</i>	Generation type, empty by default
<i>stateMax</i>	Generate states between 0 and stateMax
<i>density</i>	Average (%) of non-zeros

Definition at line 133 of file [automate.cpp](#).

References [m\\_cellHandler](#).

#### 6.1.2.3 Automate() [3/3]

```
Automate::Automate (
    QString cellHandlerFilename,
    QString ruleFilename )
```

Create an automate from files.

#### Parameters

<i>cellHandlerFilename</i>	File of the cellHandler
<i>ruleFilename</i>	File of the rules

Definition at line 144 of file [automate.cpp](#).

References [loadRules\(\)](#), and [m\\_cellHandler](#).

#### 6.1.2.4 ~Automate()

```
Automate::~Automate ( ) [virtual]
```

Destructor : free the [CellHandler](#) and the rules !

Definition at line 179 of file [automate.cpp](#).

References [m\\_cellHandler](#), and [m\\_rules](#).

### 6.1.3 Member Function Documentation

#### 6.1.3.1 addRule()

```
void Automate::addRule (
    const Rule * newRule )
```

Add a new rule to the [Automate](#). Careful, the rule will be destroyed with the [Automate](#).

Definition at line 230 of file [automate.cpp](#).

References [m\\_rules](#).

Referenced by [MainWindow::addAutomatonRules\(\)](#).

#### 6.1.3.2 addRuleFile()

```
void Automate::addRuleFile (
    QString filename )
```

Definition at line 287 of file [automate.cpp](#).

References [loadRules\(\)](#).

Referenced by [MainWindow::addAutomatonRuleFile\(\)](#).

#### 6.1.3.3 getCellHandler()

```
const CellHandler & Automate::getCellHandler ( ) const
```

Accessor of [m\\_cellHandler](#).

Definition at line 282 of file [automate.cpp](#).

References [m\\_cellHandler](#).

Referenced by [MainWindow::createTab\(\)](#), and [MainWindow::updateBoard\(\)](#).

#### 6.1.3.4 getRules()

```
const QList< const Rule * > & Automate::getRules ( ) const
```

Return all the rules.

Definition at line 248 of file [automate.cpp](#).

References [m\\_rules](#).

#### 6.1.3.5 loadRules()

```
bool Automate::loadRules (
    const QJsonArray & json ) [private]
```

Load the rules of the json given.

##### Returns

Return false if something went wrong

## Parameters

<i>json</i>	JsonObject wich contains the rules
-------------	------------------------------------

Definition at line 7 of file [automate.cpp](#).

References [MatrixRule::addNeighbourState\(\)](#), [CellHandler::getDimensions\(\)](#), [m\\_cellHandler](#), and [m\\_rules](#).

Referenced by [addRuleFile\(\)](#), and [Automate\(\)](#).

## 6.1.3.6 run()

```
bool Automate::run (
    unsigned int nbSteps = 1 )
```

Apply the rule on the cells grid nbSteps times.

## Parameters

<i>nbSteps</i>	number of iterations of the automate on the cell grid
----------------	---

Definition at line 257 of file [automate.cpp](#).

References [CellHandler::begin\(\)](#), [CellHandler::end\(\)](#), [m\\_cellHandler](#), [m\\_rules](#), and [CellHandler::nextStates\(\)](#).

## 6.1.3.7 saveAll()

```
bool Automate::saveAll (
    QString cellHandlerFilename,
    QString rulesFilename ) const
```

Save both rules and cellHandler in the differents files.

Definition at line 223 of file [automate.cpp](#).

References [saveCells\(\)](#), and [saveRules\(\)](#).

Referenced by [MainWindow::~~MainWindow\(\)](#).

## 6.1.3.8 saveCells()

```
bool Automate::saveCells (
    QString filename ) const
```

Save cellHandler.

Definition at line 214 of file [automate.cpp](#).

References [m\\_cellHandler](#), and [CellHandler::save\(\)](#).

Referenced by [saveAll\(\)](#).

#### 6.1.3.9 saveRules()

```
bool Automate::saveRules (
    QString filename ) const
```

Save automate's rules in the file.

##### Returns

False if something went wrong

Definition at line 192 of file [automate.cpp](#).

References [m\\_rules](#).

Referenced by [saveAll\(\)](#).

#### 6.1.3.10 setRulePriority()

```
void Automate::setRulePriority (
    const Rule * rule,
    unsigned int newPlace )
```

Modify the place of the rule in the priority list.

2 rules can't have the same priority rank

##### Parameters

<i>rule</i>	<a href="#">Rule</a> to move
<i>newPlace</i>	New place of the rule

Definition at line 241 of file [automate.cpp](#).

References [m\\_rules](#).

### 6.1.4 Friends And Related Function Documentation

#### 6.1.4.1 AutomateHandler

```
friend class AutomateHandler [friend]
```

Definition at line 20 of file [automate.h](#).

### 6.1.5 Member Data Documentation

#### 6.1.5.1 m\_cellHandler

```
CellHandler* Automate::m_cellHandler = nullptr [private]
```

[CellHandler](#) to go through.

Definition at line 18 of file [automate.h](#).

Referenced by [Automate\(\)](#), [getCellHandler\(\)](#), [loadRules\(\)](#), [run\(\)](#), [saveCells\(\)](#), and [~Automate\(\)](#).

#### 6.1.5.2 m\_rules

```
QList<const Rule*> Automate::m_rules [private]
```

Rules to use on the cells.

Definition at line 19 of file [automate.h](#).

Referenced by [addRule\(\)](#), [getRules\(\)](#), [loadRules\(\)](#), [run\(\)](#), [saveRules\(\)](#), [setRulePriority\(\)](#), and [~Automate\(\)](#).

The documentation for this class was generated from the following files:

- [automate.h](#)
- [automate.cpp](#)

## 6.2 AutomateHandler Class Reference

Implementation of singleton design pattern to manage the Automates.

```
#include <automatehandler.h>
```

### Public Member Functions

- [Automate \\*](#) [getAutomate](#) (int indexAutomate)  
*Get an automate from the list according to its index.*
- unsigned int [getNumberAutomates](#) () const  
*Get the number of automates contained in the automate list.*
- void [addAutomate](#) ([Automate \\*](#)automate)  
*Add an automate in the automate list.*
- void [deleteAutomate](#) ([Automate \\*](#)automate)  
*Delete an automate from the automate list.*

## Static Public Member Functions

- static [AutomateHandler](#) & [getAutomateHandler](#) ()  
*Get the unique running automate handler instance or create one if there is no instance running.*
- static void [deleteAutomateHandler](#) ()  
*Delete the unique automate handler if it exists.*

## Private Member Functions

- [AutomateHandler](#) ()  
*Construct an automate handler.*
- [AutomateHandler](#) (const [AutomateHandler](#) &a)=delete
- [AutomateHandler](#) & [operator=](#) (const [AutomateHandler](#) &a)=delete
- [~AutomateHandler](#) ()  
*Delete all the automates contained in the automate handler.*

## Private Attributes

- [QList](#)< [Automate](#) \* > [m\\_ActiveAutomates](#)  
*list of existing automates*

## Static Private Attributes

- static [AutomateHandler](#) \* [m\\_activeAutomateHandler](#) = nullptr  
*active automate handler if existing, nullptr else*

### 6.2.1 Detailed Description

Implementation of singleton design pattern to manage the Automates.

Definition at line 10 of file [automatehandler.h](#).

### 6.2.2 Constructor & Destructor Documentation

#### 6.2.2.1 [AutomateHandler](#)() [1/2]

```
AutomateHandler::AutomateHandler ( ) [private]
```

Construct an automate handler.

Definition at line 10 of file [automatehandler.cpp](#).

Referenced by [getAutomateHandler](#)().



### 6.2.2.2 AutomateHandler() [2/2]

```
AutomateHandler::AutomateHandler (
    const AutomateHandler & a ) [private], [delete]
```

### 6.2.2.3 ~AutomateHandler()

```
AutomateHandler::~~AutomateHandler ( ) [private]
```

Delete all the automates contained in the automate handler.

Definition at line 18 of file [automatehandler.cpp](#).

References [m\\_ActiveAutomates](#).

## 6.2.3 Member Function Documentation

### 6.2.3.1 addAutomate()

```
void AutomateHandler::addAutomate (
    Automate * automate )
```

Add an automate in the automate list.

#### Parameters

<i>automate</i>	to be added to the automate list
-----------------	----------------------------------

Definition at line 78 of file [automatehandler.cpp](#).

References [m\\_ActiveAutomates](#).

Referenced by [MainWindow::MainWindow\(\)](#), [MainWindow::openFile\(\)](#), and [MainWindow::receiveCellHandler\(\)](#).

### 6.2.3.2 deleteAutomate()

```
void AutomateHandler::deleteAutomate (
    Automate * automate )
```

Delete an automate from the automate list.

## Parameters

<i>automate</i>	automate to delete
-----------------	--------------------

Definition at line 89 of file [automatehandler.cpp](#).

References [m\\_ActiveAutomates](#).

Referenced by [MainWindow::closeTab\(\)](#).

## 6.2.3.3 deleteAutomateHandler()

```
void AutomateHandler::deleteAutomateHandler ( ) [static]
```

Delete the unique automate handler if it exists.

Definition at line 39 of file [automatehandler.cpp](#).

References [m\\_activeAutomateHandler](#).

## 6.2.3.4 getAutomate()

```
Automate * AutomateHandler::getAutomate (
    int indexAutomate )
```

Get an automate from the list according to its index.

## Parameters

<i>indexAutomate</i>	Index of a specific automate in the automate list
----------------------	---

## Returns

Pointer on the requested automated if the parameter index fits with the list size

Definition at line 55 of file [automatehandler.cpp](#).

References [m\\_ActiveAutomates](#).

Referenced by [MainWindow::addAutomatonRuleFile\(\)](#), [MainWindow::addAutomatonRules\(\)](#), [MainWindow::backward\(\)](#), [MainWindow::cellPressed\(\)](#), [MainWindow::changeCellValue\(\)](#), [MainWindow::createTab\(\)](#), [MainWindow::nextState\(\)](#), [MainWindow::reset\(\)](#), [MainWindow::runAutomaton\(\)](#), [MainWindow::saveToFile\(\)](#), [MainWindow::updateBoard\(\)](#), and [MainWindow::~~MainWindow\(\)](#).

### 6.2.3.5 getAutomateHandler()

```
AutomateHandler & AutomateHandler::getAutomateHandler ( ) [static]
```

Get the unique running automate handler instance or create one if there is no instance running.

#### Returns

the unique running automate handler instance

Definition at line 29 of file [automatehandler.cpp](#).

References [AutomateHandler\(\)](#), and [m\\_activeAutomateHandler](#).

Referenced by [MainWindow::addAutomatonRuleFile\(\)](#), [MainWindow::addAutomatonRules\(\)](#), [MainWindow::backward\(\)](#), [MainWindow::cellPressed\(\)](#), [MainWindow::changeCellValue\(\)](#), [MainWindow::closeTab\(\)](#), [MainWindow::createTab\(\)](#), [MainWindow::handlePlayPause\(\)](#), [MainWindow::MainWindow\(\)](#), [MainWindow::nextState\(\)](#), [MainWindow::openFile\(\)](#), [MainWindow::receiveCellHandler\(\)](#), [MainWindow::reset\(\)](#), [MainWindow::runAutomaton\(\)](#), [MainWindow::saveToFile\(\)](#), [MainWindow::setSize\(\)](#), [MainWindow::updateBoard\(\)](#), and [MainWindow::~~MainWindow\(\)](#).

### 6.2.3.6 getNumberAutomates()

```
unsigned int AutomateHandler::getNumberAutomates ( ) const
```

Get the number of automates contained in the automate list.

#### Returns

number of automates in the automate list

Definition at line 67 of file [automatehandler.cpp](#).

References [m\\_ActiveAutomates](#).

Referenced by [MainWindow::~~MainWindow\(\)](#).

### 6.2.3.7 operator=()

```
AutomateHandler& AutomateHandler::operator= (
    const AutomateHandler & a ) [private], [delete]
```

## 6.2.4 Member Data Documentation

#### 6.2.4.1 m\_activeAutomateHandler

`AutomateHandler * AutomateHandler::m_activeAutomateHandler = nullptr [static], [private]`

active automate handler if existing, nullptr else

Initialization of the static value.

Definition at line 14 of file [automatehandler.h](#).

Referenced by [deleteAutomateHandler\(\)](#), and [getAutomateHandler\(\)](#).

#### 6.2.4.2 m\_ActiveAutomates

`QList<Automate*> AutomateHandler::m_ActiveAutomates [private]`

list of existing automates

Definition at line 13 of file [automatehandler.h](#).

Referenced by [addAutomate\(\)](#), [deleteAutomate\(\)](#), [getAutomate\(\)](#), [getNumberAutomates\(\)](#), and [~AutomateHandler\(\)](#).

The documentation for this class was generated from the following files:

- [automatehandler.h](#)
- [automatehandler.cpp](#)

### 6.3 Cell Class Reference

Contains the state, the next state and the neighbours.

`#include <cell.h>`

#### Public Member Functions

- [Cell](#) (unsigned int state=0)  
*Constructs a cell with the state given. State 0 is dead state.*
- void [setState](#) (unsigned int state)  
*Set temporary state.*
- void [validState](#) ()  
*Validate temporary state.*
- void [forceState](#) (unsigned int state)  
*Force the state change.*
- unsigned int [getState](#) () const  
*Access current cell state.*
- bool [back](#) ()  
*Set the previous state.*
- void [reset](#) ()  
*Reset the cell to the 1st state.*
- bool [addNeighbour](#) (const [Cell](#) \*neighbour, const QVector< short > relativePosition)  
*Add a new neighbour to the [Cell](#).*
- QMap< QVector< short >, const [Cell](#) \* > [getNeighbours](#) () const  
*Access neighbours list.*
- const [Cell](#) \* [getNeighbour](#) (QVector< short > relativePosition) const  
*Get the neighbour asked. If not existent, return nullptr.*
- unsigned int [countNeighbours](#) (unsigned int filterState) const  
*Return the number of neighbour which have the given state.*
- unsigned int [countNeighbours](#) () const  
*Return the number of neighbour which are not dead (=0)*

## Static Public Member Functions

- static `QVector< short > getRelativePosition` (const `QVector< unsigned int > cellPosition`, const `QVector< unsigned int > neighbourPosition`)

*Get the relative position, as `neighbourPosition` minus `cellPosition`.*

## Private Attributes

- `QStack< unsigned int > m\_states`  
*Current state.*
- `unsigned int m\_nextState`  
*Temporary state, before validation.*
- `QMap< QVector< short >, const Cell * > m\_neighbours`  
*[Cell](#)'s neighbours. Key is the relative position of the neighbour.*

### 6.3.1 Detailed Description

Contains the state, the next state and the neighbours.

Definition at line 11 of file [cell.h](#).

### 6.3.2 Constructor & Destructor Documentation

#### 6.3.2.1 `Cell()`

```
Cell::Cell (
    unsigned int state = 0 )
```

Constructs a cell with the state given. State 0 is dead state.

#### Parameters

<code>state</code>	<a href="#">Cell</a> state, dead state by default
--------------------	---

Definition at line 7 of file [cell.cpp](#).

References [m\\_states](#).

### 6.3.3 Member Function Documentation

### 6.3.3.1 addNeighbour()

```
bool Cell::addNeighbour (
    const Cell * neighbour,
    const QVector< short > relativePosition )
```

Add a new neighbour to the [Cell](#).

#### Parameters

<i>relativePosition</i>	Relative position of the new neighbour
<i>neighbour</i>	New neighbour

#### Returns

False if the neighbour already exists

Definition at line 84 of file [cell.cpp](#).

References [m\\_neighbours](#).

### 6.3.3.2 back()

```
bool Cell::back ( )
```

Set the previous state.

#### Returns

Return false if we are already at the first state

Definition at line 59 of file [cell.cpp](#).

References [m\\_nextState](#), and [m\\_states](#).

### 6.3.3.3 countNeighbours() [1/2]

```
unsigned int Cell::countNeighbours (
    unsigned int filterState ) const
```

Return the number of neighbour which have the given state.

Definition at line 111 of file [cell.cpp](#).

References [m\\_neighbours](#).

Referenced by [NeighbourRule::matchCell\(\)](#).

#### 6.3.3.4 countNeighbours() [2/2]

```
unsigned int Cell::countNeighbours ( ) const
```

Return the number of neighbour which are not dead (=0)

Definition at line 124 of file [cell.cpp](#).

References [m\\_neighbours](#).

#### 6.3.3.5 forceState()

```
void Cell::forceState (
    unsigned int state )
```

Force the state change.

Is equivalent to setState followed by validState

##### Parameters

<i>state</i>	New state
--------------	-----------

Definition at line 41 of file [cell.cpp](#).

References [m\\_nextState](#), and [m\\_states](#).

Referenced by [MainWindow::changeCellValue\(\)](#).

#### 6.3.3.6 getNeighbour()

```
const Cell * Cell::getNeighbour (
    QVector< short > relativePosition ) const
```

Get the neighbour asked. If not existent, return nullptr.

Definition at line 104 of file [cell.cpp](#).

References [m\\_neighbours](#).

Referenced by [MatrixRule::matchCell\(\)](#).

### 6.3.3.7 getNeighbours()

```
QMap< QVector< short >, const Cell * > Cell::getNeighbours ( ) const
```

Access neighbours list.

The map key is the relative position of the neighbour (like -1,0 for the cell just above)

Definition at line 97 of file [cell.cpp](#).

References [m\\_neighbours](#).

### 6.3.3.8 getRelativePosition()

```
QVector< short > Cell::getRelativePosition (
    const QVector< unsigned int > cellPosition,
    const QVector< unsigned int > neighbourPosition ) [static]
```

Get the relative position, as neighbourPosition minus cellPosition.

#### Exceptions

<i>QString</i>	Different size of position vectors
----------------	------------------------------------

#### Parameters

<i>cellPosition</i>	<a href="#">Cell</a> Position
<i>neighbourPosition</i>	Neighbour absolute position

Definition at line 141 of file [cell.cpp](#).

Referenced by [CellHandler::foundNeighbours\(\)](#).

### 6.3.3.9 getState()

```
unsigned int Cell::getState ( ) const
```

Access current cell state.

Definition at line 50 of file [cell.cpp](#).

References [m\\_states](#).

Referenced by [MainWindow::cellPressed\(\)](#), [MatrixRule::matchCell\(\)](#), and [NeighbourRule::matchCell\(\)](#).



#### 6.3.3.10 reset()

```
void Cell::reset ( )
```

Reset the cell to the 1st state.

Definition at line 70 of file [cell.cpp](#).

References [m\\_nextState](#), and [m\\_states](#).

#### 6.3.3.11 setState()

```
void Cell::setState (
    unsigned int state )
```

Set temporary state.

To change current cell state, use [setState\(unsigned int state\)](#) then [validState\(\)](#). (

##### Parameters

<i>state</i>	New state
--------------	-----------

Definition at line 20 of file [cell.cpp](#).

References [m\\_nextState](#).

#### 6.3.3.12 validState()

```
void Cell::validState ( )
```

Validate temporary state.

To change current cell state, use [setState\(unsigned int state\)](#) then [validState\(\)](#).

Definition at line 30 of file [cell.cpp](#).

References [m\\_nextState](#), and [m\\_states](#).

### 6.3.4 Member Data Documentation

#### 6.3.4.1 m\_neighbours

```
QMap<QVector<short>, const Cell*> Cell::m_neighbours [private]
```

[Cell](#)'s neighbours. Key is the relative position of the neighbour.

Definition at line 37 of file [cell.h](#).

Referenced by [addNeighbour\(\)](#), [countNeighbours\(\)](#), [getNeighbour\(\)](#), and [getNeighbours\(\)](#).

#### 6.3.4.2 m\_nextState

```
unsigned int Cell::m_nextState [private]
```

Temporary state, before validation.

Definition at line 35 of file [cell.h](#).

Referenced by [back\(\)](#), [forceState\(\)](#), [reset\(\)](#), [setState\(\)](#), and [validState\(\)](#).

#### 6.3.4.3 m\_states

```
QStack<unsigned int> Cell::m_states [private]
```

Current state.

Definition at line 34 of file [cell.h](#).

Referenced by [back\(\)](#), [Cell\(\)](#), [forceState\(\)](#), [getState\(\)](#), [reset\(\)](#), and [validState\(\)](#).

The documentation for this class was generated from the following files:

- [cell.h](#)
- [cell.cpp](#)

## 6.4 CellHandler Class Reference

[Cell](#) container and cell generator.

```
#include <cellhandler.h>
```

### Classes

- class [iteratorT](#)

*Implementation of iterator design pattern with a template to generate iterator and const\_iterator at the same time.*

## Public Types

- enum [generationTypes](#) { [empty](#), [random](#), [symetric](#) }  
*Type of random generation.*
- typedef [iteratorT](#)< const [CellHandler](#), const [Cell](#) > [const\\_iterator](#)
- typedef [iteratorT](#)< [CellHandler](#), [Cell](#) > [iterator](#)

## Public Member Functions

- [CellHandler](#) (const QString filename)  
*Construct all the cells from the json file given.*
- [CellHandler](#) (const QJsonObject &json)  
*Construct all the cells from the json object given.*
- [CellHandler](#) (const QVector< unsigned int > dimensions, [generationTypes](#) type=[empty](#), unsigned int state↔Max=1, unsigned int density=20)  
*Construct a [CellHandler](#) of the given dimension.*
- virtual [~CellHandler](#) ()  
*Destroys all cells in the [CellHandler](#).*
- [Cell](#) \* [getCell](#) (const QVector< unsigned int > position) const  
*Access the cell to the given position.*
- QVector< unsigned int > [getDimensions](#) () const  
*Accessor of m\_dimensions.*
- void [nextStates](#) () const  
*Valid the state of all cells.*
- bool [previousStates](#) () const  
*Get all the cells to their previous states.*
- void [reset](#) () const  
*Reset all the cells to the 1st state.*
- virtual bool [save](#) (QString filename) const  
*Save the [CellHandler](#) current configuration in the file given.*
- virtual void [generate](#) ([generationTypes](#) type, unsigned int stateMax=1, unsigned short density=50)  
*Replace [Cell](#) values by random values (symetric or not)*
- virtual void [print](#) (std::ostream &stream) const  
*Print in the given stream the [CellHandler](#).*
- [const\\_iterator](#) [begin](#) () const  
*Give the iterator which corresponds to the current [CellHandler](#).*
- [iterator](#) [begin](#) ()  
*Give the iterator which corresponds to the current [CellHandler](#).*
- bool [end](#) () const  
*End condition of the iterator.*

## Static Public Member Functions

- static unsigned int [getMaxState](#) ()  
*Return the max state of the [CellHandler](#).*

## Protected Member Functions

- virtual bool [load](#) (const QJsonObject &json)  
*Load the config file in the [CellHandler](#).*
- virtual void [foundNeighbours](#) ()  
*Set the neighbours of each cells.*
- virtual int [positionIncrement](#) (QVector< unsigned int > &pos) const  
*Increment the QVector given by the value choosen.*
- virtual QVector< QVector< unsigned int > > \* [getListNeighboursPositionsRecursive](#) (const QVector< unsigned int > position, unsigned int dimension, QVector< unsigned int > lastAdd) const  
*Recursive function which browse the position possibilities tree.*
- QVector< QVector< unsigned int > > & [getListNeighboursPositions](#) (const QVector< unsigned int > position) const  
*Prepare the call of the recursive version of itself.*

## Protected Attributes

- QVector< unsigned int > [m\\_dimensions](#)  
*Vector of x dimensions.*
- QMap< QVector< unsigned int >, [Cell](#) \*> [m\\_cells](#)  
*Map of cells, with a x dimensions vector as key.*

### 6.4.1 Detailed Description

[Cell](#) container and cell generator.

Generate cells from a json file.

Definition at line 20 of file [cellhandler.h](#).

### 6.4.2 Member Typedef Documentation

#### 6.4.2.1 const\_iterator

```
typedef iteratorT<const CellHandler, const Cell> CellHandler::const\_iterator
```

Definition at line 80 of file [cellhandler.h](#).

#### 6.4.2.2 iterator

```
typedef iteratorT<CellHandler, Cell> CellHandler::iterator
```

Definition at line 81 of file [cellhandler.h](#).

### 6.4.3 Member Enumeration Documentation

#### 6.4.3.1 generationTypes

```
enum CellHandler::generationTypes
```

Type of random generation.

## Enumerator

empty	Only empty cells.
random	Random cells.
symetric	Random cells but with vertical symetry (on the 1st dimension component)

Definition at line 85 of file [cellhandler.h](#).

## 6.4.4 Constructor &amp; Destructor Documentation

## 6.4.4.1 CellHandler() [1/3]

```
CellHandler::CellHandler (
    const QString filename )
```

Construct all the cells from the json file given.

The size of "cells" array must be the product of all dimensions (60 in the following example). Typical Json file:

```
{
  "dimensions": "3x4x5",
  "cells": [0,1,4,4,2,5,3,4,2,4,
            4,2,5,0,0,0,0,0,0,0,
            2,4,1,1,1,1,1,2,1,1,
            0,0,0,0,0,0,2,2,2,2,
            3,4,5,1,2,0,9,0,0,0,
            1,2,0,0,0,0,0,1,2,3,2]
}
```

## Parameters

<i>filename</i>	Json file which contains the description of all the cells
-----------------	---

## Exceptions

<i>QString</i>	Unreadable file
<i>QString</i>	Empty file
<i>QString</i>	Not valid file

Definition at line 25 of file [cellhandler.cpp](#).

References [foundNeighbours\(\)](#), and [load\(\)](#).

## 6.4.4.2 CellHandler() [2/3]

```
CellHandler::CellHandler (
    const QJsonObject & json )
```

Construct all the cells from the json object given.

The size of "cells" array must be the product of all dimensions (60 in the following example). Typical Json object:

```
{
  "dimensions": "3x4x5",
  "cells": [0,1,4,4,2,5,3,4,2,4,
            4,2,5,0,0,0,0,0,0,0,
            2,4,1,1,1,1,1,2,1,1,
            0,0,0,0,0,0,2,2,2,2,
            3,4,5,1,2,0,9,0,0,0,
            1,2,0,0,0,0,1,2,3,2]
}
```

#### Parameters

<i>json</i>	Json object which contains the description of all the cells
-------------	---

#### Exceptions

<i>QString</i>	Not valid file
----------------	----------------

Definition at line 76 of file [cellhandler.cpp](#).

References [foundNeighbours\(\)](#), and [load\(\)](#).

#### 6.4.4.3 CellHandler() [3/3]

```
CellHandler::CellHandler (
    const QVector< unsigned int > dimensions,
    generationTypes type = empty,
    unsigned int stateMax = 1,
    unsigned int density = 20 )
```

Construct a [CellHandler](#) of the given dimension.

If generationTypes is given, the [CellHandler](#) won't be empty.

#### Parameters

<i>dimensions</i>	Dimensions of the <a href="#">CellHandler</a>
<i>type</i>	Generation type, empty by default
<i>stateMax</i>	Generate states between 0 and stateMax
<i>density</i>	Average (%) of non-zeros

Definition at line 98 of file [cellhandler.cpp](#).

References [empty](#), [foundNeighbours\(\)](#), [generate\(\)](#), [m\\_cells](#), [m\\_dimensions](#), and [positionIncrement\(\)](#).

#### 6.4.4.4 ~CellHandler()

```
CellHandler::~~CellHandler ( ) [virtual]
```

Destroys all cells in the [CellHandler](#).

Definition at line 130 of file [cellhandler.cpp](#).

References [m\\_cells](#).

### 6.4.5 Member Function Documentation

#### 6.4.5.1 begin() [1/2]

```
CellHandler::const_iterator CellHandler::begin ( ) const
```

Give the iterator which corresponds to the current [CellHandler](#).

Definition at line 326 of file [cellhandler.cpp](#).

Referenced by [MainWindow::changeCellValue\(\)](#), [print\(\)](#), [Automate::run\(\)](#), [save\(\)](#), and [MainWindow::updateBoard\(\)](#).

#### 6.4.5.2 begin() [2/2]

```
CellHandler::iterator CellHandler::begin ( )
```

Give the iterator which corresponds to the current [CellHandler](#).

Definition at line 319 of file [cellhandler.cpp](#).

#### 6.4.5.3 end()

```
bool CellHandler::end ( ) const
```

End condition of the iterator.

See [iterator::operator!=\(bool finished\)](#) for further information.

Definition at line 335 of file [cellhandler.cpp](#).

Referenced by [MainWindow::changeCellValue\(\)](#), [print\(\)](#), [Automate::run\(\)](#), [save\(\)](#), and [MainWindow::updateBoard\(\)](#).

#### 6.4.5.4 foundNeighbours()

```
void CellHandler::foundNeighbours ( ) [protected], [virtual]
```

Set the neighbours of each cells.

Careful, this is in  $O(n \cdot 3^d)$ , with  $n$  the number of cells and  $d$  the number of dimensions

Definition at line 433 of file [cellhandler.cpp](#).

References [getListNeighboursPositions\(\)](#), [Cell::getRelativePosition\(\)](#), [m\\_cells](#), [m\\_dimensions](#), and [positionIncrement\(\)](#).

Referenced by [CellHandler\(\)](#).

#### 6.4.5.5 generate()

```
void CellHandler::generate (
    CellHandler::generationTypes type,
    unsigned int stateMax = 1,
    unsigned short density = 50 ) [virtual]
```

Replace [Cell](#) values by random values (symetric or not)

##### Parameters

<i>type</i>	Type of random generation
<i>stateMax</i>	Generate states between 0 and stateMax
<i>density</i>	Average (%) of non-zeros

Definition at line 240 of file [cellhandler.cpp](#).

References [m\\_cells](#), [m\\_dimensions](#), [positionIncrement\(\)](#), [random](#), and [symetric](#).

Referenced by [CellHandler\(\)](#).

#### 6.4.5.6 getCell()

```
Cell * CellHandler::getCell (
    const QVector< unsigned int > position ) const
```

Access the cell to the given position.

Definition at line 140 of file [cellhandler.cpp](#).

References [m\\_cells](#).

Referenced by [MainWindow::cellPressed\(\)](#), and [MainWindow::changeCellValue\(\)](#).



6.4.5.7 `getDimensions()`

```
QVector< unsigned int > CellHandler::getDimensions ( ) const
```

Accessor of `m_dimensions`.

Definition at line 154 of file `cellhandler.cpp`.

References `m_dimensions`.

Referenced by `MainWindow::cellPressed()`, `MainWindow::changeCellValue()`, `MainWindow::createTab()`, `Automate::loadRules()`, and `MainWindow::updateBoard()`.

6.4.5.8 `getListNeighboursPositions()`

```
QVector< QVector< unsigned int > > & CellHandler::getListNeighboursPositions (
    const QVector< unsigned int > position ) const [protected]
```

Prepare the call of the recursive version of itself.

## Parameters

<i>position</i>	Position of the central cell (x1,x2,x3,...,xn)
-----------------	--

## Returns

List of positions

Definition at line 488 of file `cellhandler.cpp`.

References `getListNeighboursPositionsRecursive()`.

Referenced by `foundNeighbours()`.

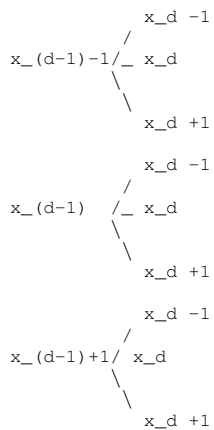
6.4.5.9 `getListNeighboursPositionsRecursive()`

```
QVector< QVector< unsigned int > > * CellHandler::getListNeighboursPositionsRecursive (
    const QVector< unsigned int > position,
    unsigned int dimension,
    QVector< unsigned int > lastAdd ) const [protected], [virtual]
```

Recursive function which browse the position possibilities tree.

Careful, the complexity is in  $O(3^{\text{dimension}})$

Piece of the tree:



The path in the tree to reach the leaf give the position

#### Parameters

<i>position</i>	Position of the cell
<i>dimension</i>	Current working dimension (number of the digit). Dimension = 2 <=> working on x2 coordinates on (x1, x2, x3, ..., xn) vector
<i>lastAdd</i>	Last position added. Like the father node of the new tree

#### Returns

List of position

Definition at line 529 of file [cellhandler.cpp](#).

References [m\\_dimensions](#).

Referenced by [getListNeighboursPositions\(\)](#).

#### 6.4.5.10 getMaxState()

```
unsigned int CellHandler::getMaxState ( ) [static]
```

Return the max state of the [CellHandler](#).

Definition at line 147 of file [cellhandler.cpp](#).

Referenced by [MainWindow::handleTabChanged\(\)](#).

## 6.4.5.11 load()

```
bool CellHandler::load (
    const QJsonObject & json ) [protected], [virtual]
```

Load the config file in the [CellHandler](#).

Exemple of a way to print cell states :

```
QVector<unsigned int> position;
for (unsigned short i = 0; i < m_dimensions.size(); i++)
{
    position.push_back(0);
}
for (unsigned int j = 0; j < m_cells.size(); j++)
{
    std::cout << m_cells.value(position)->getState() << " ";
    position.replace(0, position.at(0)+1);
    for (unsigned short i = 0; i < m_dimensions.size(); i++)
    {
        if (position.at(i) >= m_dimensions.at(i))
        {
            position.replace(i, 0);
            std::cout << std::endl;
            if (i + 1 != m_dimensions.size())
                position.replace(i+1, position.at(i+1)+1);
        }
    }
}
```

## Parameters

<i>json</i>	Json Object which contains the grid configuration
-------------	---

## Returns

False if the Json Object is not correct

Definition at line 370 of file [cellhandler.cpp](#).

References [m\\_cells](#), [m\\_dimensions](#), and [positionIncrement\(\)](#).

Referenced by [CellHandler\(\)](#).

## 6.4.5.12 nextStates()

```
void CellHandler::nextStates ( ) const
```

Valid the state of all cells.

Definition at line 161 of file [cellhandler.cpp](#).

References [m\\_cells](#).

Referenced by [Automate::run\(\)](#).

#### 6.4.5.13 positionIncrement()

```
int CellHandler::positionIncrement (
    QVector< unsigned int > & pos ) const    [protected], [virtual]
```

Increment the QVector given by the value choosen.

Careful, when the position reach the maximum, it goes to zero without leaving the function

##### Parameters

<i>pos</i>	Position to increment
------------	-----------------------

##### Returns

Number of jumped dimensions

Definition at line 463 of file [cellhandler.cpp](#).

References [m\\_dimensions](#).

Referenced by [CellHandler\(\)](#), [foundNeighbours\(\)](#), [generate\(\)](#), and [load\(\)](#).

#### 6.4.5.14 previousStates()

```
bool CellHandler::previousStates ( ) const
```

Get all the cells to their previous states.

Definition at line 171 of file [cellhandler.cpp](#).

References [m\\_cells](#).

#### 6.4.5.15 print()

```
void CellHandler::print (
    std::ostream & stream ) const    [virtual]
```

Print in the given stream the [CellHandler](#).

##### Parameters

<i>stream</i>	Stream to print into
---------------	----------------------

Definition at line 305 of file [cellhandler.cpp](#).

References [begin\(\)](#), and [end\(\)](#).

#### 6.4.5.16 reset()

```
void CellHandler::reset ( ) const
```

Reset all the cells to the 1st state.

Definition at line 183 of file [cellhandler.cpp](#).

References [m\\_cells](#).

#### 6.4.5.17 save()

```
bool CellHandler::save (
    QString filename ) const [virtual]
```

Save the [CellHandler](#) current configuration in the file given.

##### Parameters

<i>filename</i>	Path to the file
-----------------	------------------

##### Returns

False if there was a problem

##### Exceptions

<i>QString</i>	Impossible to open the file
----------------	-----------------------------

Definition at line 198 of file [cellhandler.cpp](#).

References [begin\(\)](#), [end\(\)](#), and [m\\_dimensions](#).

Referenced by [Automate::saveCells\(\)](#).

## 6.4.6 Member Data Documentation

#### 6.4.6.1 m\_cells

`QMap<QVector<unsigned int>, Cell* > CellHandler::m_cells` [protected]

Map of cells, with a x dimensions vector as key.

Definition at line 121 of file [cellhandler.h](#).

Referenced by [CellHandler\(\)](#), [foundNeighbours\(\)](#), [generate\(\)](#), [getCell\(\)](#), [load\(\)](#), [nextStates\(\)](#), [previousStates\(\)](#), [reset\(\)](#), and [~CellHandler\(\)](#).

#### 6.4.6.2 m\_dimensions

`QVector<unsigned int> CellHandler::m_dimensions` [protected]

Vector of x dimensions.

Definition at line 120 of file [cellhandler.h](#).

Referenced by [CellHandler\(\)](#), [foundNeighbours\(\)](#), [generate\(\)](#), [getDimensions\(\)](#), [getListNeighboursPositionsRecursive\(\)](#), [load\(\)](#), [positionIncrement\(\)](#), and [save\(\)](#).

The documentation for this class was generated from the following files:

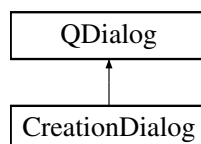
- [cellhandler.h](#)
- [cellhandler.cpp](#)

## 6.5 CreationDialog Class Reference

Automaton creation dialog box.

```
#include <creationdialog.h>
```

Inheritance diagram for CreationDialog:



### Public Slots

- void [processSettings](#) ()

### Signals

- void [settingsFilled](#) (const QVector< unsigned int > dimensions, [CellHandler::generationTypes](#) type=Cell↔Handler::generationTypes::empty, unsigned int stateMax=1, unsigned int density=20)

## Public Member Functions

- [CreationDialog](#) (QWidget \*parent=0)  
*Constructor of the cellHandler creation dialog.*

## Private Member Functions

- QGroupBox \* [createGenButtons](#) ()  
*Creates radio buttons to select cell generation type.*

## Private Attributes

- QLineEdit \* [m\\_dimensionsEdit](#)
- QSpinBox \* [m\\_densityBox](#)
- QSpinBox \* [m\\_stateMaxBox](#)
- QPushButton \* [m\\_doneBt](#)
- QGroupBox \* [m\\_groupBox](#)
- QRadioButton \* [m\\_empGen](#)
- QRadioButton \* [m\\_randGen](#)
- QRadioButton \* [m\\_symGen](#)

### 6.5.1 Detailed Description

Automaton creation dialog box.

Allow the user to input settings to create an automaton

Definition at line 13 of file [creationdialog.h](#).

### 6.5.2 Constructor & Destructor Documentation

#### 6.5.2.1 CreationDialog()

```
CreationDialog::CreationDialog (  
    QWidget * parent = 0 )
```

Constructor of the cellHandler creation dialog.

Definition at line 6 of file [creationdialog.cpp](#).

References [createGenButtons\(\)](#), [m\\_densityBox](#), [m\\_dimensionsEdit](#), [m\\_doneBt](#), [m\\_stateMaxBox](#), and [processSettings\(\)](#).

### 6.5.3 Member Function Documentation

#### 6.5.3.1 createGenButtons()

```
CreationDialog::createGenButtons ( ) [private]
```

Creates radio buttons to select cell generation type.

Validates user settings and sends them to [MainWindow](#).

Definition at line 52 of file [creationdialog.cpp](#).

References [m\\_empGen](#), [m\\_groupBox](#), [m\\_randGen](#), and [m\\_symGen](#).

Referenced by [CreationDialog\(\)](#).

#### 6.5.3.2 processSettings

```
void CreationDialog::processSettings ( ) [slot]
```

Definition at line 73 of file [creationdialog.cpp](#).

References [m\\_densityBox](#), [m\\_dimensionsEdit](#), [m\\_randGen](#), [m\\_stateMaxBox](#), [m\\_symGen](#), and [settingsFilled\(\)](#).

Referenced by [CreationDialog\(\)](#).

#### 6.5.3.3 settingsFilled

```
void CreationDialog::settingsFilled (
    const QVector< unsigned int > dimensions,
    CellHandler::generationTypes type = CellHandler::generationTypes::empty,
    unsigned int stateMax = 1,
    unsigned int density = 20 ) [signal]
```

Referenced by [processSettings\(\)](#).

### 6.5.4 Member Data Documentation

#### 6.5.4.1 m\_densityBox

```
QSpinBox* CreationDialog::m_densityBox [private]
```

Definition at line 30 of file [creationdialog.h](#).

Referenced by [CreationDialog\(\)](#), and [processSettings\(\)](#).



#### 6.5.4.2 m\_dimensionsEdit

`QLineEdit* CreationDialog::m_dimensionsEdit [private]`

Definition at line 29 of file [creationdialog.h](#).

Referenced by [CreationDialog\(\)](#), and [processSettings\(\)](#).

#### 6.5.4.3 m\_doneBt

`QPushButton* CreationDialog::m_doneBt [private]`

Definition at line 32 of file [creationdialog.h](#).

Referenced by [CreationDialog\(\)](#).

#### 6.5.4.4 m\_empGen

`QRadioButton* CreationDialog::m_empGen [private]`

Definition at line 35 of file [creationdialog.h](#).

Referenced by [createGenButtons\(\)](#).

#### 6.5.4.5 m\_groupBox

`QGroupBox* CreationDialog::m_groupBox [private]`

Definition at line 34 of file [creationdialog.h](#).

Referenced by [createGenButtons\(\)](#).

#### 6.5.4.6 m\_randGen

`QRadioButton* CreationDialog::m_randGen [private]`

Definition at line 36 of file [creationdialog.h](#).

Referenced by [createGenButtons\(\)](#), and [processSettings\(\)](#).

#### 6.5.4.7 m\_stateMaxBox

`QSpinBox* CreationDialog::m_stateMaxBox [private]`

Definition at line 31 of file [creationdialog.h](#).

Referenced by [CreationDialog\(\)](#), and [processSettings\(\)](#).

#### 6.5.4.8 m\_symGen

`QRadioButton* CreationDialog::m_symGen [private]`

Definition at line 37 of file [creationdialog.h](#).

Referenced by [createGenButtons\(\)](#), and [processSettings\(\)](#).

The documentation for this class was generated from the following files:

- [creationdialog.h](#)
- [creationdialog.cpp](#)

## 6.6 CellHandler::iteratorT< CellHandler\_T, Cell\_T > Class Template Reference

Implementation of iterator design pattern with a template to generate iterator and const\_iterator at the same time.

### Public Member Functions

- [iteratorT](#) (CellHandler\_T \*handler)  
*Construct an initial iterator to browse the [CellHandler](#).*
- [iteratorT](#) & [operator++](#) ()  
*Increment the current position and handle dimension changes.*
- Cell\_T \* [operator->](#) () const  
*Get the current cell.*
- Cell\_T \* [operator\\*](#) () const  
*Get the current cell.*
- bool [operator!=](#) (bool finished) const
- unsigned int [changedDimension](#) () const

### Private Attributes

- CellHandler\_T \* [m\\_handler](#)  
*[CellHandler](#) to go through.*
- QVector< unsigned int > [m\\_position](#)  
*Current position of the iterator.*
- bool [m\\_finished](#) = false  
*If we reach the last position.*
- QVector< unsigned int > [m\\_zero](#)  
*Nul vector of the good dimension (depend of m\_handler)*
- unsigned int [m\\_changedDimension](#)  
*Save the number of dimension change.*

## Friends

- class [CellHandler](#)

### 6.6.1 Detailed Description

```
template<typename CellHandler_T, typename Cell_T>
class CellHandler::iteratorT< CellHandler_T, Cell_T >
```

Implementation of iterator design pattern with a template to generate iterator and const\_iterator at the same time.

Example of use:

```
CellHandler handler("file.atc");
for (CellHandler::const_iterator it = handler.begin(); it != handler.end(); ++it
    )
{
    for (unsigned int i = 0; i < it.changedDimension(); i++)
        std::cout << std::endl;
    std::cout << it->getState() << " ";
}
```

This code will print each cell states and go to a new line when there is a change of dimension. So if there are 3 dimensions, there will be a empty line between 2D groups.

Definition at line 41 of file [cellhandler.h](#).

### 6.6.2 Constructor & Destructor Documentation

#### 6.6.2.1 iteratorT()

```
template<typename CellHandler_T , typename Cell_T >
CellHandler::iteratorT< CellHandler_T, Cell_T >::iteratorT (
    CellHandler_T * handler )
```

Construct an initial iterator to browse the [CellHandler](#).

#### Parameters

<i>handler</i>	<a href="#">CellHandler</a> to browse
----------------	---------------------------------------

Definition at line 569 of file [cellhandler.cpp](#).

References [CellHandler::iteratorT< CellHandler\\_T, Cell\\_T >::m\\_position](#), and [CellHandler::iteratorT< CellHandler\\_T, Cell\\_T >::m\\_z](#)

### 6.6.3 Member Function Documentation

### 6.6.3.1 `changedDimension()`

```
template<typename CellHandler_T , typename Cell_T >
unsigned int CellHandler::iteratorT< CellHandler_T, Cell_T >::changedDimension ( ) const
[inline]
```

Definition at line 66 of file [cellhandler.h](#).

References [CellHandler::iteratorT< CellHandler\\_T, Cell\\_T >::m\\_changedDimension](#).

### 6.6.3.2 `operator!==( )`

```
template<typename CellHandler_T , typename Cell_T >
bool CellHandler::iteratorT< CellHandler_T, Cell_T >::operator!= (
    bool finished ) const [inline]
```

Definition at line 65 of file [cellhandler.h](#).

References [CellHandler::iteratorT< CellHandler\\_T, Cell\\_T >::m\\_finished](#).

### 6.6.3.3 `operator*( )`

```
template<typename CellHandler_T , typename Cell_T >
Cell_T* CellHandler::iteratorT< CellHandler_T, Cell_T >::operator* ( ) const [inline]
```

Get the current cell.

Definition at line 61 of file [cellhandler.h](#).

References [CellHandler::iteratorT< CellHandler\\_T, Cell\\_T >::m\\_handler](#), and [CellHandler::iteratorT< CellHandler\\_T, Cell\\_T >::m\\_p](#).

### 6.6.3.4 `operator++( )`

```
template<typename CellHandler_T , typename Cell_T >
iteratorT& CellHandler::iteratorT< CellHandler_T, Cell_T >::operator++ ( ) [inline]
```

Increment the current position and handle dimension changes.

Definition at line 47 of file [cellhandler.h](#).

References [CellHandler::iteratorT< CellHandler\\_T, Cell\\_T >::m\\_changedDimension](#), [CellHandler::iteratorT< CellHandler\\_T, Cell\\_T >::m\\_handler](#), [CellHandler::iteratorT< CellHandler\\_T, Cell\\_T >::m\\_position](#), and [CellHandler::iteratorT< CellHandler\\_T, Cell\\_T >::m\\_zero](#).

#### 6.6.3.5 operator->()

```
template<typename CellHandler_T , typename Cell_T >
Cell_T* CellHandler::iteratorT< CellHandler_T, Cell_T >::operator-> ( ) const [inline]
```

Get the current cell.

Definition at line 57 of file [cellhandler.h](#).

References [CellHandler::iteratorT< CellHandler\\_T, Cell\\_T >::m\\_handler](#), and [CellHandler::iteratorT< CellHandler\\_T, Cell\\_T >::m\\_p](#).

### 6.6.4 Friends And Related Function Documentation

#### 6.6.4.1 CellHandler

```
template<typename CellHandler_T , typename Cell_T >
friend class CellHandler [friend]
```

Definition at line 43 of file [cellhandler.h](#).

### 6.6.5 Member Data Documentation

#### 6.6.5.1 m\_changedDimension

```
template<typename CellHandler_T , typename Cell_T >
unsigned int CellHandler::iteratorT< CellHandler_T, Cell_T >::m_changedDimension [private]
```

Save the number of dimension change.

Definition at line 77 of file [cellhandler.h](#).

Referenced by [CellHandler::iteratorT< CellHandler\\_T, Cell\\_T >::changedDimension\(\)](#), and [CellHandler::iteratorT< CellHandler\\_T, C](#).

#### 6.6.5.2 m\_finished

```
template<typename CellHandler_T , typename Cell_T >
bool CellHandler::iteratorT< CellHandler_T, Cell_T >::m_finished = false [private]
```

If we reach the last position.

Definition at line 75 of file [cellhandler.h](#).

Referenced by [CellHandler::iteratorT< CellHandler\\_T, Cell\\_T >::operator!=\(\)](#), and [CellHandler::iteratorT< CellHandler\\_T, Cell\\_T >::](#).

### 6.6.5.3 m\_handler

```
template<typename CellHandler_T , typename Cell_T >
CellHandler_T* CellHandler::iteratorT< CellHandler_T, Cell_T >::m_handler [private]
```

[CellHandler](#) to go through.

Definition at line 73 of file [cellhandler.h](#).

Referenced by [CellHandler::iteratorT< CellHandler\\_T, Cell\\_T >::operator\\*\(\)](#), [CellHandler::iteratorT< CellHandler\\_T, Cell\\_T >::operator++\(\)](#), and [CellHandler::iteratorT< CellHandler\\_T, Cell\\_T >::operator->\(\)](#).

### 6.6.5.4 m\_position

```
template<typename CellHandler_T , typename Cell_T >
QVector<unsigned int> CellHandler::iteratorT< CellHandler_T, Cell_T >::m_position [private]
```

Current position of the iterator.

Definition at line 74 of file [cellhandler.h](#).

Referenced by [CellHandler::iteratorT< CellHandler\\_T, Cell\\_T >::iteratorT\(\)](#), [CellHandler::iteratorT< CellHandler\\_T, Cell\\_T >::operator++\(\)](#), and [CellHandler::iteratorT< CellHandler\\_T, Cell\\_T >::operator->\(\)](#).

### 6.6.5.5 m\_zero

```
template<typename CellHandler_T , typename Cell_T >
QVector<unsigned int> CellHandler::iteratorT< CellHandler_T, Cell_T >::m_zero [private]
```

Nul vector of the good dimension (depend of m\_handler)

Definition at line 76 of file [cellhandler.h](#).

Referenced by [CellHandler::iteratorT< CellHandler\\_T, Cell\\_T >::iteratorT\(\)](#), and [CellHandler::iteratorT< CellHandler\\_T, Cell\\_T >::operator++\(\)](#).

The documentation for this class was generated from the following files:

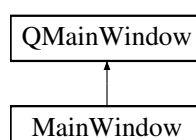
- [cellhandler.h](#)
- [cellhandler.cpp](#)

## 6.7 MainWindow Class Reference

Simulation window.

```
#include <mainwindow.h>
```

Inheritance diagram for MainWindow:



## Public Slots

- void `openFile ()`  
*Opens a file browser for the user to select automaton files and creates an automaton.*
- void `saveToFile ()`  
*Allows user to select a location and saves automaton's state and settings.*
- void `openCreationWindow ()`  
*Opens the automaton creation window.*
- void `receiveCellHandler` (const QVector< unsigned int > dimensions, `CellHandler::generationTypes` type=`CellHandler::generationTypes::empty`, unsigned int stateMax=1, unsigned int density=20)  
*Creates a new cellHandler with the provided arguments and updates the board with the created cellHandler.*
- void `addAutomatonRules` (QList< const `Rule` \*> rules)  
*Adds a list of rules to the last Automaton.*
- void `addAutomatonRuleFile` (QString path)  
*Adds a list of rules to the last Automaton from a given file.*
- void `forward ()`  
*Show the Automaton's next state.*
- void `backward ()`  
*Show the Automaton's previous state.*
- void `closeTab` (int n)  
*Closes the tab at index n. Before closing, prompts the user to save the automaton.*
- void `runAutomaton ()`  
*Runs the automaton simulation. Displays a new state on the board at regular intervals, set by the user in the interface.*
- void `handlePlayPause ()`  
*Handles the press event of the play/pause button.*
- void `reset ()`  
*Resets the current Automaton, by setting its cells to their initial state.*
- void `cellPressed` (int i, int j)  
*Handles board cell press event.*
- void `changeCellValue ()`  
*Sets the selected cell's value to the one set by the user.*
- void `handleTabChanged ()`  
*Handles tab change.*
- void `setSize` (int newCellSize)  
*Change the size of the board.*

## Public Member Functions

- `MainWindow` (QWidget \*parent=nullptr)  
*Constructor of the main window.*
- virtual `~MainWindow ()`  
*Destructor of the main window.*

## Private Member Functions

- void [createButtons](#) ()  
*Creates and connects buttons for the [MainWindow](#).*
- void [createToolBar](#) ()  
*Creates the [ToolBar](#) for the [MainWindow](#).*
- void [createBoard](#) ()
- QWidget \* [createTab](#) ()  
*Creates a new [Tab](#) with an empty board.*
- void [createTabs](#) ()  
*Creates a [QTabWidget](#) for the main window and displays it.*
- void [addEmptyRow](#) (unsigned int n)  
*Add an empty row at the end of the board.*
- void [updateBoard](#) (int index)  
*Updates cells on the board on the tab at the given index with the [cellHandler](#)'s cells states.*
- void [nextState](#) (unsigned int n)  
*Shows the *n*th next state of the automaton on the board.*
- QTableWidgetItem \* [getBoard](#) (int n)  
*Returns the board of the *n*-th tab.*

## Static Private Member Functions

- static virtual QColor [getColor](#) (int cellState)  
*Return the color wich correspond to the cellState.*
- static virtual QColor [hookMoreColor](#) (int cellState)  
*Allow the user to create more colors.*

## Private Attributes

- QTabWidget \* [m\\_tabs](#)  
*Tabs for the main window.*
- QIcon [m\\_playIcon](#)
- QIcon [m\\_pauseIcon](#)
- QPushButton \* [m\\_playPauseBt](#)
- QPushButton \* [m\\_nextStateBt](#)
- QPushButton \* [m\\_previousStateBt](#)
- QPushButton \* [m\\_openAutomatonBt](#)
- QPushButton \* [m\\_saveAutomatonBt](#)
- QPushButton \* [m\\_newAutomatonBt](#)
- QPushButton \* [m\\_resetBt](#)
- QSpinBox \* [m\\_timeStep](#)  
*Simulation time step duration input.*
- QSpinBox \* [m\\_cellSetter](#)  
*Cell state manual modification.*
- QTimer \* [m\\_timer](#)  
*Timer running between simulation steps.*
- QSlider \* [m\\_zoom](#)  
*Slider for the zoom.*
- bool [m\\_running](#)  
*If the automaton is running.*



- `QToolBar * m_toolBar`  
*Toolbar containing the buttons.*
- `int m_currentCellX`
- `int m_currentCellY`
- `unsigned int m_boardHSize = 25`
- `unsigned int m_boardVSize = 25`
- `unsigned int m_cellSize = 30`

### 6.7.1 Detailed Description

Simulation window.

Displays the automaton's current state as a board and contains user interaction components.

Definition at line 18 of file [mainwindow.h](#).

### 6.7.2 Constructor & Destructor Documentation

#### 6.7.2.1 MainWindow()

```
MainWindow::MainWindow (
    QWidget * parent = nullptr ) [explicit]
```

Constructor of the main window.

Definition at line 7 of file [mainwindow.cpp](#).

References [AutomateHandler::addAutomate\(\)](#), [createButtons\(\)](#), [createTab\(\)](#), [createTabs\(\)](#), [createToolBar\(\)](#), [AutomateHandler::getAutomateHandler\(\)](#), [m\\_running](#), [m\\_tabs](#), [m\\_timeStep](#), [m\\_zoom](#), and [updateBoard\(\)](#).

#### 6.7.2.2 ~MainWindow()

```
MainWindow::~MainWindow ( ) [virtual]
```

Destructor of the main window.

It is here that the settings are saved

Definition at line 51 of file [mainwindow.cpp](#).

References [AutomateHandler::getAutomate\(\)](#), [AutomateHandler::getAutomateHandler\(\)](#), [AutomateHandler::getNumberAutomates\(\)](#), [m\\_timeStep](#), [m\\_zoom](#), and [Automate::saveAll\(\)](#).

### 6.7.3 Member Function Documentation

### 6.7.3.1 addAutomatonRuleFile

```
void MainWindow::addAutomatonRuleFile (
    QString path ) [slot]
```

Adds a list of rules to the last Automaton from a given file.

Definition at line 531 of file [mainwindow.cpp](#).

References [Automate::addRuleFile\(\)](#), [AutomateHandler::getAutomate\(\)](#), and [AutomateHandler::getAutomateHandler\(\)](#).

Referenced by [openFile\(\)](#), and [receiveCellHandler\(\)](#).

### 6.7.3.2 addAutomatonRules

```
void MainWindow::addAutomatonRules (
    QList< const Rule *> rules ) [slot]
```

Adds a list of rules to the last Automaton.

Definition at line 520 of file [mainwindow.cpp](#).

References [Automate::addRule\(\)](#), [AutomateHandler::getAutomate\(\)](#), and [AutomateHandler::getAutomateHandler\(\)](#).

Referenced by [openFile\(\)](#), and [receiveCellHandler\(\)](#).

### 6.7.3.3 addEmptyRow()

```
void MainWindow::addEmptyRow (
    unsigned int n ) [private]
```

Add an empty row at the end of the board.

Used only in case of 1 dimension automaton

#### Parameters

<i>n</i>	Index of the board
----------	--------------------

Definition at line 491 of file [mainwindow.cpp](#).

References [getBoard\(\)](#), and [m\\_cellSize](#).

Referenced by [updateBoard\(\)](#).

#### 6.7.3.4 backward

```
void MainWindow::backward ( ) [slot]
```

Show the Automaton's previous state.

Definition at line 605 of file [mainwindow.cpp](#).

References [AutomateHandler::getAutomate\(\)](#), [AutomateHandler::getAutomateHandler\(\)](#), [m\\_tabs](#), and [updateBoard\(\)](#).

Referenced by [createButtons\(\)](#).

#### 6.7.3.5 cellPressed

```
void MainWindow::cellPressed (
    int i,
    int j ) [slot]
```

Handles board cell press event.

Definition at line 614 of file [mainwindow.cpp](#).

References [AutomateHandler::getAutomate\(\)](#), [AutomateHandler::getAutomateHandler\(\)](#), [CellHandler::getCell\(\)](#), [CellHandler::getDimensions\(\)](#), [Cell::getState\(\)](#), [m\\_cellSetter](#), [m\\_currentCellX](#), [m\\_currentCellY](#), and [m\\_tabs](#).

Referenced by [createTab\(\)](#).

#### 6.7.3.6 changeCellValue

```
void MainWindow::changeCellValue ( ) [slot]
```

Sets the selected cell's value to the one set by the user.

Definition at line 636 of file [mainwindow.cpp](#).

References [CellHandler::begin\(\)](#), [CellHandler::end\(\)](#), [Cell::forceState\(\)](#), [AutomateHandler::getAutomate\(\)](#), [AutomateHandler::getAutomateHandler\(\)](#), [getBoard\(\)](#), [CellHandler::getCell\(\)](#), [getColor\(\)](#), [CellHandler::getDimensions\(\)](#), [m\\_cellSetter](#), [m\\_currentCellX](#), [m\\_currentCellY](#), [m\\_tabs](#), and [updateBoard\(\)](#).

Referenced by [createToolBar\(\)](#).

#### 6.7.3.7 closeTab

```
void MainWindow::closeTab (
    int n ) [slot]
```

Closes the tab at index n. Before closing, prompts the user to save the automaton.

Definition at line 509 of file [mainwindow.cpp](#).

References [AutomateHandler::deleteAutomate\(\)](#), [AutomateHandler::getAutomateHandler\(\)](#), [m\\_tabs](#), and [saveToFile\(\)](#).

Referenced by [createTabs\(\)](#).

#### 6.7.3.8 createBoard()

```
void MainWindow::createBoard ( ) [private]
```

#### 6.7.3.9 createButtons()

```
void MainWindow::createButtons ( ) [private]
```

Creates and connects buttons for the [MainWindow](#).

Definition at line 71 of file [mainwindow.cpp](#).

References [backward\(\)](#), [forward\(\)](#), [handlePlayPause\(\)](#), [m\\_cellSize](#), [m\\_newAutomatonBt](#), [m\\_nextStateBt](#), [m\\_openAutomatonBt](#), [m\\_pauselcon](#), [m\\_playlcon](#), [m\\_playPauseBt](#), [m\\_previousStateBt](#), [m\\_resetBt](#), [m\\_saveAutomatonBt](#), [m\\_zoom](#), [nextState\(\)](#), [openCreationWindow\(\)](#), [openFile\(\)](#), [reset\(\)](#), [saveToFile\(\)](#), and [setSize\(\)](#).

Referenced by [MainWindow\(\)](#).

#### 6.7.3.10 createTab()

```
QWidget * MainWindow::createTab ( ) [private]
```

Creates a new Tab with an empty board.

Definition at line 212 of file [mainwindow.cpp](#).

References [cellPressed\(\)](#), [AutomateHandler::getAutomate\(\)](#), [AutomateHandler::getAutomateHandler\(\)](#), [Automate::getCellHandler\(\)](#), [CellHandler::getDimensions\(\)](#), and [m\\_cellSize](#).

Referenced by [MainWindow\(\)](#), [openFile\(\)](#), and [receiveCellHandler\(\)](#).

#### 6.7.3.11 createTabs()

```
void MainWindow::createTabs ( ) [private]
```

Creates a QTabWidget for the main window and displays it.

Definition at line 476 of file [mainwindow.cpp](#).

References [closeTab\(\)](#), [handleTabChanged\(\)](#), and [m\\_tabs](#).

Referenced by [MainWindow\(\)](#), [openFile\(\)](#), and [receiveCellHandler\(\)](#).

#### 6.7.3.12 createToolBar()

```
void MainWindow::createToolBar ( ) [private]
```

Creates the toolBar for the [MainWindow](#).

Definition at line 159 of file [mainwindow.cpp](#).

References [changeCellValue\(\)](#), [m\\_cellSetter](#), [m\\_newAutomatonBt](#), [m\\_nextStateBt](#), [m\\_openAutomatonBt](#), [m\\_playPauseBt](#), [m\\_previousStateBt](#), [m\\_resetBt](#), [m\\_saveAutomatonBt](#), [m\\_timeStep](#), [m\\_toolBar](#), and [m\\_zoom](#).

Referenced by [MainWindow\(\)](#).

#### 6.7.3.13 forward

```
void MainWindow::forward ( ) [slot]
```

Show the Automaton's next state.

Definition at line 400 of file [mainwindow.cpp](#).

References [nextState\(\)](#).

Referenced by [createButtons\(\)](#).

#### 6.7.3.14 getBoard()

```
QTableWidget * MainWindow::getBoard (
    int n ) [private]
```

Returns the board of the n-th tab.

Definition at line 407 of file [mainwindow.cpp](#).

References [m\\_tabs](#).

Referenced by [addEmptyRow\(\)](#), [changeCellValue\(\)](#), [reset\(\)](#), [setSize\(\)](#), and [updateBoard\(\)](#).

#### 6.7.3.15 getColor()

```
QColor MainWindow::getColor (
    int cellState ) [static], [private], [virtual]
```

Return the color wich correspond to the cellState.

The maximal state supported is 21. Use hookMoreColor to add more.

Definition at line 415 of file [mainwindow.cpp](#).

References [hookMoreColor\(\)](#).

Referenced by [changeCellValue\(\)](#), and [updateBoard\(\)](#).

#### 6.7.3.16 handlePlayPause

```
void MainWindow::handlePlayPause ( ) [slot]
```

Handles the press event of the play/pause button.

Definition at line 539 of file [mainwindow.cpp](#).

References [AutomateHandler::getAutomateHandler\(\)](#), [m\\_pauseIcon](#), [m\\_playIcon](#), [m\\_playPauseBt](#), [m\\_running](#), [m\\_timer](#), [m\\_timeStep](#), and [runAutomaton\(\)](#).

Referenced by [createButtons\(\)](#).

#### 6.7.3.17 handleTabChanged

```
void MainWindow::handleTabChanged ( ) [slot]
```

Handles tab change.

Definition at line 672 of file [mainwindow.cpp](#).

References [CellHandler::getMaxState\(\)](#), [m\\_cellSetter](#), [m\\_currentCellX](#), [m\\_currentCellY](#), [m\\_playIcon](#), [m\\_playPauseBt](#), [m\\_running](#), [m\\_tabs](#), and [m\\_timer](#).

Referenced by [createTabs\(\)](#).

#### 6.7.3.18 hookMoreColor()

```
static virtual QColor MainWindow::hookMoreColor (
    int cellState ) [inline], [static], [private], [virtual]
```

Allow the user to create more colors.

Definition at line 69 of file [mainwindow.h](#).

Referenced by [getColor\(\)](#).

#### 6.7.3.19 nextState()

```
void MainWindow::nextState (
    unsigned int n ) [private]
```

Shows the nth next state of the automaton on the board.

Definition at line 333 of file [mainwindow.cpp](#).

References [AutomateHandler::getAutomate\(\)](#), [AutomateHandler::getAutomateHandler\(\)](#), [m\\_tabs](#), and [updateBoard\(\)](#).

Referenced by [createButtons\(\)](#), and [forward\(\)](#).

#### 6.7.3.20 openCreationWindow

```
void MainWindow::openCreationWindow ( ) [slot]
```

Opens the automaton creation window.

Definition at line 298 of file [mainwindow.cpp](#).

References [receiveCellHandler\(\)](#).

Referenced by [createButtons\(\)](#).

#### 6.7.3.21 openFile

```
void MainWindow::openFile ( ) [slot]
```

Opens a file browser for the user to select automaton files and creates an automaton.

Definition at line 258 of file [mainwindow.cpp](#).

References [AutomateHandler::addAutomate\(\)](#), [addAutomatonRuleFile\(\)](#), [addAutomatonRules\(\)](#), [createTab\(\)](#), [createTabs\(\)](#), [AutomateHandler::getAutomateHandler\(\)](#), [m\\_tabs](#), and [updateBoard\(\)](#).

Referenced by [createButtons\(\)](#).

#### 6.7.3.22 receiveCellHandler

```
void MainWindow::receiveCellHandler (
    const QVector< unsigned int > dimensions,
    CellHandler::generationTypes type = CellHandler::generationTypes::empty,
    unsigned int stateMax = 1,
    unsigned int density = 20 ) [slot]
```

Creates a new cellHandler with the provided arguments and updates the board with the created cellHandler.

Definition at line 311 of file [mainwindow.cpp](#).

References [AutomateHandler::addAutomate\(\)](#), [addAutomatonRuleFile\(\)](#), [addAutomatonRules\(\)](#), [createTab\(\)](#), [createTabs\(\)](#), [AutomateHandler::getAutomateHandler\(\)](#), [m\\_tabs](#), and [updateBoard\(\)](#).

Referenced by [openCreationWindow\(\)](#).

#### 6.7.3.23 reset

```
void MainWindow::reset ( ) [slot]
```

Resets the current Automaton, by setting its cells to their initial state.

Definition at line 578 of file [mainwindow.cpp](#).

References [AutomateHandler::getAutomate\(\)](#), [AutomateHandler::getAutomateHandler\(\)](#), [getBoard\(\)](#), [m\\_tabs](#), and [updateBoard\(\)](#).

Referenced by [createButtons\(\)](#).

#### 6.7.3.24 runAutomaton

```
void MainWindow::runAutomaton ( ) [slot]
```

Runs the automaton simulation. Displays a new state on the board at regular intervals, set by the user in the interface.

Definition at line 566 of file [mainwindow.cpp](#).

References [AutomateHandler::getAutomate\(\)](#), [AutomateHandler::getAutomateHandler\(\)](#), [m\\_running](#), [m\\_tabs](#), and [updateBoard\(\)](#).

Referenced by [handlePlayPause\(\)](#).



#### 6.7.3.25 saveToFile

```
void MainWindow::saveToFile ( ) [slot]
```

Allows user to select a location and saves automaton's state and settings.

Definition at line 278 of file [mainwindow.cpp](#).

References [AutomateHandler::getAutomate\(\)](#), [AutomateHandler::getAutomateHandler\(\)](#), and [m\\_tabs](#).

Referenced by [closeTab\(\)](#), and [createButtons\(\)](#).

#### 6.7.3.26 setSize

```
void MainWindow::setSize (
    int newCellSize ) [slot]
```

Change the size of the board.

## Parameters

<i>newCellSize</i>	New <a href="#">Cell</a> size
--------------------	-------------------------------

Definition at line 689 of file [mainwindow.cpp](#).

References [AutomateHandler::getAutomateHandler\(\)](#), [getBoard\(\)](#), [m\\_cellSize](#), and [m\\_tabs](#).

Referenced by [createButtons\(\)](#).

### 6.7.3.27 updateBoard()

```
void MainWindow::updateBoard (
    int index ) [private]
```

Updates cells on the board on the tab at the given index with the cellHandler's cells states.

Definition at line 350 of file [mainwindow.cpp](#).

References [addEmptyRow\(\)](#), [CellHandler::begin\(\)](#), [CellHandler::end\(\)](#), [AutomateHandler::getAutomate\(\)](#), [AutomateHandler::getAutomateHandler\(\)](#), [getBoard\(\)](#), [Automate::getCellHandler\(\)](#), [getColor\(\)](#), [CellHandler::getDimensions\(\)](#), and [m\\_tabs](#).

Referenced by [backward\(\)](#), [changeCellValue\(\)](#), [MainWindow\(\)](#), [nextState\(\)](#), [openFile\(\)](#), [receiveCellHandler\(\)](#), [reset\(\)](#), and [runAutomaton\(\)](#).

## 6.7.4 Member Data Documentation

### 6.7.4.1 m\_boardHSize

```
unsigned int MainWindow::m_boardHSize = 25 [private]
```

Definition at line 51 of file [mainwindow.h](#).

### 6.7.4.2 m\_boardVSize

```
unsigned int MainWindow::m_boardVSize = 25 [private]
```

Definition at line 52 of file [mainwindow.h](#).

#### 6.7.4.3 m\_cellSetter

```
QSpinBox* MainWindow::m_cellSetter [private]
```

[Cell](#) state manual modification.

Definition at line 39 of file [mainwindow.h](#).

Referenced by [cellPressed\(\)](#), [changeCellValue\(\)](#), [createToolBar\(\)](#), and [handleTabChanged\(\)](#).

#### 6.7.4.4 m\_cellSize

```
unsigned int MainWindow::m_cellSize = 30 [private]
```

Definition at line 53 of file [mainwindow.h](#).

Referenced by [addEmptyRow\(\)](#), [createButtons\(\)](#), [createTab\(\)](#), and [setSize\(\)](#).

#### 6.7.4.5 m\_currentCellX

```
int MainWindow::m_currentCellX [private]
```

Definition at line 47 of file [mainwindow.h](#).

Referenced by [cellPressed\(\)](#), [changeCellValue\(\)](#), and [handleTabChanged\(\)](#).

#### 6.7.4.6 m\_currentCellY

```
int MainWindow::m_currentCellY [private]
```

Definition at line 48 of file [mainwindow.h](#).

Referenced by [cellPressed\(\)](#), [changeCellValue\(\)](#), and [handleTabChanged\(\)](#).

#### 6.7.4.7 m\_newAutomatonBt

```
QPushButton* MainWindow::m_newAutomatonBt [private]
```

Definition at line 34 of file [mainwindow.h](#).

Referenced by [createButtons\(\)](#), and [createToolBar\(\)](#).

#### 6.7.4.8 m\_nextStateBt

```
QToolButton* MainWindow::m_nextStateBt [private]
```

Definition at line 30 of file [mainwindow.h](#).

Referenced by [createButtons\(\)](#), and [createToolBar\(\)](#).

#### 6.7.4.9 m\_openAutomatonBt

```
QToolButton* MainWindow::m_openAutomatonBt [private]
```

Definition at line 32 of file [mainwindow.h](#).

Referenced by [createButtons\(\)](#), and [createToolBar\(\)](#).

#### 6.7.4.10 m\_pauseIcon

```
QIcon MainWindow::m_pauseIcon [private]
```

Definition at line 26 of file [mainwindow.h](#).

Referenced by [createButtons\(\)](#), and [handlePlayPause\(\)](#).

#### 6.7.4.11 m\_playIcon

```
QIcon MainWindow::m_playIcon [private]
```

Definition at line 25 of file [mainwindow.h](#).

Referenced by [createButtons\(\)](#), [handlePlayPause\(\)](#), and [handleTabChanged\(\)](#).

#### 6.7.4.12 m\_playPauseBt

```
QToolButton* MainWindow::m_playPauseBt [private]
```

Definition at line 29 of file [mainwindow.h](#).

Referenced by [createButtons\(\)](#), [createToolBar\(\)](#), [handlePlayPause\(\)](#), and [handleTabChanged\(\)](#).

#### 6.7.4.13 m\_previousStateBt

```
QToolButton* MainWindow::m_previousStateBt [private]
```

Definition at line 31 of file [mainwindow.h](#).

Referenced by [createButtons\(\)](#), and [createToolBar\(\)](#).

#### 6.7.4.14 m\_resetBt

```
QToolButton* MainWindow::m_resetBt [private]
```

Definition at line 35 of file [mainwindow.h](#).

Referenced by [createButtons\(\)](#), and [createToolBar\(\)](#).

#### 6.7.4.15 m\_running

```
bool MainWindow::m_running [private]
```

If the automaton is running.

Definition at line 44 of file [mainwindow.h](#).

Referenced by [handlePlayPause\(\)](#), [handleTabChanged\(\)](#), [MainWindow\(\)](#), and [runAutomaton\(\)](#).

#### 6.7.4.16 m\_saveAutomatonBt

```
QToolButton* MainWindow::m_saveAutomatonBt [private]
```

Definition at line 33 of file [mainwindow.h](#).

Referenced by [createButtons\(\)](#), and [createToolBar\(\)](#).

#### 6.7.4.17 m\_tabs

```
QTabWidget* MainWindow::m_tabs [private]
```

Tabs for the main window.

Definition at line 22 of file [mainwindow.h](#).

Referenced by [backward\(\)](#), [cellPressed\(\)](#), [changeCellValue\(\)](#), [closeTab\(\)](#), [createTabs\(\)](#), [getBoard\(\)](#), [handleTabChanged\(\)](#), [MainWindow\(\)](#), [nextState\(\)](#), [openFile\(\)](#), [receiveCellHandler\(\)](#), [reset\(\)](#), [runAutomaton\(\)](#), [saveToFile\(\)](#), [setSize\(\)](#), and [updateBoard\(\)](#).

#### 6.7.4.18 m\_timer

```
QTimer* MainWindow::m_timer [private]
```

Timer running between simulation steps.

Definition at line 40 of file [mainwindow.h](#).

Referenced by [handlePlayPause\(\)](#), and [handleTabChanged\(\)](#).

#### 6.7.4.19 m\_timeStep

```
QSpinBox* MainWindow::m_timeStep [private]
```

Simulation time step duration input.

Definition at line 38 of file [mainwindow.h](#).

Referenced by [createToolBar\(\)](#), [handlePlayPause\(\)](#), [MainWindow\(\)](#), and [~MainWindow\(\)](#).

#### 6.7.4.20 m\_toolBar

```
QToolBar* MainWindow::m_toolBar [private]
```

Toolbar containing the buttons.

Definition at line 45 of file [mainwindow.h](#).

Referenced by [createToolBar\(\)](#).

#### 6.7.4.21 m\_zoom

```
QSlider* MainWindow::m_zoom [private]
```

Slider for the zoom.

Definition at line 42 of file [mainwindow.h](#).

Referenced by [createButtons\(\)](#), [createToolBar\(\)](#), [MainWindow\(\)](#), and [~MainWindow\(\)](#).

The documentation for this class was generated from the following files:

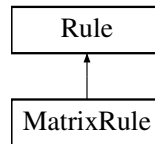
- [mainwindow.h](#)
- [mainwindow.cpp](#)

## 6.8 MatrixRule Class Reference

Manage specific rules, about specific values of specific neighbour.

```
#include <matrixrule.h>
```

Inheritance diagram for MatrixRule:



### Public Member Functions

- [MatrixRule](#) (unsigned int finalState, QVector< unsigned int > currentStates=QVector< unsigned int >())  
*Constructor.*
- virtual bool [matchCell](#) (const [Cell](#) \*cell) const  
*Tells if the cell match the rule.*
- virtual void [addNeighbourState](#) (QVector< short > relativePosition, unsigned int matchState)  
*Add a possible state to a relative position.*
- virtual void [addNeighbourState](#) (QVector< short > relativePosition, QVector< unsigned int > matchStates)  
*Add multiples possible states to existents states.*
- QJsonObject [toJson](#) () const  
*Return a QJsonObject to save the rule.*

### Protected Attributes

- QMap< QVector< short >, QVector< unsigned int > > [m\\_matrix](#)  
*Key correspond to relative position and the value to matchable states.*

### 6.8.1 Detailed Description

Manage specific rules, about specific values of specific neighbour.

Definition at line 13 of file [matrixrule.h](#).

### 6.8.2 Constructor & Destructor Documentation

#### 6.8.2.1 MatrixRule()

```
MatrixRule::MatrixRule (
    unsigned int finalState,
    QVector< unsigned int > currentStates = QVector<unsigned int>() )
```

Constructor.

## Parameters

<i>finalState</i>	Final state if the rule match the cell
<i>currentStates</i>	Possibles states of the cell. Nothing means all states

Definition at line 21 of file [matrixrule.cpp](#).

### 6.8.3 Member Function Documentation

#### 6.8.3.1 addNeighbourState() [1/2]

```
void MatrixRule::addNeighbourState (
    QVector< short > relativePosition,
    unsigned int matchState ) [virtual]
```

Add a possible state to a relative position.

Definition at line 67 of file [matrixrule.cpp](#).

References [m\\_matrix](#).

Referenced by [getRuleFromNumber\(\)](#), and [Automate::loadRules\(\)](#).

#### 6.8.3.2 addNeighbourState() [2/2]

```
void MatrixRule::addNeighbourState (
    QVector< short > relativePosition,
    QVector< unsigned int > matchStates ) [virtual]
```

Add multiples possible states to existents states.

Definition at line 74 of file [matrixrule.cpp](#).

References [m\\_matrix](#).

#### 6.8.3.3 matchCell()

```
bool MatrixRule::matchCell (
    const Cell * cell ) const [virtual]
```

Tells if the cell match the rule.



## Parameters

<i>cell</i>	<a href="#">Cell</a> to test
-------------	------------------------------

## Returns

True if the cell match the rule

Implements [Rule](#).

Definition at line 30 of file [matrixrule.cpp](#).

References [Cell::getNeighbour\(\)](#), [Cell::getState\(\)](#), [Rule::m\\_currentCellPossibleValues](#), and [m\\_matrix](#).

## 6.8.3.4 toJson()

```
QJsonObject MatrixRule::toJson ( ) const [virtual]
```

Return a QJsonObject to save the rule.

Implements [Rule](#).

Definition at line 82 of file [matrixrule.cpp](#).

References [m\\_matrix](#), and [Rule::toJson\(\)](#).

## 6.8.4 Member Data Documentation

## 6.8.4.1 m\_matrix

```
QMap<QVector<short>, QVector<unsigned int> > MatrixRule::m_matrix [protected]
```

Key correspond to relative position and the value to matchable states.

Definition at line 28 of file [matrixrule.h](#).

Referenced by [addNeighbourState\(\)](#), [matchCell\(\)](#), and [toJson\(\)](#).

The documentation for this class was generated from the following files:

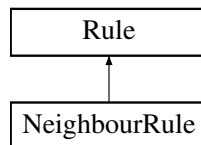
- [matrixrule.h](#)
- [matrixrule.cpp](#)

## 6.9 NeighbourRule Class Reference

Contains the rule condition and the output state if that condition is satisfied The rule modifies a cell depending on the number of its neighbours belonging to a range.

```
#include <neighbourrule.h>
```

Inheritance diagram for NeighbourRule:



### Public Member Functions

- [NeighbourRule](#) (unsigned int outputState, QVector< unsigned int > currentCellValues, QPair< unsigned int, unsigned int > intervalNbrNeighbour, QSet< unsigned int > neighbourValues=QSet< unsigned int >())  
*Constructs a neighbour rule with the parameters.*
- [~NeighbourRule](#) ()
- bool [matchCell](#) (const [Cell](#) \*c) const  
*Checks if the input cell satisfies the rule condition.*
- QJsonObject [toJson](#) () const  
*Return a QJsonObject to save the rule.*

### Protected Member Functions

- bool [inInterval](#) (unsigned int matchingNeighbours) const  
*Checks if the number of neighbours matching the state condition belongs to the condition interval.*

### Protected Attributes

- QPair< unsigned int, unsigned int > [m\\_neighbourInterval](#)  
*Stores the rule condition regarding the number of neighbours.*
- QSet< unsigned int > [m\\_neighbourPossibleValues](#)  
*Stores the possible values of the neighbours to fit with the rule.*

#### 6.9.1 Detailed Description

Contains the rule condition and the output state if that condition is satisfied The rule modifies a cell depending on the number of its neighbours belonging to a range.

Definition at line 13 of file [neighbourrule.h](#).

#### 6.9.2 Constructor & Destructor Documentation

### 6.9.2.1 NeighbourRule()

```
NeighbourRule::NeighbourRule (
    unsigned int outputState,
    QVector< unsigned int > currentCellValues,
    QPair< unsigned int, unsigned int > intervalNbrNeighbour,
    QSet< unsigned int > neighbourValues = QSet<unsigned int>() )
```

Constructs a neighbour rule with the parameters.

Definition at line 95 of file [neighbourrule.cpp](#).

References [m\\_neighbourInterval](#).

### 6.9.2.2 ~NeighbourRule()

```
NeighbourRule::~NeighbourRule ( )
```

Definition at line 104 of file [neighbourrule.cpp](#).

## 6.9.3 Member Function Documentation

### 6.9.3.1 inInterval()

```
bool NeighbourRule::inInterval (
    unsigned int matchingNeighbours ) const [protected]
```

Checks if the number of neighbours matching the state condition belongs to the condition interval.

#### Parameters

<i>matchingNeighbours</i>	Number of neighbours matching the rule condition regarding their values
---------------------------	---

#### Returns

True if the number of neighbours matches with the interval condition

Definition at line 84 of file [neighbourrule.cpp](#).

References [m\\_neighbourInterval](#).

Referenced by [matchCell\(\)](#).

### 6.9.3.2 matchCell()

```
bool NeighbourRule::matchCell (
    const Cell * c ) const [virtual]
```

Checks if the input cell satisfies the rule condition.

#### Parameters

<i>c</i>	current cell
----------	--------------

#### Returns

True if the cell matches the rule condition

Implements [Rule](#).

Definition at line 115 of file [neighbourrule.cpp](#).

References [Cell::countNeighbours\(\)](#), [Cell::getState\(\)](#), [inInterval\(\)](#), [Rule::m\\_currentCellPossibleValues](#), and [m\\_neighbourPossibleValues](#).

### 6.9.3.3 toJson()

```
QJsonObject NeighbourRule::toJson ( ) const [virtual]
```

Return a QJsonObject to save the rule.

Implements [Rule](#).

Definition at line 146 of file [neighbourrule.cpp](#).

References [m\\_neighbourInterval](#), [m\\_neighbourPossibleValues](#), and [Rule::toJson\(\)](#).

## 6.9.4 Member Data Documentation

### 6.9.4.1 m\_neighbourInterval

```
QPair<unsigned int , unsigned int> NeighbourRule::m_neighbourInterval [protected]
```

Stores the rule condition regarding the number of neighbours.

Definition at line 16 of file [neighbourrule.h](#).

Referenced by [inInterval\(\)](#), [NeighbourRule\(\)](#), and [toJson\(\)](#).

## 6.9.4.2 m\_neighbourPossibleValues

```
QSet<unsigned int> NeighbourRule::m_neighbourPossibleValues [protected]
```

Stores the possible values of the neighbours to fit with the rule.

Definition at line 18 of file [neighbourrule.h](#).

Referenced by [matchCell\(\)](#), and [toJson\(\)](#).

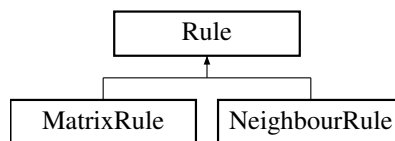
The documentation for this class was generated from the following files:

- [neighbourrule.h](#)
- [neighbourrule.cpp](#)

## 6.10 Rule Class Reference

```
#include <rule.h>
```

Inheritance diagram for Rule:



### Public Member Functions

- [Rule](#) (QVector< unsigned int > currentCellValues, unsigned int outputState)  
*Constructor of [Rule](#).*
- virtual QJsonObject [toJson](#) () const =0  
*Create a QJsonObject for the current rule.*
- virtual [~Rule](#) ()
- virtual bool [matchCell](#) (const [Cell](#) \*c) const =0  
*Verify if the cell match the rule.*
- unsigned int [getCellOutputState](#) () const  
*Get the rule output state that will be the next state if the cell matches the rule condition.*

### Protected Attributes

- QVector< unsigned int > [m\\_currentCellPossibleValues](#)  
*Stores the possible values of the current cell as part of the rule condition.*
- unsigned int [m\\_cellOutputState](#)  
*Stores the output state of the cell if the condition is matched.*

### 6.10.1 Detailed Description

Definition at line 13 of file [rule.h](#).

### 6.10.2 Constructor & Destructor Documentation

#### 6.10.2.1 Rule()

```
Rule::Rule (
    QVector< unsigned int > currentCellValues,
    unsigned int outputState )
```

Constructor of [Rule](#).

##### Parameters

<i>currentCellValues</i>	List of possibles values for the current cell
<i>outputState</i>	Next cell state

Definition at line 7 of file [rule.cpp](#).

#### 6.10.2.2 ~Rule()

```
virtual Rule::~Rule ( ) [inline], [virtual]
```

Definition at line 22 of file [rule.h](#).

### 6.10.3 Member Function Documentation

#### 6.10.3.1 getCellOutputState()

```
unsigned int Rule::getCellOutputState ( ) const
```

Get the rule output state that will be the next state if the cell matches the rule condition.

Definition at line 32 of file [rule.cpp](#).

References [m\\_cellOutputState](#).

### 6.10.3.2 matchCell()

```
virtual bool Rule::matchCell (
    const Cell * c ) const [pure virtual]
```

Verify if the cell match the rule.

Using :

```
if (rule.matchCell(&cell))
    cell.setState(rule.getCellOutputState());
```

#### Parameters

<i>c</i>	<a href="#">Cell</a> to test
----------	------------------------------

Implemented in [NeighbourRule](#), and [MatrixRule](#).

### 6.10.3.3 toJson()

```
QJsonObject Rule::toJson ( ) const [pure virtual]
```

Create a QJsonObject for the current rule.

Implemented in [NeighbourRule](#), and [MatrixRule](#).

Definition at line 15 of file [rule.cpp](#).

References [m\\_cellOutputState](#), and [m\\_currentCellPossibleValues](#).

Referenced by [MatrixRule::toJson\(\)](#), and [NeighbourRule::toJson\(\)](#).

## 6.10.4 Member Data Documentation

### 6.10.4.1 m\_cellOutputState

```
unsigned int Rule::m_cellOutputState [protected]
```

Stores the output state of the cell if the condition is matched.

Definition at line 17 of file [rule.h](#).

Referenced by [getCellOutputState\(\)](#), and [toJson\(\)](#).

#### 6.10.4.2 m\_currentCellPossibleValues

```
QVector<unsigned int> Rule::m_currentCellPossibleValues [protected]
```

Stores the possible values of the current cell as part of the rule condition.

Definition at line 16 of file [rule.h](#).

Referenced by [MatrixRule::matchCell\(\)](#), [NeighbourRule::matchCell\(\)](#), and [toJson\(\)](#).

The documentation for this class was generated from the following files:

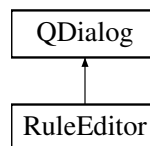
- [rule.h](#)
- [rule.cpp](#)

## 6.11 RuleEditor Class Reference

Dialog for editing the rules.

```
#include <ruleeditor.h>
```

Inheritance diagram for RuleEditor:



### Public Slots

- void [removeRule](#) ()  
*Remove the selected rule.*
- void [addRule](#) ()  
*Add the rule contained in the fields.*
- void [importFile](#) ()  
*Import a rule file.*
- void [sendRules](#) ()  
*Action when we click sur "Done".*

### Signals

- void [rulesFilled](#) (QList< const [Rule](#) \*> rules)
- void [fileImported](#) (QString path)

### Public Member Functions

- [RuleEditor](#) (unsigned int dimensions, QWidget \*parent=nullptr)  
*Constructor of the dialog for rule creation.*



## Private Attributes

- `QList< const Rule * > m_rules`
- `QListWidget * m_rulesListWidget`
- `QTableWidget * m_rulesTable`
- `QSpinBox * m_outputStateBox`
- `QLineEdit * m_currentStatesEdit`
- `QLineEdit * m_neighbourStatesEdit`
- `QSpinBox * m_upperNeighbourBox`
- `QSpinBox * m_lowerNeighbourBox`
- `QSpinBox * m_automatonNumber`
- `QPushButton * m_addBt`
- `QPushButton * m_doneBt`
- `QPushButton * m_removeBt`
- `QPushButton * m_importBt`
- `unsigned int m_selectedRule`
- `unsigned int m_dimensions`

*Dimensions of the automaton, to manage 1D dimensions.*

### 6.11.1 Detailed Description

Dialog for editing the rules.

Definition at line 9 of file [ruleeditor.h](#).

### 6.11.2 Constructor & Destructor Documentation

#### 6.11.2.1 RuleEditor()

```
RuleEditor::RuleEditor (
    unsigned int dimensions,
    QWidget * parent = nullptr ) [explicit]
```

Constructor of the dialog for rule creation.

#### Parameters

<i>dimensions</i>	Dimensions of the created automaton
-------------------	-------------------------------------

Definition at line 6 of file [ruleeditor.cpp](#).

References [addRule\(\)](#), [importFile\(\)](#), [m\\_addBt](#), [m\\_automatonNumber](#), [m\\_currentStatesEdit](#), [m\\_dimensions](#), [m\\_doneBt](#), [m\\_importBt](#), [m\\_lowerNeighbourBox](#), [m\\_neighbourStatesEdit](#), [m\\_outputStateBox](#), [m\\_removeBt](#), [m\\_rulesListWidget](#), [m\\_selectedRule](#), [m\\_upperNeighbourBox](#), [removeRule\(\)](#), and [sendRules\(\)](#).

### 6.11.3 Member Function Documentation

#### 6.11.3.1 addRule

```
void RuleEditor::addRule ( ) [slot]
```

Add the rule contained in the fields.

Definition at line 89 of file [ruleeditor.cpp](#).

References [m\\_currentStatesEdit](#), [m\\_lowerNeighbourBox](#), [m\\_neighbourStatesEdit](#), [m\\_outputStateBox](#), [m\\_rules](#), [m\\_rulesListWidget](#), and [m\\_upperNeighbourBox](#).

Referenced by [RuleEditor\(\)](#).

#### 6.11.3.2 fileImported

```
void RuleEditor::fileImported (
    QString path ) [signal]
```

Referenced by [importFile\(\)](#).

#### 6.11.3.3 importFile

```
void RuleEditor::importFile ( ) [slot]
```

Import a rule file.

Definition at line 137 of file [ruleeditor.cpp](#).

References [fileImported\(\)](#).

Referenced by [RuleEditor\(\)](#).

#### 6.11.3.4 removeRule

```
void RuleEditor::removeRule ( ) [slot]
```

Remove the selected rule.

Definition at line 114 of file [ruleeditor.cpp](#).

References [m\\_rules](#), and [m\\_rulesListWidget](#).

Referenced by [RuleEditor\(\)](#).

#### 6.11.3.5 rulesFilled

```
void RuleEditor::rulesFilled (
    QList< const Rule *> rules ) [signal]
```

Referenced by [sendRules\(\)](#).

#### 6.11.3.6 sendRules

```
void RuleEditor::sendRules ( ) [slot]
```

Action when we click sur "Done".

Definition at line 121 of file [ruleeditor.cpp](#).

References [generate1DRules\(\)](#), [m\\_automatonNumber](#), [m\\_dimensions](#), [m\\_rules](#), and [rulesFilled\(\)](#).

Referenced by [RuleEditor\(\)](#).

### 6.11.4 Member Data Documentation

#### 6.11.4.1 m\_addBt

```
QPushButton* RuleEditor::m_addBt [private]
```

Definition at line 23 of file [ruleeditor.h](#).

Referenced by [RuleEditor\(\)](#).

#### 6.11.4.2 m\_automatonNumber

```
QSpinBox* RuleEditor::m_automatonNumber [private]
```

Definition at line 21 of file [ruleeditor.h](#).

Referenced by [RuleEditor\(\)](#), and [sendRules\(\)](#).

#### 6.11.4.3 m\_currentStatesEdit

```
QLineEdit* RuleEditor::m_currentStatesEdit [private]
```

Definition at line 17 of file [ruleeditor.h](#).

Referenced by [addRule\(\)](#), and [RuleEditor\(\)](#).

#### 6.11.4.4 m\_dimensions

```
unsigned int RuleEditor::m_dimensions [private]
```

Dimensions of the automaton, to manage 1D dimensions.

Definition at line 29 of file [ruleeditor.h](#).

Referenced by [RuleEditor\(\)](#), and [sendRules\(\)](#).

#### 6.11.4.5 m\_doneBt

```
QPushButton* RuleEditor::m_doneBt [private]
```

Definition at line 24 of file [ruleeditor.h](#).

Referenced by [RuleEditor\(\)](#).

#### 6.11.4.6 m\_importBt

```
QPushButton* RuleEditor::m_importBt [private]
```

Definition at line 26 of file [ruleeditor.h](#).

Referenced by [RuleEditor\(\)](#).

#### 6.11.4.7 m\_lowerNeighbourBox

```
QSpinBox* RuleEditor::m_lowerNeighbourBox [private]
```

Definition at line 20 of file [ruleeditor.h](#).

Referenced by [addRule\(\)](#), and [RuleEditor\(\)](#).

#### 6.11.4.8 m\_neighbourStatesEdit

```
QLineEdit* RuleEditor::m_neighbourStatesEdit [private]
```

Definition at line 18 of file [ruleeditor.h](#).

Referenced by [addRule\(\)](#), and [RuleEditor\(\)](#).

#### 6.11.4.9 m\_outputStateBox

```
QSpinBox* RuleEditor::m_outputStateBox [private]
```

Definition at line 16 of file [ruleeditor.h](#).

Referenced by [addRule\(\)](#), and [RuleEditor\(\)](#).

#### 6.11.4.10 m\_removeBt

```
QPushButton* RuleEditor::m_removeBt [private]
```

Definition at line 25 of file [ruleeditor.h](#).

Referenced by [RuleEditor\(\)](#).

#### 6.11.4.11 m\_rules

```
QList<const Rule*> RuleEditor::m_rules [private]
```

Definition at line 12 of file [ruleeditor.h](#).

Referenced by [addRule\(\)](#), [removeRule\(\)](#), and [sendRules\(\)](#).

#### 6.11.4.12 m\_rulesListWidget

```
QListWidget* RuleEditor::m_rulesListWidget [private]
```

Definition at line 13 of file [ruleeditor.h](#).

Referenced by [addRule\(\)](#), [removeRule\(\)](#), and [RuleEditor\(\)](#).

#### 6.11.4.13 m\_rulesTable

```
QTableWidget* RuleEditor::m_rulesTable [private]
```

Definition at line 14 of file [ruleeditor.h](#).

#### 6.11.4.14 m\_selectedRule

```
unsigned int RuleEditor::m_selectedRule [private]
```

Definition at line 28 of file [ruleeditor.h](#).

Referenced by [RuleEditor\(\)](#).

#### 6.11.4.15 m\_upperNeighbourBox

```
QSpinBox* RuleEditor::m_upperNeighbourBox [private]
```

Definition at line 19 of file [ruleeditor.h](#).

Referenced by [addRule\(\)](#), and [RuleEditor\(\)](#).

The documentation for this class was generated from the following files:

- [ruleeditor.h](#)
- [ruleeditor.cpp](#)

# Chapter 7

## File Documentation

### 7.1 automate.cpp File Reference

```
#include "automate.h"
```

#### Functions

- `QList< const Rule * > generate1DRules` (unsigned int automatonNumber)  
*Generate the rules which corresponds to the automaton number.*
- `const MatrixRule * getRuleFromNumber` (int previousConfiguration, int nextState)  
*Create a rule from previous configuration and the next state.*

#### 7.1.1 Function Documentation

##### 7.1.1.1 `generate1DRules()`

```
QList<const Rule *> generate1DRules (  
    unsigned int automatonNumber )
```

Generate the rules which corresponds to the automaton number.

#### Parameters

<i>automatonNumber</i>	Number of the automaton (in [0, 255])
------------------------	---------------------------------------

#### Returns

List of rule pointers

Definition at line [320](#) of file [automate.cpp](#).

References [getRuleFromNumber\(\)](#).

Referenced by [RuleEditor::sendRules\(\)](#).

### 7.1.1.2 getRuleFromNumber()

```
const MatrixRule* getRuleFromNumber (
    int previousConfiguration,
    int nextState )
```

Create a rule from previous configuration and the next state.

#### Parameters

<i>previousConfiguration</i>	Previous states (left neighbour, cell and right neighbour)
<i>nextState</i>	Next state of the <a href="#">Cell</a>

#### Returns

New rule

Definition at line 348 of file [automate.cpp](#).

References [MatrixRule::addNeighbourState\(\)](#).

Referenced by [generate1DRules\(\)](#).

## 7.2 automate.cpp

```
00001 #include "automate.h"
00002
00007 bool Automate::loadRules(const QJsonArray &json)
00008 {
00009
00010     for (QJsonArray::const_iterator it = json.begin(); it != json.end(); ++it)
00011     {
00012         if (!it->isObject())
00013             return false;
00014         QJsonObject ruleJson = it->toObject();
00015
00016         if (!ruleJson.contains("type") || !ruleJson["type"].isString())
00017             return false;
00018         if (!ruleJson.contains("finalState") || !ruleJson["finalState"].isDouble())
00019             return false;
00020         if (!ruleJson.contains("currentStates") || !ruleJson["currentStates"].isArray())
00021             return false;
00022
00023         QVector<unsigned int> currentStates;
00024         QJsonArray statesJson = ruleJson["currentStates"].toArray();
00025         for (int i = 0; i < statesJson.size(); i++)
00026         {
00027             if (!statesJson.at(i).isDouble())
00028                 return false;
00029             currentStates.push_back(statesJson.at(i).toInt());
00030         }
00031
00032         if (!ruleJson["type"].toString().compare("neighbour", Qt::CaseInsensitive))
00033         {
00034             if (!ruleJson.contains("neighbourNumberMin") || !ruleJson["neighbourNumberMin"].isDouble())
00035                 return false;
```



```

00036         if (!ruleJson.contains("neighbourNumberMax") || !ruleJson["neighbourNumberMax"].isDouble())
00037             return false;
00038
00039
00040
00041         QPair<unsigned int, unsigned int> nbrNeighbourInterval(ruleJson["neighbourNumberMin"].toInt(),
ruleJson["neighbourNumberMax"].toInt());
00042         NeighbourRule *newRule;
00043         if (ruleJson.contains("neighbourStates"))
00044         {
00045             if (!ruleJson["neighbourStates"].isArray())
00046                 return false;
00047             QSet<unsigned int> neighbourStates;
00048
00049             QJsonArray statesJson = ruleJson["neighbourStates"].toArray();
00050             for (int i = 0; i < statesJson.size(); i++)
00051             {
00052                 if (!statesJson.at(i).isDouble())
00053                     return false;
00054                 neighbourStates.insert(statesJson.at(i).toInt());
00055             }
00056             newRule = new NeighbourRule((unsigned int)ruleJson["finalState"].toInt(),
currentStates, nbrNeighbourInterval, neighbourStates);
00057         }
00058         else
00059             newRule = new NeighbourRule((unsigned int)ruleJson["finalState"].toInt(),
currentStates, nbrNeighbourInterval);
00060         m_rules.push_back(newRule);
00061     }
00062     else if (!ruleJson["type"].toString().compare("matrix", Qt::CaseInsensitive))
00063     {
00064         MatrixRule *newRule = new MatrixRule((unsigned int)ruleJson["finalState"].
toInt(), currentStates);
00065         if (ruleJson.contains("neighbours"))
00066         {
00067             if (!ruleJson["neighbours"].isArray())
00068                 return false;
00069             QJsonArray neighboursJson = ruleJson["neighbours"].toArray();
00070             for (int i = 0; i < neighboursJson.size(); i++)
00071             {
00072                 if (!neighboursJson.at(i).isObject())
00073                     return false;
00074
00075                 if (!neighboursJson.at(i).toObject().contains("relativePosition") || !neighboursJson.at
(i).toObject()["relativePosition"].isArray())
00076                     return false;
00077                 if (!neighboursJson.at(i).toObject().contains("neighbourStates") || !neighboursJson.at
(i).toObject()["neighbourStates"].isArray())
00078                     return false;
00079
00080                 QVector<unsigned int> neighbourStates;
00081
00082
00083                 QJsonArray statesJson = neighboursJson.at(i).toObject()["neighbourStates"].toArray();
00084                 for (int j = 0; j < statesJson.size(); j++)
00085                 {
00086                     if (!statesJson.at(j).isDouble())
00087                         return false;
00088                     neighbourStates.push_back(statesJson.at(j).toInt());
00089                 }
00090
00091                 QVector<short> relativePosition;
00092                 QJsonArray positionJson = neighboursJson.at(i).toObject()["relativePosition"].toArray()
;
00093                 for (int j = 0; j < positionJson.size(); j++)
00094                 {
00095                     if (!positionJson.at(j).isDouble())
00096                         return false;
00097                     relativePosition.push_back(positionJson.at(j).toInt());
00098                 }
00099                 if (relativePosition.size() != m_cellHandler->
getDimensions().size())
00100                     return false;
00101                 newRule->addNeighbourState(relativePosition, neighbourStates);
00102             }
00103         }
00104     }
00105     m_rules.push_back(newRule);
00106
00107
00108 }
00109 else
00110     return false;
00111
00112 }
00113 return true;
00114 }

```

```

00115
00120 Automate::Automate(QString cellHandlerFilename)
00121 {
00122     m_cellHandler = new CellHandler(cellHandlerFilename);
00123 }
00124 }
00125
00133 Automate::Automate(const QVector<unsigned int> dimensions,
00134     CellHandler::generationTypes type, unsigned int stateMax, unsigned int density)
00135 {
00136     m_cellHandler = new CellHandler(dimensions, type, stateMax, density);
00137 }
00138
00144 Automate::Automate(QString cellHandlerFilename, QString ruleFilename)
00145 {
00146     m_cellHandler = new CellHandler(cellHandlerFilename);
00147
00148     QFile ruleFile(ruleFilename);
00149     if (!ruleFile.open(QIODevice::ReadOnly | QIODevice::Text)) {
00150         qWarning("Couldn't open given file.");
00151         throw QString(QObject::tr("Couldn't open given file"));
00152     }
00153
00154     QJsonParseError parseErr;
00155     QJsonDocument loadDoc(QJsonDocument::fromJson(ruleFile.readAll(), &parseErr));
00156
00157     ruleFile.close();
00158
00159
00160     if (loadDoc.isNull() || loadDoc.isEmpty())
00161     {
00162         qWarning() << "Could not read data : ";
00163         qWarning() << parseErr.errorString();
00164         throw QString(parseErr.errorString());
00165     }
00166
00167     if (!loadDoc.isArray())
00168     {
00169         qWarning() << "We need an array of rules !";
00170         throw QString(QObject::tr("We need an array of rules!"));
00171     }
00172
00173     loadRules(loadDoc.array());
00174 }
00175 }
00176
00179 Automate::~Automate()
00180 {
00181     delete m_cellHandler;
00182     for (QList<const Rule*>::iterator it = m_rules.begin(); it != m_rules.end(); ++it)
00183     {
00184         delete *it;
00185     }
00186 }
00187 }
00188
00192 bool Automate::saveRules(QString filename) const
00193 {
00194     QFile ruleFile(filename);
00195     if (!ruleFile.open(QIODevice::WriteOnly | QIODevice::Text)) {
00196         qWarning("Couldn't open given file.");
00197         throw QString(QObject::tr("Couldn't open given file"));
00198     }
00199
00200     QJsonArray array;
00201
00202     for (QList<const Rule*>::const_iterator it = m_rules.cbegin(); it !=
00203 m_rules.cend(); ++it)
00204         array.append((*it)->toJson());
00205
00206     QJsonDocument doc(array);
00207
00208     ruleFile.write(doc.toJson());
00209
00210     return true;
00211 }
00212
00214 bool Automate::saveCells(QString filename) const
00215 {
00216     if (m_cellHandler != nullptr)
00217         return m_cellHandler->save(filename);
00218     return false;
00219 }
00220
00223 bool Automate::saveAll(QString cellHandlerFilename, QString rulesFilename) const
00224 {

```

```

00225     return saveRules(rulesFilename) && saveCells(cellHandlerFilename);
00226 }
00227
00230 void Automate::addRule(const Rule *newRule)
00231 {
00232     m_rules.push_back(newRule);
00233 }
00234
00241 void Automate::setRulePriority(const Rule *rule, unsigned int newPlace)
00242 {
00243     m_rules.move(m_rules.indexOf(rule), newPlace);
00244 }
00245
00248 const QList<const Rule *> &Automate::getRules() const
00249 {
00250     return m_rules;
00251 }
00252
00257 bool Automate::run(unsigned int nbSteps) //void instead ?
00258 {
00259     for(unsigned int i = 0; i<nbSteps; ++i)
00260     {
00261         for (CellHandler::iterator it = m_cellHandler->
begin(); it != m_cellHandler->end(); ++it)
00262         {
00263             for (QList<const Rule*>::iterator rule = m_rules.begin(); rule !=
m_rules.end() ; ++rule)
00264             {
00265                 if((*rule)->matchCell(*it)) //if the cell matches with the rule, its state is changed
00266                 {
00267                     it->setState((*rule)->getCellOutputState());
00268                     break;
00269                 }
00270             }
00271         }
00272     }
00273     m_cellHandler->nextStates(); //apply the changes to all the cells
00274     simultaneously
00275 }
00276 return true;
00277
00278 }
00279
00282 const CellHandler &Automate::getCellHandler() const
00283 {
00284     return *m_cellHandler;
00285 }
00286
00287 void Automate::addRuleFile(QString filename){
00288     QFile ruleFile(filename);
00289     if (!ruleFile.open(QIODevice::ReadOnly | QIODevice::Text)) {
00290         qWarning("Couldn't open given file.");
00291         throw QString(QObject::tr("Couldn't open given file"));
00292     }
00293
00294     QJsonParseError parseErr;
00295     QJsonDocument loadDoc(QJsonDocument::fromJson(ruleFile.readAll(), &parseErr));
00296
00297     ruleFile.close();
00298
00299
00300     if (loadDoc.isNull() || loadDoc.isEmpty())
00301     {
00302         qWarning() << "Could not read data : ";
00303         qWarning() << parseErr.errorString();
00304         throw QString(parseErr.errorString());
00305     }
00306
00307     if (!loadDoc.isArray())
00308     {
00309         qWarning() << "We need an array of rules !";
00310         throw QString(QObject::tr("We need an array of rules!"));
00311     }
00312
00313     loadRules(loadDoc.array());
00314 }
00315
00320 QList<const Rule *> generate1DRules(unsigned int automatonNumber)
00321 {
00322     if (automatonNumber > 256) throw QString(QObject::tr("Automaton number not defined"));
00323     QList<const Rule*> ruleList;
00324     unsigned short int p = 128;
00325     int i = 7;
00326     while (i >= 0) {
00327         if (automatonNumber >= p)
00328         {

```

```

00329         ruleList.push_back((Rule*)getRuleFromNumber(i, 1));
00330         //numeroBit.push_back('1');
00331         automatonNumber -= p;
00332     }
00333     else {
00334         ruleList.push_back((Rule*)getRuleFromNumber(i, 0));
00335     }
00336     i--;
00337     p = p / 2;
00338 }
00339
00340 return ruleList;
00341 }
00342
00343 const MatrixRule* getRuleFromNumber(int previousConfiguration, int nextState)
00344 {
00345     if (previousConfiguration > 7 || previousConfiguration < 0)
00346         throw QString(QObject::tr("Configuration not possible"));
00347
00348     MatrixRule* newRule;
00349     switch(previousConfiguration)
00350     {
00351     case 0: // 000
00352         newRule = new MatrixRule(nextState, {0});
00353         newRule->addNeighbourState(QVector<short>{-1}, 0);
00354         newRule->addNeighbourState(QVector<short>{1}, 0);
00355         break;
00356     case 1: // 001
00357         newRule = new MatrixRule(nextState, {0});
00358         newRule->addNeighbourState(QVector<short>{-1}, 0);
00359         newRule->addNeighbourState(QVector<short>{1}, 1);
00360         break;
00361     case 2: // 010
00362         newRule = new MatrixRule(nextState, {1});
00363         newRule->addNeighbourState(QVector<short>{-1}, 0);
00364         newRule->addNeighbourState(QVector<short>{1}, 0);
00365         break;
00366     case 3: // 011
00367         newRule = new MatrixRule(nextState, {1});
00368         newRule->addNeighbourState(QVector<short>{-1}, 1);
00369         newRule->addNeighbourState(QVector<short>{1}, 1);
00370         break;
00371     case 4: // 100
00372         newRule = new MatrixRule(nextState, {0});
00373         newRule->addNeighbourState(QVector<short>{-1}, 1);
00374         newRule->addNeighbourState(QVector<short>{1}, 0);
00375         break;
00376     case 5: // 101
00377         newRule = new MatrixRule(nextState, {0});
00378         newRule->addNeighbourState(QVector<short>{-1}, 1);
00379         newRule->addNeighbourState(QVector<short>{1}, 1);
00380         break;
00381     case 6: // 110
00382         newRule = new MatrixRule(nextState, {1});
00383         newRule->addNeighbourState(QVector<short>{-1}, 1);
00384         newRule->addNeighbourState(QVector<short>{1}, 0);
00385         break;
00386     case 7: // 111
00387         newRule = new MatrixRule(nextState, {1});
00388         newRule->addNeighbourState(QVector<short>{-1}, 1);
00389         newRule->addNeighbourState(QVector<short>{1}, 1);
00390         break;
00391     }
00392     return newRule;
00393 }
00394
00400

```

## 7.3 automate.h File Reference

```

#include <QVector>
#include <QList>
#include "cellhandler.h"
#include "rule.h"
#include "neighbourrule.h"
#include "matrixrule.h"

```

## Classes

- class [Automate](#)  
*Manage the application of rules on the cells.*

## Functions

- QList< const [Rule](#) \* > [generate1DRules](#) (unsigned int automatonNumber)  
*Generate the rules which corresponds to the automaton number.*
- const [MatrixRule](#) \* [getRuleFromNumber](#) (int previousConfiguration, int nextState)  
*Create a rule from previous configuration and the next state.*

### 7.3.1 Function Documentation

#### 7.3.1.1 generate1DRules()

```
QList<const Rule*> generate1DRules (
    unsigned int automatonNumber )
```

Generate the rules which corresponds to the automaton number.

##### Parameters

<i>automatonNumber</i>	Number of the automaton (in [0, 255])
------------------------	---------------------------------------

##### Returns

List of rule pointers

Definition at line 320 of file [automate.cpp](#).

References [getRuleFromNumber\(\)](#).

Referenced by [RuleEditor::sendRules\(\)](#).

#### 7.3.1.2 getRuleFromNumber()

```
const MatrixRule* getRuleFromNumber (
    int previousConfiguration,
    int nextState )
```

Create a rule from previous configuration and the next state.

## Parameters

<i>previousConfiguration</i>	Previous states (left neighbour, cell and right neighbour)
<i>nextState</i>	Next state of the <a href="#">Cell</a>

## Returns

New rule

Definition at line 348 of file [automate.cpp](#).

References [MatrixRule::addNeighbourState\(\)](#).

Referenced by [generate1DRules\(\)](#).

## 7.4 automate.h

```

00001 #ifndef AUTOMATE_H
00002 #define AUTOMATE_H
00003 #include <QVector>
00004 #include <QList>
00005
00006 #include "cellhandler.h"
00007 #include "rule.h"
00008 #include "neighbourrule.h"
00009 #include "matrixrule.h"
00010
00011
00015 class Automate
00016 {
00017 private:
00018     CellHandler* m_cellHandler = nullptr;
00019     QList<const Rule*> m_rules;
00020     friend class AutomateHandler;
00021
00022     bool loadRules(const QJsonArray &json);
00023 public:
00024     Automate(QString filename);
00025     Automate(const QVector<unsigned int> dimensions,
00026             CellHandler::generationTypes type =
00027             CellHandler::empty, unsigned int stateMax = 1, unsigned int density = 20);
00026     Automate(QString cellHandlerFilename, QString ruleFilename);
00027     virtual ~Automate();
00028
00029     bool saveRules(QString filename) const ;
00030     bool saveCells(QString filename) const ;
00031     bool saveAll(QString cellHandlerFilename, QString rulesFilename) const ;
00032
00033     void addRuleFile(QString filename);
00034     void addRule(const Rule* newRule);
00035     void setRulePriority(const Rule* rule, unsigned int newPlace);
00036     const QList<const Rule *> &getRules() const;
00037
00038
00039
00040 public:
00041     bool run(unsigned int nbSteps = 1);
00042     const CellHandler& getCellHandler() const;
00043 };
00044
00045 QList<const Rule*> generate1DRules(unsigned int automatonNumber);
00046 const MatrixRule *getRuleFromNumber(int previousConfiguration, int nextState);
00047
00048 #endif // AUTOMATE_H

```

## 7.5 automatehandler.cpp File Reference

```
#include "automatehandler.h"
```

## 7.6 automatehandler.cpp

```

00001 #include "automatehandler.h"
00002
00003 AutomateHandler * AutomateHandler::m_activeAutomateHandler
00004     = nullptr;
00005
00006
00007
00010 AutomateHandler::AutomateHandler()
00011 {
00012 }
00013
00014
00015
00018 AutomateHandler::~AutomateHandler()
00019 {
00020     while(!m_ActiveAutomates.empty())
00021         delete(m_ActiveAutomates.first());
00022 }
00023
00024
00029 AutomateHandler & AutomateHandler::getAutomateHandler()
00030 {
00031     if (!m_activeAutomateHandler)
00032         m_activeAutomateHandler = new AutomateHandler;
00033     return *m_activeAutomateHandler;
00034 }
00035
00036
00039 void AutomateHandler::deleteAutomateHandler()
00040 {
00041     if(m_activeAutomateHandler)
00042     {
00043         delete m_activeAutomateHandler;
00044         m_activeAutomateHandler = nullptr;
00045     }
00046 }
00047
00048
00055 Automate * AutomateHandler::getAutomate(int indexAutomate){
00056     if(indexAutomate > m_ActiveAutomates.size())
00057         return nullptr;
00058     return m_ActiveAutomates.at(indexAutomate);
00059 }
00060
00061
00067 unsigned int AutomateHandler::getNumberAutomates() const
00068 {
00069     return m_ActiveAutomates.size();
00070 }
00071
00072
00078 void AutomateHandler::addAutomate(Automate * automate)
00079 {
00080     m_ActiveAutomates.append(automate);
00081 }
00082
00083
00089 void AutomateHandler::deleteAutomate(Automate * automate)
00090 {
00091     if(m_ActiveAutomates.contains(automate))
00092     {
00093         delete automate;
00094         m_ActiveAutomates.removeOne(automate);
00095     }
00096 }

```

## 7.7 automatehandler.h File Reference

```
#include "automate.h"
```

### Classes

- class [AutomateHandler](#)

*Implementation of singleton design pattern to manage the Automates.*

## 7.8 automatehandler.h

```

00001 #ifndef AUTOMATEHANDLER_H
00002 #define AUTOMATEHANDLER_H
00003
00004 #include "automate.h"
00005
00006
00010 class AutomateHandler
00011 {
00012 private:
00013     QList<Automate*> m_ActiveAutomates;
00014     static AutomateHandler * m_activeAutomateHandler;
00015
00016     AutomateHandler();
00017     AutomateHandler(const AutomateHandler & a) = delete;
00018     AutomateHandler & operator=(const AutomateHandler & a) = delete;
00019     ~AutomateHandler();
00020
00021 public:
00022     static AutomateHandler & getAutomateHandler();
00023     static void deleteAutomateHandler();
00024
00025     Automate * getAutomate(int indexAutomate);
00026     unsigned int getNumberAutomates() const;
00027
00028     void addAutomate(Automate * automate);
00029     void deleteAutomate(Automate * automate);
00030 };
00031
00032
00033 #endif // AUTOMATEHANDLER_H

```

## 7.9 cell.cpp File Reference

```
#include "cell.h"
```

## 7.10 cell.cpp

```

00001 #include "cell.h"
00002
00007 Cell::Cell(unsigned int state):
00008     m_nextState(state)
00009 {
00010     m_states.push(state);
00011 }
00012
00020 void Cell::setState(unsigned int state)
00021 {
00022     m_nextState = state;
00023 }
00024
00030 void Cell::validState()
00031 {
00032     m_states.push(m_nextState);
00033 }
00034
00041 void Cell::forceState(unsigned int state)
00042 {
00043     m_nextState = state;
00044     m_states.pop();
00045     m_states.push(m_nextState);
00046 }
00047
00050 unsigned int Cell::getState() const
00051 {
00052     return m_states.top();
00053 }
00054
00059 bool Cell::back()
00060 {
00061     if (m_states.size() <= 1)

```



```

00062         return false;
00063         m_states.pop();
00064         m_nextState = m_states.top();
00065         return true;
00066     }
00067
00070 void Cell::reset()
00071 {
00072     while (m_states.size() > 1)
00073         m_states.pop();
00074     m_nextState = m_states.top();
00075 }
00076
00084 bool Cell::addNeighbour(const Cell* neighbour, const QVector<short> relativePosition)
00085 {
00086     if (m_neighbours.count(relativePosition))
00087         return false;
00088
00089     m_neighbours.insert(relativePosition, neighbour);
00090     return true;
00091 }
00092
00097 QMap<QVector<short>, const Cell *> Cell::getNeighbours() const
00098 {
00099     return m_neighbours;
00100 }
00101
00104 const Cell *Cell::getNeighbour(QVector<short> relativePosition) const
00105 {
00106     return m_neighbours.value(relativePosition, nullptr);
00107 }
00108
00111 unsigned int Cell::countNeighbours(unsigned int filterState) const
00112 {
00113     unsigned int count = 0;
00114     for (QMap<QVector<short>, const Cell *>::const_iterator it = m_neighbours.begin(); it !=
00115          m_neighbours.end(); ++it)
00116     {
00117         if ((*it)->getState() == filterState)
00118             count++;
00119     }
00120     return count;
00121 }
00124 unsigned int Cell::countNeighbours() const
00125 {
00126     unsigned int count = 0;
00127     for (QMap<QVector<short>, const Cell *>::const_iterator it = m_neighbours.begin(); it !=
00128          m_neighbours.end(); ++it)
00129     {
00130         if ((*it)->getState() != 0)
00131             count++;
00132     }
00133     return count;
00134 }
00141 QVector<short> Cell::getRelativePosition(const QVector<unsigned int> cellPosition,
00142                                         const QVector<unsigned int> neighbourPosition)
00143 {
00144     if (cellPosition.size() != neighbourPosition.size())
00145     {
00146         throw QString(QObject::tr("Different size of position vectors"));
00147     }
00148     QVector<short> relativePosition;
00149     for (short i = 0; i < cellPosition.size(); i++)
00150         relativePosition.push_back(neighbourPosition.at(i) - cellPosition.at(i));
00151     return relativePosition;
00152 }

```

## 7.11 cell.h File Reference

```

#include <QVector>
#include <QDebug>
#include <QStack>

```

## Classes

- class [Cell](#)

*Contains the state, the next state and the neighbours.*

## 7.12 cell.h

```

00001 #ifndef CELL_H
00002 #define CELL_H
00003
00004 #include <QVector>
00005 #include <QDebug>
00006 #include <QStack>
00007
00011 class Cell
00012 {
00013 public:
00014     Cell(unsigned int state = 0);
00015
00016     void setState(unsigned int state);
00017     void validState();
00018     void forceState(unsigned int state);
00019     unsigned int getState() const;
00020
00021     bool back();
00022     void reset();
00023
00024     bool addNeighbour(const Cell* neighbour, const QVector<short> relativePosition);
00025     QMap<QVector<short>, const Cell*> getNeighbours() const;
00026     const Cell* getNeighbour(QVector<short> relativePosition) const;
00027
00028     unsigned int countNeighbours(unsigned int filterState) const;
00029     unsigned int countNeighbours() const;
00030
00031     static QVector<short> getRelativePosition(const QVector<unsigned int> cellPosition,
00032 const QVector<unsigned int> neighbourPosition);
00033 private:
00034     QStack<unsigned int> m_states;
00035     unsigned int m_nextState;
00036
00037     QMap<QVector<short>, const Cell*> m_neighbours;
00038 };
00039
00040 #endif // CELL_H

```

## 7.13 cellhandler.cpp File Reference

```

#include <iostream>
#include "cellhandler.h"

```

## 7.14 cellhandler.cpp

```

00001 #include <iostream>
00002 #include "cellhandler.h"
00003
00025 CellHandler::CellHandler(const QString filename)
00026 {
00027     QFile loadFile(filename);
00028     if (!loadFile.open(QIODevice::ReadOnly | QIODevice::Text)) {
00029         qWarning("Couldn't open given file.");
00030         throw QString(QObject::tr("Couldn't open given file"));
00031     }
00032
00033     QJsonParseError parseErr;
00034     QJsonDocument loadDoc(QJsonDocument::fromJson(loadFile.readAll(), &parseErr));

```

```

00035
00036     loadFile.close();
00037
00038
00039     if (loadDoc.isNull() || loadDoc.isEmpty()) {
00040         qWarning() << "Could not read data : ";
00041         qWarning() << parseErr.errorString();
00042         throw QString(parseErr.errorString());
00043     }
00044
00045     // Loading of the json file
00046     if (!load(loadDoc.object()))
00047     {
00048         qWarning("File not valid");
00049         throw QString(QObject::tr("File not valid"));
00050     }
00051
00052     foundNeighbours();
00053
00054
00055 }
00056
00076 CellHandler::CellHandler(const QJsonObject& json)
00077 {
00078     if (!load(json))
00079     {
00080         qWarning("Json not valid");
00081         throw QString(QObject::tr("Json not valid"));
00082     }
00083
00084     foundNeighbours();
00085
00086 }
00087
00088
00098 CellHandler::CellHandler(const QVector<unsigned int> dimensions,
00099     generationTypes type, unsigned int stateMax, unsigned int density)
00100 {
00101     m_dimensions = dimensions;
00102     QVector<unsigned int> position;
00103     unsigned int size = 1;
00104
00105     // Set position vector to 0
00106     for (unsigned short i = 0; i < m_dimensions.size(); i++)
00107     {
00108         position.push_back(0);
00109         size *= m_dimensions.at(i);
00110     }
00111
00112
00113     // Creation of cells
00114     for (unsigned int j = 0; j < size; j++)
00115     {
00116         m_cells.insert(position, new Cell(0));
00117
00118         positionIncrement(position);
00119     }
00120
00121     foundNeighbours();
00122
00123     if (type != empty)
00124         generate(type, stateMax, density);
00125
00126 }
00127
00130 CellHandler::~CellHandler()
00131 {
00132     for (QMap<QVector<unsigned int>, Cell* >::iterator it = m_cells.begin(); it !=
00133         m_cells.end(); ++it)
00134     {
00135         delete it.value();
00136     }
00137
00138
00140 Cell *CellHandler::getCell(const QVector<unsigned int> position) const
00141 {
00142     return m_cells.value(position);
00143 }
00144
00147 unsigned int CellHandler::getMaxState()
00148 {
00149     return QColor::colorNames().size()-2;
00150 }
00151
00154 QVector<unsigned int> CellHandler::getDimensions() const
00155 {

```

```

00156     return m_dimensions;
00157 }
00158
00161 void CellHandler::nextStates() const
00162 {
00163     for (QMap<QVector<unsigned int>, Cell* >::const_iterator it =
m_cells.begin(); it != m_cells.end(); ++it)
00164     {
00165         it.value()->validState();
00166     }
00167 }
00168
00171 bool CellHandler::previousStates() const
00172 {
00173     for (QMap<QVector<unsigned int>, Cell* >::const_iterator it =
m_cells.begin(); it != m_cells.end(); ++it)
00174     {
00175         if (!it.value()->back())
00176             return false;
00177     }
00178     return true;
00179 }
00180
00183 void CellHandler::reset() const
00184 {
00185     for (QMap<QVector<unsigned int>, Cell* >::const_iterator it =
m_cells.begin(); it != m_cells.end(); ++it)
00186     {
00187         it.value()->reset();
00188     }
00189 }
00190
00198 bool CellHandler::save(QString filename) const
00199 {
00200     QFile saveFile(filename);
00201     if (!saveFile.open(QIODevice::WriteOnly)) {
00202         qWarning("Couldn't create or open given file.");
00203         throw QString(QObject::tr("Couldn't create or open given file"));
00204     }
00205
00206     QJsonObject json;
00207     QString stringDimension;
00208     // Creation of the dimension string
00209     for (int i = 0; i < m_dimensions.size(); i++)
00210     {
00211         if (i != 0)
00212             stringDimension.push_back("x");
00213         stringDimension.push_back(QString::number(m_dimensions.at(i)));
00214     }
00215     json["dimensions"] = QJsonValue(stringDimension);
00216
00217     QJsonArray cells;
00218     for (CellHandler::const_iterator it = begin(); it !=
end(); ++it)
00219     {
00220         cells.append(QJsonValue((int)it->getState()));
00221     }
00222     json["cells"] = cells;
00223
00224     //json["maxState"] = QJsonValue((int)m_maxState);
00225
00226     QJsonDocument saveDoc(json);
00227     saveFile.write(saveDoc.toJson());
00228
00229     saveFile.close();
00230     return true;
00231 }
00232 }
00233
00240 void CellHandler::generate(CellHandler::generationTypes
type, unsigned int stateMax, unsigned short density)
00241 {
00242     if (type == random)
00243     {
00244         QVector<unsigned int> position;
00245         for (unsigned short i = 0; i < m_dimensions.size(); i++)
00246         {
00247             position.push_back(0);
00248         }
00249         QRandomGenerator generator((float)qrand()*(float)time_t()/RAND_MAX);
00250         for (int j = 0; j < m_cells.size(); j++)
00251         {
00252             unsigned int state = 0;
00253             // 0 have (1-density)% of chance of being generate
00254             if (generator.generateDouble()*100.0 < density)
00255                 state = (float)generator.generateDouble()*stateMax +1;
00256             if (state > stateMax)

```

```

00257         state = stateMax;
00258         m_cells.value(position)->forceState(state);
00259
00260         positionIncrement(position);
00261     }
00262 }
00263 else if (type == symetric)
00264 {
00265     QVector<unsigned int> position;
00266     for (short i = 0; i < m_dimensions.size(); i++)
00267     {
00268         position.push_back(0);
00269     }
00270
00271     QRandomGenerator generator((float)qrand()*(float)time_t()/RAND_MAX);
00272     QVector<unsigned int> savedStates;
00273     for (int j = 0; j < m_cells.size(); j++)
00274     {
00275         if (j % m_dimensions.at(0) == 0)
00276             savedStates.clear();
00277         if (j % m_dimensions.at(0) < (m_dimensions.at(0)+1) / 2)
00278         {
00279             unsigned int state = 0;
00280             // 0 have (1-density)% of chance of being generate
00281             if (generator.generateDouble()*100.0 < density)
00282                 state = (float)(generator.generateDouble()*stateMax) + 1;
00283             if (state > stateMax)
00284                 state = stateMax;
00285             savedStates.push_back(state);
00286             m_cells.value(position)->forceState(state);
00287         }
00288         else
00289         {
00290             unsigned int i = savedStates.size() - (j % m_dimensions.at(0) - (
00291 m_dimensions.at(0)-1)/2 + (m_dimensions.at(0) % 2 == 0 ? 0 : 1));
00292             m_cells.value(position)->forceState(savedStates.at(i));
00293             positionIncrement(position);
00294         }
00295     }
00296 }
00297 }
00298 }
00299 }
00300
00305 void CellHandler::print(std::ostream &stream) const
00306 {
00307     for (const_iterator it = begin(); it != end(); ++it)
00308     {
00309         for (unsigned int d = 0; d < it->changedDimension(); d++)
00310             stream << std::endl;
00311         stream << it->getState() << " ";
00312     }
00313 }
00314
00315 }
00316
00319 CellHandler::iterator CellHandler::begin()
00320 {
00321     return iterator(this);
00322 }
00323
00326 CellHandler::const_iterator CellHandler::begin() const
00327 {
00328     return const_iterator(this);
00329 }
00330
00335 bool CellHandler::end() const
00336 {
00337     return true;
00338 }
00339
00370 bool CellHandler::load(const QJsonObject &json)
00371 {
00372     if (!json.contains("dimensions") || !json["dimensions"].isString())
00373         return false;
00374
00375     // RegExp to validate dimensions field format : "10x10"
00376     QRegExpValidator dimensionValidator(QRegExp("[0-9]*x[0-9]*"));
00377     QString stringDimensions = json["dimensions"].toString();
00378     int pos = 0;
00379     if (dimensionValidator.validate(stringDimensions, pos) != QRegExpValidator::Acceptable)
00380         return false;
00381
00382     // Split of dimensions field : "10x10" => "10", "10"
00383     QRegExp rx("x");
00384     QStringList list = json["dimensions"].toString().split(rx, QString::SkipEmptyParts);

```

```

00385
00386     int product = 1;
00387     // Dimensions construction
00388     for (int i = 0; i < list.size(); i++)
00389     {
00390         product = product * list.at(i).toInt();
00391         m_dimensions.push_back(list.at(i).toInt());
00392     }
00393     if (!json.contains("cells") || !json["cells"].isArray())
00394         return false;
00395
00396     QJsonArray cells = json["cells"].toArray();
00397     if (cells.size() != product)
00398         return false;
00399
00400     QVector<unsigned int> position;
00401     // Set position vector to 0
00402     for (unsigned short i = 0; i < m_dimensions.size(); i++)
00403     {
00404         position.push_back(0);
00405     }
00406
00407     // Creation of cells
00408     for (int j = 0; j < cells.size(); j++)
00409     {
00410         if (!cells.at(j).isDouble())
00411             return false;
00412         if (cells.at(j).toDouble() < 0)
00413             return false;
00414         m_cells.insert(position, new Cell(cells.at(j).toDouble()));
00415         positionIncrement(position);
00416     }
00417
00418     //if (!json.contains("maxState") || !json["maxState"].isDouble())
00419     //    return false;
00420     //m_maxState = json["maxState"].toInt();
00421
00422     return true;
00423 }
00424
00425 void CellHandler::foundNeighbours()
00426 {
00427     QVector<unsigned int> currentPosition;
00428     // Set position vector to 0
00429     for (unsigned short i = 0; i < m_dimensions.size(); i++)
00430     {
00431         currentPosition.push_back(0);
00432     }
00433     // Modification of all the cells
00434     for (int j = 0; j < m_cells.size(); j++)
00435     {
00436         // Get the list of the neighbours positions
00437         // This function is recursive
00438         QVector<QVector<unsigned int> > listPosition(getListNeighboursPositions(
00439             currentPosition));
00440
00441         // Adding neighbours
00442         for (int i = 0; i < listPosition.size(); i++)
00443         {
00444             m_cells.value(currentPosition)->addNeighbour(m_cells.value(listPosition.at(i)),
00445                 Cell::getRelativePosition(currentPosition, listPosition.at(i)));
00446             positionIncrement(currentPosition);
00447         }
00448     }
00449 }
00450
00451 int CellHandler::positionIncrement(QVector<unsigned int> &pos) const
00452 {
00453     int changedDimension = 0;
00454     pos.replace(0, pos.at(0) + 1);
00455     // Carry management
00456     for (unsigned short i = 0; i < m_dimensions.size(); i++)
00457     {
00458         if (pos.at(i) >= m_dimensions.at(i))
00459         {
00460             pos.replace(i, 0);
00461             changedDimension++;
00462             if (i + 1 != m_dimensions.size())
00463                 pos.replace(i+1, pos.at(i+1)+1);
00464         }
00465     }
00466     return changedDimension;
00467 }
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481

```

```

00482
00488 QVector<QVector<unsigned int> > & CellHandler::getListNeighboursPositions
    (const QVector<unsigned int> position) const
00489 {
00490     QVector<QVector<unsigned int> > *list = getListNeighboursPositionsRecursive
    (position, position.size(), position);
00491     // We remove the position of the cell
00492     list->removeAll(position);
00493     return *list;
00494 }
00495
00529 QVector<QVector<unsigned int> > *
    CellHandler::getListNeighboursPositionsRecursive(const
    QVector<unsigned int> position, unsigned int dimension, QVector<unsigned int> lastAdd) const
00530 {
00531     if (dimension == 0) // Stop condition
00532     {
00533         QVector<QVector<unsigned int> > *list = new QVector<QVector<unsigned int> >;
00534         return list;
00535     }
00536     QVector<QVector<unsigned int> > *listPositions = new QVector<QVector<unsigned int> >;
00537
00538     QVector<unsigned int> modifiedPosition(lastAdd);
00539
00540     // "x_d - 1" tree
00541     if (modifiedPosition.at(dimension-1) != 0) // Avoid "negative" position
00542         modifiedPosition.replace(dimension-1, position.at(dimension-1) - 1);
00543     listPositions->append(*getListNeighboursPositionsRecursive(position,
    dimension -1, modifiedPosition));
00544     if (!listPositions->count(modifiedPosition))
00545         listPositions->push_back(modifiedPosition);
00546
00547     // "x_d" tree
00548     modifiedPosition.replace(dimension-1, position.at(dimension-1));
00549     listPositions->append(*getListNeighboursPositionsRecursive(position,
    dimension -1, modifiedPosition));
00550     if (!listPositions->count(modifiedPosition))
00551         listPositions->push_back(modifiedPosition);
00552
00553     // "x_d + 1" tree
00554     if (modifiedPosition.at(dimension -1) + 1 < m_dimensions.at(dimension-1)) // Avoid position
    out of the cell space
00555         modifiedPosition.replace(dimension-1, position.at(dimension-1) +1);
00556     listPositions->append(*getListNeighboursPositionsRecursive(position,
    dimension -1, modifiedPosition));
00557     if (!listPositions->count(modifiedPosition))
00558         listPositions->push_back(modifiedPosition);
00559
00560     return listPositions;
00561 }
00562 }
00563
00568 template<typename CellHandler_T, typename Cell_T>
00569 CellHandler::iteratorT<CellHandler_T, Cell_T>::iteratorT
    (CellHandler_T *handler):
00570     m_handler(handler), m_changedDimension(0)
00571 {
00572     // Initialisation of m_position
00573     for (unsigned short i = 0; i < handler->m_dimensions.size(); i++)
00574     {
00575         m_position.push_back(0);
00576     }
00577     m_zero = m_position;
00578 }

```

## 7.15 cellhandler.h File Reference

```

#include <QString>
#include <QFile>
#include <QJsonDocument>
#include <QtWidgets>
#include <QMap>
#include <QRegExpValidator>
#include <QDebug>
#include "cell.h"

```

## Classes

- class [CellHandler](#)  
*Cell container and cell generator.*
- class [CellHandler::iteratorT< CellHandler\\_T, Cell\\_T >](#)  
*Implementation of iterator design pattern with a template to generate iterator and const\_iterator at the same time.*

## 7.16 cellhandler.h

```

00001 #ifndef CELLHANDLER_H
00002 #define CELLHANDLER_H
00003
00004 #include <QString>
00005 #include <QFile>
00006 #include <QJsonDocument>
00007 #include <QtWidgets>
00008 #include <QMap>
00009 #include <QRegExpValidator>
00010 #include <QDebug>
00011
00012 #include "cell.h"
00013
00014
00015
00020 class CellHandler
00021 {
00022
00040     template <typename CellHandler_T, typename Cell_T>
00041     class iteratorT
00042     {
00043         friend class CellHandler;
00044     public:
00045         iteratorT(CellHandler_T* handler);
00047         iteratorT& operator++() {
00048             m_changedDimension = m_handler->positionIncrement(
m_position);
00049             // If we return to zero, we have finished
00050             if (m_position == m_zero)
00051                 m_finished = true;
00052
00053             return *this;
00054
00055         }
00057         Cell_T* operator->() const {
00058             return m_handler->m_cells.value(m_position);
00059         }
00061         Cell_T* operator*() const {
00062             return m_handler->m_cells.value(m_position);
00063         }
00064
00065         bool operator!=(bool finished) const { return (m_finished != finished); }
00066         unsigned int changedDimension() const {
00067             return m_changedDimension;
00068         }
00069
00070
00071
00072     private:
00073         CellHandler_T *m_handler;
00074         QVector<unsigned int> m_position;
00075         bool m_finished = false;
00076         QVector<unsigned int> m_zero;
00077         unsigned int m_changedDimension;
00078     };
00079 public:
00080     typedef iteratorT<const CellHandler, const Cell>
const_iterator;
00081     typedef iteratorT<CellHandler, Cell> iterator;
00082
00085     enum generationTypes {
00086         empty,
00087         random,
00088         symetric
00089     };
00090
00091     CellHandler(const QString filename);
00092     CellHandler(const QJsonObject &json);
00093     CellHandler(const QVector<unsigned int> dimensions,

```



```

generationTypes type = empty, unsigned int stateMax = 1, unsigned int density = 20);
00094     virtual ~CellHandler();
00095
00096     Cell* getCell(const QVector<unsigned int> position) const;
00097     static unsigned int getMaxState();
00098     QVector<unsigned int> getDimensions() const;
00099     void nextStates() const;
00100     bool previousStates() const;
00101     void reset() const;
00102
00103     virtual bool save(QString filename) const;
00104
00105     virtual void generate(generationTypes type, unsigned int stateMax = 1, unsigned
short density = 50);
00106     virtual void print(std::ostream &stream) const;
00107
00108     const_iterator begin() const;
00109     iterator begin();
00110     bool end() const;
00111
00112
00113 protected:
00114     virtual bool load(const QJsonObject &json);
00115     virtual void foundNeighbours();
00116     virtual int positionIncrement(QVector<unsigned int> &pos) const;
00117     virtual QVector<QVector<unsigned int> > *getListNeighboursPositionsRecursive
(const QVector<unsigned int> position, unsigned int dimension, QVector<unsigned int> lastAdd) const;
00118     QVector<QVector<unsigned int> > &getListNeighboursPositions(const
QVector<unsigned int> position) const;
00119
00120     QVector<unsigned int> m_dimensions;
00121     QMap<QVector<unsigned int>, Cell* > m_cells;
00122 };
00123
00124 template class CellHandler::iteratorT<CellHandler, Cell>;
00125 template class CellHandler::iteratorT<const CellHandler, const Cell>
;
00126
00127 #endif // CELLHANDLER_H

```

## 7.17 creationdialog.cpp File Reference

```

#include "creationdialog.h"
#include <iostream>

```

## 7.18 creationdialog.cpp

```

00001 #include "creationdialog.h"
00002 #include <iostream>
00003
00006 CreationDialog::CreationDialog(QWidget *parent) : QDialog(parent)
00007 {
00008     QLabel *m_dimLabel= new QLabel(tr("Write your dimensions and their size, separated by a comma.\n"
00009     "For instance, '25,25 ' will create a 2-dimensional 25x25 Automaton. "));
00010     QLabel *m_densityLabel = new QLabel(tr("Density :"));
00011     QLabel *m_stateMaxLabel = new QLabel(tr("Max state :"));
00012     m_densityBox = new QSpinBox();
00013     m_densityBox->setValue(20);
00014     m_stateMaxBox = new QSpinBox();
00015     m_stateMaxBox->setValue(1);
00016
00017     QHBoxLayout *densityLayout = new QHBoxLayout();
00018     densityLayout->addWidget(m_densityLabel);
00019     densityLayout->addWidget(m_densityBox);
00020
00021     QHBoxLayout *stateMaxLayout = new QHBoxLayout();
00022     stateMaxLayout->addWidget(m_stateMaxLabel);
00023     stateMaxLayout->addWidget(m_stateMaxBox);
00024
00025     m_dimensionsEdit = new QLineEdit();
00026     QRegExp rgx("[0-9]+,");
00027     QRegExpValidator *v = new QRegExpValidator(rgx, this);
00028     m_dimensionsEdit->setValidator(v);
00029     m_doneBt = new QPushButton(tr("Done !"));

```

```

00030
00031     QVBoxLayout *layout = new QVBoxLayout;
00032
00033     QGroupBox *grpBox = createGenButtons();
00034
00035     layout->addWidget(m_dimLabel);
00036     layout->addWidget(m_dimensionsEdit);
00037     layout->addLayout(densityLayout);
00038     layout->addLayout(stateMaxLayout);
00039     layout->addWidget(grpBox);
00040     layout->addWidget(m_doneBt);
00041     setLayout(layout);
00042
00043     connect(m_doneBt, SIGNAL(clicked(bool)), this, SLOT(processSettings()));
00044
00045 }
00046
00052 QGroupBox *CreationDialog::createGenButtons(){
00053     m_groupBox = new QGroupBox(tr("Cell generation settings"));
00054     m_empGen = new QRadioButton(tr("&Empty Board"));
00055     m_randGen = new QRadioButton(tr("&Random"));
00056     m_symGen = new QRadioButton(tr("&Symmetrical"));
00057
00058     QVBoxLayout *layout = new QVBoxLayout;
00059     layout->addWidget(m_empGen);
00060     layout->addWidget(m_randGen);
00061     layout->addWidget(m_symGen);
00062
00063     m_groupBox->setLayout(layout);
00064
00065     return m_groupBox;
00066 }
00067
00073 void CreationDialog::processSettings(){
00074     QString dimensions = m_dimensionsEdit->text();
00075     if(dimensions.length() == 0){
00076         QMessageBox messageBox;
00077         messageBox.critical(0,"Error","You must specify valid dimensions !");
00078         messageBox.setFixedSize(500,200);
00079     }
00080     else{
00081         CellHandler::generationTypes genType;
00082         if(m_symGen->isChecked()) genType = CellHandler::generationTypes::symetric;
00083         else if(m_randGen->isChecked()) genType = CellHandler::generationTypes::random;
00084         else genType = CellHandler::generationTypes::empty;
00085         QStringList dimList = m_dimensionsEdit->text().split(",");
00086         QVector<unsigned int> dimensions;
00087         for(int i = 0; i < dimList.size(); i++) dimensions.append(dimList.at(i).toInt());
00088
00089         emit settingsFilled(dimensions, genType, m_stateMaxBox->value(),
00090             m_densityBox->value());
00091         this->close();
00092     }
00093 }
00094

```

## 7.19 creationdialog.h File Reference

```

#include <QtWidgets>
#include "cellhandler.h"

```

### Classes

- class [CreationDialog](#)  
*Automaton creation dialog box.*

## 7.20 creationdialog.h

```

00001 #ifndef CREATIONDIALOG_H
00002 #define CREATIONDIALOG_H
00003
00004 #include <QtWidgets>
00005 #include "cellhandler.h"
00006
00013 class CreationDialog : public QDialog
00014 {
00015     Q_OBJECT
00016
00017 public:
00018     CreationDialog(QWidget *parent = 0);
00019
00020 signals:
00021     void settingsFilled(const QVector<unsigned int> dimensions,
00022                         CellHandler::generationTypes type =
00023                         CellHandler::generationTypes::empty,
00024                         unsigned int stateMax = 1, unsigned int density = 20);
00025
00025 public slots:
00026     void processSettings();
00027
00028 private:
00029     QLineEdit *m_dimensionsEdit;
00030     QSpinBox *m_densityBox;
00031     QSpinBox *m_stateMaxBox;
00032     QPushButton *m_doneBt;
00033
00034     QGroupBox *m_groupBox;
00035     QRadioButton *m_empGen;
00036     QRadioButton *m_randGen;
00037     QRadioButton *m_symGen;
00038
00039     QGroupBox *createGenButtons();
00040
00041
00042
00043
00044
00045
00046 };
00047
00048 #endif // CREATIONDIALOG_H

```

## 7.21 main.cpp File Reference

```

#include <QApplication>
#include <QDebug>
#include "cell.h"
#include "mainwindow.h"
#include "ruleeditor.h"
#include <iostream>

```

### Functions

- int [main](#) (int argc, char \*argv[])

### 7.21.1 Function Documentation

### 7.21.1.1 main()

```
int main (
    int argc,
    char * argv[] )
```

Definition at line 7 of file [main.cpp](#).

## 7.22 main.cpp

```
00001 #include <QApplication>
00002 #include <QDebug>
00003 #include "cell.h"
00004 #include "mainwindow.h"
00005 #include "ruleeditor.h"
00006 #include <iostream>
00007 int main(int argc, char * argv[])
00008 {
00009     QApplication app(argc, argv);
00010     QApplication::setAttribute(Qt::AA_UseHighDpiPixmaps);
00011
00012     app.setOrganizationName("LO21-project");
00013     app.setApplicationName("AutoCell");
00014
00015     MainWindow w;
00016     w.show();
00017
00018     return app.exec();
00019
00020 }
```

## 7.23 mainwindow.cpp File Reference

```
#include "mainwindow.h"
#include <iostream>
#include "math.h"
```

## 7.24 mainwindow.cpp

```
00001 #include "mainwindow.h"
00002 #include <iostream>
00003 #include "math.h"
00004
00007 MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent)
00008 {
00009     createButtons();
00010     createToolBar();
00011
00012
00013     setMinimumSize(500, 500);
00014     setWindowTitle("AutoCell");
00015
00016     m_tabs = NULL;
00017     m_running = false;
00018
00019     QSettings settings;
00020     int nbAutomate = settings.value("nbAutomate").toInt();
00021     for (int i = 0; i < nbAutomate; i++)
00022     {
00023         QString fileName = QString(".automate"+QString::number(i));
00024         try{
00025             AutomateHandler::getAutomateHandler().
addAutomate(new Automate(QString(fileName+".atc"), QString(fileName+".atr")));
00026             if(m_tabs == NULL)
```

```

00027         createTabs();
00028         m_tabs->addTab(createTab(), "Automaton "+ QString::number(
AutomateHandler::getAutomateHandler().getNumberAutomates()));
00029         updateBoard(AutomateHandler::getAutomateHandler().
getNumberAutomates()-1);
00030     }
00031     catch (QString &s)
00032     {
00033         QMessageBox msgBox;
00034         msgBox.warning(0,"Error",s);
00035         msgBox.setFixedSize(500,200);
00036     }
00037     QFile fichier(QString(fileName + ".atc"));
00038     fichier.remove();
00039     fichier.close();
00040     QFile fichier2(QString(fileName + ".atr"));
00041     fichier2.remove();
00042 }
00043 m_zoom->setValue(settings.value("zoom").toInt());
00044 m_timeStep->setValue(settings.value("timestamp").toInt());
00045 }
00046
00051 MainWindow::~MainWindow()
00052 {
00053     // Saving settings for further sessions
00054     QSettings settings;
00055     settings.setValue("nbAutomate", AutomateHandler::getAutomateHandler(
).getNumberAutomates());
00056     settings.setValue("zoom", m_zoom->value());
00057     settings.setValue("timestamp", m_timeStep->value());
00058
00059     for (unsigned int i = 0; i < AutomateHandler::getAutomateHandler().
getNumberAutomates(); i++)
00060     {
00061         AutomateHandler::getAutomateHandler().
getAutomate(i)->saveAll(QString("automate"+QString::number(i)+".atc"), QString("
.automate"+QString::number(i)+".atr"));
00062     }
00063 }
00064 }
00065
00066
00071 void MainWindow::createButtons(){
00072
00073     QPixmap previousStatePm(":/icons/icons/fast-backward.svg");
00074     QPixmap previousStateHoveredPm(":/icons/icons/fast-backward-full.svg");
00075     QPixmap nextStatePm(":/icons/icons/fast-forward.svg");
00076     QPixmap nextStateHoveredPm(":/icons/icons/fast-forward-full.svg");
00077     QPixmap playPm(":/icons/icons/play.svg");
00078     QPixmap playHoveredPm(":/icons/icons/play-full.svg");
00079     QPixmap newPm(":/icons/icons/new.svg");
00080     QPixmap openPm(":/icons/icons/open.svg");
00081     QPixmap savePm(":/icons/icons/save.svg");
00082     QPixmap pausePm(":/icons/icons/pause.svg");
00083     QPixmap resetPm(":/icons/icons/reset.svg");
00084
00085     QIcon previousStateIcon;
00086     QIcon nextStateIcon;
00087     QIcon newIcon;
00088     QIcon saveIcon;
00089     QIcon openIcon;
00090     QIcon resetIcon;
00091
00092     previousStateIcon.addPixmap(previousStatePm, QIcon::Normal, QIcon::Off);
00093     previousStateIcon.addPixmap(previousStateHoveredPm, QIcon::Active, QIcon::Off);
00094     nextStateIcon.addPixmap(nextStatePm, QIcon::Normal, QIcon::Off);
00095     nextStateIcon.addPixmap(nextStateHoveredPm, QIcon::Active, QIcon::Off);
00096     m_playIcon.addPixmap(playPm, QIcon::Normal, QIcon::Off);
00097     m_playIcon.addPixmap(playHoveredPm, QIcon::Active, QIcon::Off);
00098     m_pauseIcon.addPixmap(pausePm, QIcon::Normal, QIcon::Off);
00099     newIcon.addPixmap(newPm, QIcon::Normal, QIcon::Off);
00100     saveIcon.addPixmap(savePm, QIcon::Normal, QIcon::Off);
00101     openIcon.addPixmap(openPm, QIcon::Normal, QIcon::Off);
00102     resetIcon.addPixmap(resetPm, QIcon::Normal, QIcon::Off);
00103
00104     QAction *playPause = new QAction(m_playIcon, tr("Play"), this);
00105     QAction *nextState = new QAction(nextStateIcon, tr("&Next state"), this);
00106     QAction *previousState = new QAction(previousStateIcon, tr("&Previous state"), this);
00107     QAction *openAutomaton = new QAction(openIcon, tr("Open automaton"), this);
00108     QAction *saveAutomaton = new QAction(saveIcon, tr("Save automaton"), this);
00109     QAction *newAutomaton = new QAction(newIcon, tr("New automaton"), this);
00110     QAction *resetAutomaton = new QAction(resetIcon, tr("Reset automaton"), this);
00111
00112     m_previousStateBt = new QPushButton(this);
00113     m_nextStateBt = new QPushButton(this);
00114     m_playPauseBt = new QPushButton(this);
00115     m_saveAutomatonBt = new QPushButton(this);

```

```

00116     m_newAutomatonBt = new QPushButton(this);
00117     m_openAutomatonBt = new QPushButton(this);
00118     m_resetBt = new QPushButton(this);
00119
00120     m_previousStateBt->setDefaultAction(previousState);
00121     m_nextStateBt->setDefaultAction(nextState);
00122     m_playPauseBt->setDefaultAction(playPause);
00123     m_saveAutomatonBt->setDefaultAction(saveAutomaton);
00124     m_newAutomatonBt->setDefaultAction(newAutomaton);
00125     m_openAutomatonBt->setDefaultAction(openAutomaton);
00126     m_resetBt->setDefaultAction(resetAutomaton);
00127
00128     m_previousStateBt->setIconSize(QSize(30,30));
00129     m_nextStateBt->setIconSize(QSize(30,30));
00130     m_playPauseBt->setIconSize(QSize(30,30));
00131     m_saveAutomatonBt->setIconSize(QSize(30,30));
00132     m_newAutomatonBt->setIconSize(QSize(30,30));
00133     m_openAutomatonBt->setIconSize(QSize(30,30));
00134     m_resetBt->setIconSize(QSize(30,30));
00135
00136
00137     m_zoom = new QSlider(Qt::Horizontal);
00138     m_zoom->setValue(m_cellSize);
00139     m_zoom->setMinimum(4);
00140     m_zoom->setMaximum(100);
00141     m_zoom->setFixedWidth(100);
00142
00143
00144     connect(m_openAutomatonBt, SIGNAL(clicked(bool)), this, SLOT(
00145         openFile()));
00146     connect(m_newAutomatonBt, SIGNAL(clicked(bool)), this, SLOT(
00147         openCreationWindow()));
00148     connect(m_saveAutomatonBt, SIGNAL(clicked(bool)), this, SLOT(
00149         saveToFile()));
00150     connect(m_nextStateBt, SIGNAL(clicked(bool)), this, SLOT(
00151         forward()));
00152     connect(m_previousStateBt, SIGNAL(clicked(bool)), this, SLOT(
00153         backward()));
00154     connect(m_playPauseBt, SIGNAL(clicked(bool)), this, SLOT(
00155         handlePlayPause()));
00156     connect(m_resetBt, SIGNAL(clicked(bool)), this, SLOT(reset()));
00157     connect(m_zoom, SIGNAL(valueChanged(int)), this, SLOT(setSize(int)));
00158 }
00159 void MainWindow::createToolBar(){
00160     m_toolBar = new QToolBar(this);
00161     QLabel *timeStepLabel = new QLabel(tr("Timestep(ms)"),this);
00162     m_timeStep = new QSpinBox(this);
00163     m_timeStep->setMaximum(10000);
00164     m_timeStep->setValue(500);
00165     timeStepLabel->setFixedWidth(90);
00166     m_timeStep->setFixedWidth(60);
00167     m_toolBar->setMovable(false);
00168
00169     QLabel *cellSetterLabel = new QLabel(tr("Cell value"));
00170     m_cellSetter = new QSpinBox(this);
00171     connect(m_cellSetter, SIGNAL(valueChanged(int)),this, SLOT(
00172         changeCellValue()));
00173     QLabel *zoomLabel = new QLabel(tr("Zoom"),this);
00174     QVBoxLayout* zoomLayout = new QVBoxLayout();
00175     zoomLayout->addWidget(zoomLabel, Qt::AlignCenter);
00176     zoomLayout->addWidget(m_zoom, Qt::AlignCenter);
00177
00178     QVBoxLayout* tsLayout = new QVBoxLayout();
00179     tsLayout->addWidget(timeStepLabel, Qt::AlignCenter);
00180     tsLayout->addWidget(m_timeStep, Qt::AlignCenter);
00181
00182     QVBoxLayout* csLayout = new QVBoxLayout();
00183     csLayout->addWidget(cellSetterLabel, Qt::AlignCenter);
00184     csLayout->addWidget(m_cellSetter, Qt::AlignCenter);
00185
00186     QHBoxLayout *tbLayout = new QHBoxLayout(this);
00187     tbLayout->addLayout(zoomLayout);
00188     tbLayout->addWidget(m_newAutomatonBt, Qt::AlignCenter);
00189     tbLayout->addWidget(m_openAutomatonBt, Qt::AlignCenter);
00190     tbLayout->addWidget(m_saveAutomatonBt, Qt::AlignCenter);
00191     tbLayout->addWidget(m_previousStateBt, Qt::AlignCenter);
00192     tbLayout->addWidget(m_playPauseBt, Qt::AlignCenter);
00193     tbLayout->addWidget(m_nextStateBt, Qt::AlignCenter);
00194     tbLayout->addWidget(m_resetBt, Qt::AlignCenter);
00195     tbLayout->addLayout(tsLayout);
00196     tbLayout->addLayout(csLayout);
00197
00198
00199     tbLayout->setAlignment(Qt::AlignCenter);

```

```

00200     QWidget* wrapper = new QWidget(this);
00201     wrapper->setLayout(tblLayout);
00202     m_toolBar->addWidget(wrapper);
00203     addToolBar(m_toolBar);
00204
00205
00206 }
00207
00212 QWidget* MainWindow::createTab(){
00213     QWidget *tab = new QWidget(this);
00214     QVBoxLayout *layout = new QVBoxLayout(this);
00215     QVector<unsigned int> dimensions = AutomateHandler::getAutomateHandler
00216     ().getAutomate(AutomateHandler::getAutomateHandler()).
00217     getNumberAutomates()-1->getCellHandler().getDimensions();
00218     int boardVSize = 0;
00219     int boardHSize = 0;
00220     if(dimensions.size() > 1){
00221         boardVSize = dimensions[0];
00222         boardHSize = dimensions[1];
00223     }
00224     else{
00225         boardVSize = 1;
00226         boardHSize = dimensions[0];
00227     }
00228     QTableWidgetItem* board = new QTableWidgetItem(boardVSize, boardHSize, this);
00229     board->setFixedSize(boardHSize*m_cellSize,boardVSize*
00230     m_cellSize);
00231     //setMinimumSize(m_boardHSize*m_cellSize,100+m_boardVSize*m_cellSize);
00232     board->horizontalHeader()->setVisible(false);
00233     board->verticalHeader()->setVisible(false);
00234     board->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
00235     board->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
00236     board->setEditTriggers(QAbstractItemView::NoEditTriggers);
00237     for(int col = 0; col < boardHSize; ++col)
00238         board->setColumnWidth(col, m_cellSize);
00239     for(int row = 0; row < boardVSize; ++row) {
00240         board->setRowHeight(row, m_cellSize);
00241         for(int col = 0; col < boardHSize; ++col) {
00242             board->setItem(row, col, new QTableWidgetItem(""));
00243             board->item(row, col)->setBackgroundColor("white");
00244             board->item(row, col)->setTextColor("black");
00245         }
00246     }
00247     QScrollArea *scrollArea = new QScrollArea(this);
00248     scrollArea->setWidget(board);
00249
00250     layout->setContentsMargins(0,0,0,0);
00251     layout->addWidget(scrollArea);
00252     tab->setLayout(layout);
00253     connect(board, SIGNAL(cellClicked(int,int)), this, SLOT(cellPressed(int,int)));
00254     return tab;
00255 }
00256
00258 void MainWindow::openFile(){
00259     QString fileName = QFileDialog::getOpenFileName(this, tr("Open Cell file"), ".",
00260     tr("Automaton cell files (*.atc)"));
00261     if(!fileName.isEmpty()){
00262         AutomateHandler::getAutomateHandler().
00263         addAutomate(new Automate(fileName));
00264         if(m_tabs == NULL) createTabs();
00265         m_tabs->addTab(createTab(), "Automaton "+ QString::number(
00266         AutomateHandler::getAutomateHandler().getNumberAutomates()+1));
00267         updateBoard(AutomateHandler::getAutomateHandler().
00268         getNumberAutomates()-1);
00269     }
00270     RuleEditor* ruleEditor = new RuleEditor(
00271     AutomateHandler::getAutomateHandler().getAutomate(
00272     AutomateHandler::getAutomateHandler().getNumberAutomates()-1->
00273     getCellHandler().getDimensions().size(), this);
00274     connect(ruleEditor, SIGNAL(fileImported(QString)),this,SLOT(
00275     addAutomatonRuleFile(QString)));
00276     connect(ruleEditor, SIGNAL(rulesFilled(QList<const NeighbourRule*>)), this, SLOT(
00277     addAutomatonRules(QList<const NeighbourRule*>)));
00278     ruleEditor->show();
00279 }
00280
00281 void MainWindow::saveToFile(){
00282     if(AutomateHandler::getAutomateHandler().getNumberAutomates() > 0){
00283         QString automatonFileName = QFileDialog::getSaveFileName(this, tr("Save Automaton cell
00284         configuration"),
00285         ".", tr("Automaton Cells file (*.atc)"));
00286         AutomateHandler::getAutomateHandler().
00287         getAutomate(m_tabs->currentIndex()->saveCells(automatonFileName+".atc"));
00288         QString ruleFileName = QFileDialog::getSaveFileName(this, tr("Save Automaton rules"),

```

```

00284                                     ".", tr("Automaton Rules file (*.atr));
00285     AutomateHandler::getAutomateHandler().
getAutomate(m_tabs->currentIndex())->saveRules(ruleFileName+".atr");
00286 }
00287 else{
00288     QMessageBox msgBox;
00289     msgBox.critical(0,"Error","Please create or import an Automaton first !");
00290     msgBox.setFixedSize(500,200);
00291 }
00292 }
00293
00298 void MainWindow::openCreationWindow(){
00299     CreationDialog *window = new CreationDialog(this);
00300     connect(window, SIGNAL(settingsFilled(QVector<uint>,
CellHandler::generationTypes,uint,uint)),
00301         this, SLOT(receiveCellHandler(QVector<uint>,
CellHandler::generationTypes,uint,uint)));
00302     window->show();
00303 }
00304
00311 void MainWindow::receiveCellHandler(const QVector<unsigned int> dimensions,
00312                                     CellHandler::generationTypes type,
00313                                     unsigned int stateMax, unsigned int density){
00314     AutomateHandler::getAutomateHandler().
addAutomate(new Automate(dimensions, type, stateMax, density));
00315
00316     if(m_tabs == NULL) createTabs();
00317     QWidget* newTab = createTab();
00318     m_tabs->addTab(newTab, "Automaton "+ QString::number(
AutomateHandler::getAutomateHandler().getNumberAutomates()));
00319     m_tabs->setCurrentWidget(newTab);
00320     updateBoard(AutomateHandler::getAutomateHandler().
getNumberAutomates()-1);
00321
00322     RuleEditor* ruleEditor = new RuleEditor(
AutomateHandler::getAutomateHandler().getAutomate(
AutomateHandler::getAutomateHandler().getNumberAutomates()-1)->
getCellHandler().getDimensions().size(), this);
00323     connect(ruleEditor, SIGNAL(fileImported(QString)),this,SLOT(
addAutomatonRuleFile(QString)));
00324     connect(ruleEditor, SIGNAL(rulesFilled(QList<const Rule*>)), this, SLOT(
addAutomatonRules(QList<const Rule*>)));
00325     ruleEditor->show();
00326 }
00327 }
00328
00333 void MainWindow::nextState(unsigned int n){
00334     if(AutomateHandler::getAutomateHandler().getNumberAutomates()== 0){
00335         QMessageBox msgBox;
00336         msgBox.critical(0,"Error","Please create or import an Automaton first !");
00337         msgBox.setFixedSize(500,200);
00338     }
00339     else{
00340
00341         AutomateHandler::getAutomateHandler().
getAutomate(m_tabs->currentIndex())->run(n);
00342         updateBoard(m_tabs->currentIndex());
00343     }
00344 }
00345
00350 void MainWindow::updateBoard(int index){
00351     if(AutomateHandler::getAutomateHandler().getNumberAutomates()== 0){
00352         QMessageBox msgBox;
00353         msgBox.critical(0,"Error","Please create or import an Automaton first !");
00354         msgBox.setFixedSize(500,200);
00355     }
00356     else{
00357
00358         const CellHandler* cellHandler = &(
AutomateHandler::getAutomateHandler().
getAutomate(index)->getCellHandler());
00359         QVector<unsigned int> dimensions = cellHandler->getDimensions();
00360         QTableWidgetItem* board = getBoard(index);
00361         if(dimensions.size() > 1){
00362             int i = 0;
00363             int j = 0;
00364             for (CellHandler::const_iterator it = cellHandler->
begin(); it != cellHandler->end() && it.changedDimension() < 2; ++it){
00365                 if(it.changedDimension() > 0){
00366                     i = 0;
00367                     j++;
00368                 }
00369                 board->item(i,j)->setBackgroundColor(getColor(it->getState()));
00370                 i++;
00371             }
00372         }
00373         else{ // dimension = 1

```



```

00374         if (board->rowCount() != 1)
00375             addEmptyRow(index);
00376         int i = board->rowCount() -1;
00377         int j = 0;
00378         for (CellHandler::const_iterator it = cellHandler->
begin(); it != cellHandler->end() && it->changedDimension() < 1; ++it){
00379             board->item(i,j)->setBackgroundColor(getColor(it->getState()));
00380             j++;
00381         }
00382         if (board->rowCount() == 1)
00383             addEmptyRow(index);
00384
00385         // Go to bottom
00386         QScrollArea *scrool = static_cast<QScrollArea*>(m_tabs->currentWidget()->layout()->itemAt
(0)->widget());
00387
00388         scrool->verticalScrollBar()->setSliderPosition(scrool->verticalScrollBar()->maximum());
00389
00390     }
00391 }
00392 }
00393 }
00394 }
00395
00400 void MainWindow::forward(){
00401     nextState(1);
00402 }
00403
00407 QTableWidgetItem* MainWindow::getBoard(int n){
00408     return m_tabs->widget(n)->findChild<QTableWidgetItem *>();
00409 }
00410
00415 QColor MainWindow::getColor(int cellState)
00416 {
00417     if (cellState > QColor::colorNames().size() -2)
00418         return Qt::black;
00419     switch (cellState)
00420     {
00421     case 0:
00422         return Qt::white;
00423     case 1:
00424         return Qt::black;
00425     case 2:
00426         return Qt::red;
00427     case 3:
00428         return Qt::green;
00429     case 4:
00430         return Qt::blue;
00431     case 5:
00432         return Qt::yellow;
00433     case 6:
00434         return QColor(170, 110, 40); // brown
00435     case 7:
00436         return QColor(145,30, 180); // purple
00437     case 8:
00438         return QColor(245,130,48); // orange
00439     case 9:
00440         return Qt::cyan;
00441     case 10:
00442         return Qt::magenta;
00443     case 11:
00444         return QColor(210, 245, 60); // Lime
00445     case 12:
00446         return QColor(250, 190, 190); // pink
00447     case 13:
00448         return QColor(0,128,128); // Teal
00449     case 14:
00450         return QColor(230, 190, 255); // Lavender
00451     case 15:
00452         return QColor(255, 250, 200); // beige
00453     case 16:
00454         return QColor(128, 0,0); // Maroon
00455     case 17:
00456         return QColor(170, 255, 195); // Mint
00457     case 18:
00458         return QColor(128, 128, 0); // Olive
00459     case 19:
00460         return QColor(255, 215, 180); // Coral
00461     case 20:
00462         return QColor(0,0,128); // Navy
00463     case 21:
00464         return Qt::gray;
00465     }
00466 }
00467
00468
00469     return hookMoreColor(cellState);

```

```

00470 }
00471
00476 void MainWindow::createTabs(){
00477     m_tabs = new QTabWidget(this);
00478     m_tabs->setMovable(true);
00479     m_tabs->setTabsClosable(true);
00480     setCentralWidget(m_tabs);
00481     connect(m_tabs, SIGNAL(tabCloseRequested(int)), this, SLOT(closeTab(int)));
00482     connect(m_tabs, SIGNAL(currentChanged(int)), this, SLOT(
        handleTabChanged()));
00483 }
00484
00491 void MainWindow::addEmptyRow(unsigned int n)
00492 {
00493     QTableWidget *board = getBoard(n);
00494     board->setFixedHeight(board->height() + m_cellSize);
00495     unsigned int row = board->rowCount();
00496     board->insertRow(row);
00497     board->setRowHeight(row, m_cellSize);
00498     for(int col = 0; col < board->columnCount(); ++col) {
00499         board->setItem(row, col, new QTableWidgetItem(""));
00500         board->item(row, col)->setBackgroundColor("white");
00501         board->item(row, col)->setTextColor("black");
00502     }
00503 }
00504
00509 void MainWindow::closeTab(int n){
00510     m_tabs->setCurrentIndex(n);
00511     saveToFile();
00512     AutomateHandler::getAutomateHandler().
        deleteAutomate(AutomateHandler::getAutomateHandler().
            getAutomate(n));
00513     m_tabs->removeTab(n);
00514 }
00515
00520 void MainWindow::addAutomatonRules(QList<const Rule *> rules){
00521     for(int i = 0 ; i < rules.size();i++)
00522     {
00523         AutomateHandler::getAutomateHandler().
            getAutomate(AutomateHandler::getAutomateHandler().
                getNumberAutomates()-1)->addRule(rules.at(i));
00524     }
00525 }
00526
00531 void MainWindow::addAutomatonRuleFile(QString path){
00532     AutomateHandler::getAutomateHandler().
        getAutomate(AutomateHandler::getAutomateHandler().
            getNumberAutomates()-1)->addRuleFile(path);
00533 }
00534
00539 void MainWindow::handlePlayPause(){
00540     if(AutomateHandler::getAutomateHandler().getNumberAutomates()== 0){
00541         QMessageBox msgBox;
00542         msgBox.critical(0,"Error","Please create or import an Automaton first !");
00543         msgBox.setFixedSize(500,200);
00544     }
00545     else{
00546         if(m_running){
00547             m_playPauseBt->setIcon(m_playIcon);
00548             delete m_timer;
00549         }
00550         else {
00551             m_playPauseBt->setIcon(m_pauseIcon);
00552             m_timer = new QTimer(this);
00553             connect(m_timer, SIGNAL(timeout()), this, SLOT(runAutomaton()));
00554             m_timer->start(m_timeStep->value());
00555         }
00556         m_running = !m_running;
00557     }
00558
00559
00560 }
00561
00566 void MainWindow::runAutomaton(){
00567     if(m_running){
00568         AutomateHandler::getAutomateHandler().
            getAutomate(m_tabs->currentIndex()->run();
00569         QCoreApplication::processEvents();
00570         updateBoard(m_tabs->currentIndex());
00571         QCoreApplication::processEvents();
00572     }
00573 }
00574
00578 void MainWindow::reset(){
00579     if(AutomateHandler::getAutomateHandler().getNumberAutomates()== 0){
00580         QMessageBox msgBox;
00581         msgBox.critical(0,"Error","Please create or import an Automaton first !");

```

```

00582         msgBox.setFixedSize(500,200);
00583     }
00584     else{
00585         //QTableWidget *board = getBoard(m_tabs->currentIndex());
00586         //board->setRowCount(1);
00587         //board->setFixedHeight(m_cellSize);
00588
00589         AutomateHandler::getAutomateHandler().
getAutomate(m_tabs->currentIndex()->getCellHandler().reset();
00590         if (AutomateHandler::getAutomateHandler().getAutomate(
m_tabs->currentIndex()->getCellHandler().getDimensions().size() == 1)
00591         {
00592             QTableWidget *board = getBoard(m_tabs->currentIndex());
00593             board->setRowCount(0);
00594             board->setFixedHeight(0);
00595         }
00596         updateBoard(m_tabs->currentIndex());
00597     }
00598 }
00599
00600
00605 void MainWindow::backward() {
00606     AutomateHandler::getAutomateHandler().
getAutomate(m_tabs->currentIndex()->getCellHandler().previousStates();
00607     updateBoard(m_tabs->currentIndex());
00608 }
00609
00614 void MainWindow::cellPressed(int i, int j){
00615     QVector<unsigned int> coord;
00616
00617     m_currentCellX = i;
00618     m_currentCellY = j;
00619     const CellHandler* cellHandler = &(
AutomateHandler::getAutomateHandler().
getAutomate(m_tabs->currentIndex()->getCellHandler());
00620     if (cellHandler->getDimensions().size() > 1){
00621         coord.append(i);
00622         coord.append(j);
00623         m_cellSetter->setValue(cellHandler->getCell(coord)->
getState());
00624     }
00625     else{
00626         coord.append(j);
00627         m_cellSetter->setValue(cellHandler->getCell(coord)->
getState());
00628     }
00629 }
00630
00631
00636 void MainWindow::changeCellValue() {
00637     if (AutomateHandler::getAutomateHandler().getNumberAutomates() == 0) {
00638         QMessageBox msgBox;
00639         msgBox.critical(0, "Error", "Please create or import an Automaton first !");
00640         msgBox.setFixedSize(500,200);
00641     }
00642     else{
00643         if (m_currentCellX > -1 && m_currentCellY > -1) {
00644             const CellHandler* cellHandler = &(
AutomateHandler::getAutomateHandler().
getAutomate(m_tabs->currentIndex()->getCellHandler());
00645             QVector<unsigned int> coord;
00646             if (cellHandler->getDimensions().size() > 1) {
00647                 coord.append(m_currentCellX);
00648                 coord.append(m_currentCellY);
00649                 cellHandler->getCell(coord)->forceState(
m_cellSetter->value());
00650                 updateBoard(m_tabs->currentIndex());
00651             }
00652             else{
00653                 coord.append(m_currentCellY);
00654                 cellHandler->getCell(coord)->forceState(
m_cellSetter->value());
00655                 QTableWidget *board = getBoard(m_tabs->currentIndex());
00656                 int i = 0;
00657                 int j = 0;
00658                 for (CellHandler::const_iterator it = cellHandler->
begin(); it != cellHandler->end() && it->changedDimension() < 1; ++it){
00659                     board->item(i, j)->setBackgroundColor(getColor(it->getState()));
00660                     j++;
00661                 }
00662             }
00663         }
00664     }
00665 }
00666 }
00667
00672 void MainWindow::handleTabChanged() {

```

```

00673     if (m_tabs->currentIndex() >= 0) {
00674         m_cellSetter->setMaximum(CellHandler::getMaxState());
00675         m_currentCellX = -1;
00676         m_currentCellY = -1;
00677         if (m_running) {
00678             m_playPauseBt->setIcon(m_playIcon);
00679             delete m_timer;
00680             m_running = !m_running;
00681         }
00682     }
00683 }
00684 }
00685
00689 void MainWindow::setSize(int newCellSize)
00690 {
00691     m_cellSize = newCellSize;
00692     if (AutomateHandler::getAutomateHandler().getNumberAutomates() != 0)
00693     {
00694         for (int i = 0; i < m_tabs->count(); i++)
00695         {
00696             QTableWidget* board = getBoard(i);
00697             if (m_cellSize < 10)
00698                 board->setShowGrid(false);
00699             else
00700                 board->setShowGrid(true);
00701             for (int row = 0; row < board->rowCount(); row++)
00702                 board->setRowHeight(row, m_cellSize);
00703             for (int col = 0; col < board->columnCount(); col++)
00704                 board->setColumnWidth(col, m_cellSize);
00705             board->setFixedSize(board->columnCount() * m_cellSize, board->rowCount() *
m_cellSize);
00706         }
00707     }
00708 }

```

## 7.25 mainwindow.h File Reference

```

#include <QMainWindow>
#include <QtWidgets>
#include "cellhandler.h"
#include "automate.h"
#include "creationdialog.h"
#include "automatehandler.h"
#include "ruleeditor.h"

```

### Classes

- class [MainWindow](#)  
*Simulation window.*

## 7.26 mainwindow.h

```

00001 #ifndef MAINWINDOW_H
00002 #define MAINWINDOW_H
00003
00004 #include <QMainWindow>
00005 #include <QtWidgets>
00006 #include "cellhandler.h"
00007 #include "automate.h"
00008 #include "creationdialog.h"
00009 #include "automatehandler.h"
00010 #include "ruleeditor.h"
00011
00018 class MainWindow : public QMainWindow
00019 {
00020     Q_OBJECT

```

```

00021
00022     QTabWidget *m_tabs;
00023
00024     //Icons saved for reuse
00025     QIcon m_playIcon;
00026     QIcon m_pauseIcon;
00027
00028     //Buttons
00029     QToolButton *m_playPauseBt;
00030     QToolButton *m_nextStateBt;
00031     QToolButton *m_previousStateBt;
00032     QToolButton *m_openAutomatonBt;
00033     QToolButton *m_saveAutomatonBt;
00034     QToolButton *m_newAutomatonBt;
00035     QToolButton *m_resetBt;
00036
00037
00038     QSpinBox *m_timeStep;
00039     QSpinBox *m_cellSetter;
00040     QTimer* m_timer;
00041
00042     QSlider *m_zoom;
00043
00044     bool m_running;
00045     QToolBar *m_toolBar;
00046
00047     int m_currentCellX;
00048     int m_currentCellY;
00049
00050     // Board size settings
00051     unsigned int m_boardHSize = 25;
00052     unsigned int m_boardVSize = 25;
00053     unsigned int m_cellSize = 30;
00054
00055     void createButtons();
00056     void createToolBar();
00057     void createBoard();
00058     QWidget* createTab();
00059     void createTabs();
00060
00061     void addEmptyRow(unsigned int n);
00062     void updateBoard(int index);
00063     void nextState(unsigned int n);
00064     QTableWidgetItem* getBoard(int n);
00065
00066     static virtual QColor getColor(int cellState);
00069     static virtual QColor hookMoreColor(int cellState) { return QColor((Qt::GlobalColor) (
cellState +2));};
00070
00071
00072 public:
00073     explicit MainWindow(QWidget *parent = nullptr);
00074     virtual ~MainWindow();
00075
00076 signals:
00077
00078 public slots:
00079     void openFile();
00080     void saveToFile();
00081     void openCreationWindow();
00082     void receiveCellHandler(const QVector<unsigned int> dimensions,
00083                             CellHandler::generationTypes type =
00084                             CellHandler::generationTypes::empty,
00085                             unsigned int stateMax = 1, unsigned int density = 20);
00086     void addAutomatonRules(QList<const Rule *> rules);
00087     void addAutomatonRuleFile(QString path);
00088     void forward();
00089     void backward();
00090     void closeTab(int n);
00091     void runAutomaton();
00092     void handlePlayPause();
00093     void reset();
00094     void cellPressed(int i, int j);
00095     void changeCellValue();
00096     void handleTabChanged();
00097     void setSize(int newCellSize);
00098 };
00099
00100 #endif // MAINWINDOW_H

```

## 7.27 matrixrule.cpp File Reference

```
#include "matrixrule.h"
```

### Functions

- `QVector< unsigned int > fillInterval (unsigned int min, unsigned int max)`  
Returns a vector fill of the integers between min and max (all included)

### 7.27.1 Function Documentation

#### 7.27.1.1 fillInterval()

```
QVector<unsigned int> fillInterval (
    unsigned int min,
    unsigned int max )
```

Returns a vector fill of the integers between min and max (all included)

#### Returns

Interval

#### Parameters

<i>min</i>	Minimal value (included)
<i>max</i>	Maximal value (included)

Definition at line 8 of file [matrixrule.cpp](#).

## 7.28 matrixrule.cpp

```
00001 #include "matrixrule.h"
00002
00008 QVector<unsigned int> fillInterval(unsigned int min, unsigned int max)
00009 {
00010     QVector<unsigned int> interval;
00011     for (unsigned int i = min; i <= max ; i++)
00012         interval.push_back(i);
00013
00014     return interval;
00015 }
00016
00021 MatrixRule::MatrixRule(unsigned int finalState, QVector<unsigned int> currentStates)
00022 :
00023     Rule(currentStates, finalState)
00024 {
00025 }
```

```

00030 bool MatrixRule::matchCell(const Cell *cell) const
00031 {
00032     // Check cell state
00033     if (!m_currentCellPossibleValues.contains(cell->
00034         getState()))
00035     {
00036         return false;
00037     }
00038     // Check neighbours
00039     bool matched = true;
00040     // Rappel : QMap<relativePosition, possibleStates>
00041     for (QMap<QVector<short>, QVector<unsigned int> >::const_iterator it =
00042         m_matrix.begin(); it != m_matrix.end(); ++it)
00043     {
00044         if (cell->getNeighbour(it.key()) != nullptr) // Border management
00045         {
00046             if (!it.value().contains(cell->getNeighbour(it.key())->getState()))
00047             {
00048                 matched = false;
00049                 break;
00050             }
00051         }
00052         else
00053         {
00054             if (!it.value().contains(0))
00055             {
00056                 matched = false;
00057                 break;
00058             }
00059         }
00060     }
00061     return matched;
00062 }
00063
00064 void MatrixRule::addNeighbourState(QVector<short> relativePosition, unsigned
00065     int matchState)
00066 {
00067     m_matrix[relativePosition].push_back(matchState);
00068 }
00069
00070 void MatrixRule::addNeighbourState(QVector<short> relativePosition,
00071     QVector<unsigned int> matchStates)
00072 {
00073     for (QVector<unsigned int>::const_iterator it = matchStates.begin(); it != matchStates.end(); ++it)
00074     {
00075         m_matrix[relativePosition].push_back(*it);
00076     }
00077 }
00078
00079 QJsonObject MatrixRule::toJson() const
00080 {
00081     QJsonObject object(Rule::toJson());
00082     object.insert("type", QJsonValue("matrix"));
00083     QJsonArray neighbours;
00084     for (QMap<QVector<short>, QVector<unsigned int> >::const_iterator it =
00085         m_matrix.begin(); it != m_matrix.end(); ++it)
00086     {
00087         QJsonObject aNeighbour;
00088         QJsonArray relativePosition;
00089         for (unsigned int i = 0; i < it.key().size(); i++)
00090         {
00091             relativePosition.append(QJsonValue((int) it.key().at(i)));
00092         }
00093         aNeighbour.insert("relativePosition", relativePosition);
00094         QJsonArray neighbourStates;
00095         for (unsigned int i = 0; i < it.value().size(); i++)
00096         {
00097             neighbourStates.append(QJsonValue((int) it.value().at(i)));
00098         }
00099         aNeighbour.insert("neighbourStates", neighbourStates);
00100         neighbours.append(aNeighbour);
00101     }
00102     object.insert("neighbours", neighbours);
00103     return object;
00104 }
00105
00106 }

```

## 7.29 matrixrule.h File Reference

```
#include <QVector>
#include <QMap>
#include "cell.h"
#include "rule.h"
```

### Classes

- class [MatrixRule](#)  
*Manage specific rules, about specific values of specific neighbour.*

### Functions

- [QVector< unsigned int > fillInterval](#) (unsigned int min, unsigned int max)  
*Returns a vector fill of the integers between min and max (all included)*

### 7.29.1 Function Documentation

#### 7.29.1.1 fillInterval()

```
QVector<unsigned int> fillInterval (
    unsigned int min,
    unsigned int max )
```

Returns a vector fill of the integers between min and max (all included)

#### Returns

Interval

#### Parameters

<i>min</i>	Minimal value (included)
<i>max</i>	Maximal value (included)

Definition at line 8 of file [matrixrule.cpp](#).

## 7.30 matrixrule.h

```
00001 #ifndef MATRIXRULE_H
00002 #define MATRIXRULE_H
```



```

00003
00004 #include <QVector>
00005 #include <QMap>
00006 #include "cell.h"
00007 #include "rule.h"
00008
00009 QVector<unsigned int> fillInterval(unsigned int min, unsigned int max);
00010
00013 class MatrixRule : public Rule
00014 {
00015     public:
00016         MatrixRule(unsigned int finalState, QVector<unsigned int> currentStates =
00017             QVector<unsigned int>());
00018
00019         virtual bool matchCell(const Cell* cell) const;
00020         virtual void addNeighbourState(QVector<short> relativePosition, unsigned int
00021             matchState);
00022         virtual void addNeighbourState(QVector<short> relativePosition, QVector<unsigned
00023             int> matchStates);
00024
00025         QJsonObject toJson() const;
00026
00027     protected:
00028         QMap<QVector<short>, QVector<unsigned int> > m_matrix;
00029 };
00030
00031
00032
00033 #endif // MATRIXRULE_H

```

## 7.31 neighbourrule.cpp File Reference

```
#include "neighbourrule.h"
```

## 7.32 neighbourrule.cpp

```

00001 #include "neighbourrule.h"
00002
00084 bool NeighbourRule::inInterval(unsigned int matchingNeighbours) const
00085 {
00086     if (matchingNeighbours >= m_neighbourInterval.first && matchingNeighbours <=
00087         m_neighbourInterval.second)
00088         return true;
00089     else
00090         return false;
00091 }
00092
00095 NeighbourRule::NeighbourRule(unsigned int outputState, QVector<unsigned int>
00096     currentCellValues, QPair<unsigned int, unsigned int> intervalNbrNeighbour, QSet<unsigned int> neighbourValues)
00097 :
00098     Rule(currentCellValues, outputState), m_neighbourInterval(intervalNbrNeighbour),
00099     m_neighbourPossibleValues(neighbourValues)
00100 {
00101     if (m_neighbourInterval.second == 0)
00102         throw QString(QObject::tr("Low value of the number of neighbour interval can't be 0"));
00103     if (m_neighbourInterval.first > m_neighbourInterval.second)
00104         throw QString(QObject::tr("The interval must be (x,y) with x <= y"));
00105 }
00106
00104 NeighbourRule::~NeighbourRule()
00105 {
00106 }
00107
00108
00115 bool NeighbourRule::matchCell(const Cell *c) const
00116 {
00117     unsigned int matchingNeighbours = 0;
00118     if (!m_currentCellPossibleValues.contains(c->
00119         getState()))
00120         return false;
00121 }

```

```

00121 // QSet<unsigned int> set;
00122 //QSet<unsigned int> m_neighbourPossibleValues;
00123 //set<<3<<2<<5<<9;
00124 QSet<unsigned int>::const_iterator i = m_neighbourPossibleValues.constBegin();
00125 if (i == m_neighbourPossibleValues.constEnd()) // All possibles values (except
0)
00126 {
00127     matchingNeighbours = c->countNeighbours();
00128 }
00129 else
00130 {
00131     while (i != m_neighbourPossibleValues.constEnd()) {
00132         //std::cout<<*i;
00133         matchingNeighbours += c->countNeighbours(*i);
00134         ++i;
00135     }
00136 }
00137 if(!inInterval(matchingNeighbours))
00138     return false; //the rule cannot be applied to the cell
00139
00140 return true; //the rule can be applied to the cell
00141
00142 }
00143
00146 QJsonObject NeighbourRule::toJson() const
00147 {
00148     QJsonObject object (Rule::toJson());
00149
00150     object.insert("type", QJsonValue("neighbour"));
00151     object.insert("neighbourNumberMin", QJsonValue((int)m_neighbourInterval.first));
00152     object.insert("neighbourNumberMax", QJsonValue((int)m_neighbourInterval.second));
00153
00154     QJsonArray neighbourState;
00155     for (QSet<unsigned int>::const_iterator it = m_neighbourPossibleValues.begin()
; it != m_neighbourPossibleValues.end(); ++it)
00156     {
00157         neighbourState.append(QJsonValue((int)*it));
00158     }
00159     object.insert("neighbourStates", neighbourState);
00160
00161     return object;
00162 }

```

### 7.33 neighbourrule.h File Reference

```

#include <QPair>
#include <QSet>
#include "cell.h"
#include "rule.h"

```

#### Classes

- class [NeighbourRule](#)

*Contains the rule condition and the output state if that condition is satisfied The rule modifies a cell depending on the number of its neighbours belonging to a range.*

### 7.34 neighbourrule.h

```

00001 #ifndef NEIGHBOURRULE_H
00002 #define NEIGHBOURRULE_H
00003
00004 #include <QPair>
00005 #include <QSet>
00006 #include "cell.h"
00007 #include "rule.h"
00008
00013 class NeighbourRule : public Rule

```

```

00014 {
00015     protected:
00016         QPair<unsigned int , unsigned int> m_neighbourInterval;
00017         //ATTENTION check that first is lower than second
00018         QSet<unsigned int> m_neighbourPossibleValues;
00019         bool inInterval(unsigned int matchingNeighbours) const;
00020         //bool load(const QJsonObject &json);
00021     public:
00022         NeighbourRule(unsigned int outputState, QVector<unsigned int> currentCellValues,
00023             QPair<unsigned int, unsigned int> intervalNbrNeighbour, QSet<unsigned int> neighbourValues = QSet<unsigned int>());
00024         ~NeighbourRule();
00025         bool matchCell(const Cell * c) const;
00026         QJsonObject toJson() const;
00027 };
00028
00029 #endif // NEIGHBOURRULE_H

```

## 7.35 presentation.md File Reference

## 7.36 presentation.md

```

00001 \mainpage Presentation
00002
00003 # Le projet
00004 Dans le cadre de l'UV L021, nous avons réalisé le projet AutoCell. Ce projet vise à nous faire
00005 concevoir un programme informatique à partir d'un cahier des charges. Le projet a été réalisé en C++ avec le
00006 framework Qt. Les membres de l'équipe sont Alexandre Bertolotto, Elias Hariz et Maxime Escourrou, tous les 3 en
00007 GI02.
00008
00009 # Qu'est-ce qu'un automate cellulaire ?
00010 D'après Wikipédia :
00011 "Un automate cellulaire consiste en une grille régulière de « cellules » contenant chacune un « état »
00012 choisi parmi un ensemble fini et qui peut évoluer au cours du temps. L'état d'une cellule au temps t+1 est
00013 fonction de l'état au temps t d'un nombre fini de cellules appelé son « voisinage ». À chaque nouvelle unité
00014 de temps, les mêmes règles sont appliquées simultanément à toutes les cellules de la grille, produisant une
00015 nouvelle « génération » de cellules dépendant entièrement de la génération précédente."
00016
00017 Le concept des automates cellulaires a été découvert dans les années 1940 par Stanislaw Ulam et John
00018 von Neumann au laboratoire de Los Alamos (USA). C'est dans les années 1970 que le concept s'est étendu, avec
00019 le jeu de la vie de la vie du mathématicien britannique John Horton Conway. Le jeu de la vie sera présenté
00020 plus en détail dans la partie suivante.
00021
00022
00023

```

## 7.37 rule.cpp File Reference

```
#include "rule.h"
```

## 7.38 rule.cpp

```

00001 #include "rule.h"
00002
00003 Rule::Rule(QVector<unsigned int> currentCellValues, unsigned int outputState):
00004     m_currentCellPossibleValues(currentCellValues), m_cellOutputState(outputState)
00005 {
00006 }
00007
00008 QJsonObject Rule::toJson() const
00009 {
00010     QJsonObject object;
00011     object.insert("finalState", QJsonValue((int)m_cellOutputState));
00012     QJsonArray currentStates;

```

```

00021     for (int i = 0; i < m_currentCellPossibleValues.size(); i++)
00022     {
00023         currentStates.append(QJsonValue((int)m_currentCellPossibleValues.at(i)))
00024     };
00025     object.insert("currentStates", currentStates);
00026
00027     return object;
00028 }
00029
00032 unsigned int Rule::getCellOutputState() const
00033 {
00034     return m_cellOutputState;
00035 }
00036

```

## 7.39 rule.h File Reference

```

#include <QVector>
#include <QJsonObject>
#include <QJsonArray>
#include "cell.h"

```

### Classes

- class [Rule](#)

## 7.40 rule.h

```

00001 #ifndef RULE_H
00002 #define RULE_H
00003
00004 #include <QVector>
00005 #include <QJsonObject>
00006 #include <QJsonArray>
00007 #include "cell.h"
00008
00009
00013 class Rule
00014 {
00015 protected:
00016     QVector<unsigned int> m_currentCellPossibleValues;
00017     unsigned int m_cellOutputState;
00018 public:
00019     Rule(QVector<unsigned int> currentCellValues, unsigned int outputState);
00020
00021     virtual QJsonObject toJson() const = 0;
00022     virtual ~Rule() {}
00032     virtual bool matchCell(const Cell * c) const = 0;
00033     unsigned int getCellOutputState() const;
00034
00035 };
00036
00037 #endif // RULE_H

```

## 7.41 ruleeditor.cpp File Reference

```

#include "ruleeditor.h"

```

## 7.42 ruleeditor.cpp

```

00001 #include "ruleeditor.h"
00002
00006 RuleEditor::RuleEditor(unsigned int dimensions, QWidget *parent) : QDialog(parent),
    m_dimensions(dimensions)
00007 {
00008     QGridLayout *rulesInputLayout = new QGridLayout();
00009     QHBoxLayout *hlayout = new QHBoxLayout();
00010     if (m_dimensions > 1)
00011     {
00012         m_selectedRule = -1;
00013
00014         m_rulesListWidget = new QListWidget(this);
00015         QLabel *rulesLabel = new QLabel("Rules ",this);
00016         QVBoxLayout *rulesListLayout = new QVBoxLayout();
00017         rulesListLayout->addWidget(rulesLabel);
00018         rulesListLayout->addWidget(m_rulesListWidget);
00019         hlayout->addLayout(rulesListLayout);
00020
00021         rulesInputLayout->addWidget(new QLabel("Current cell values :",this),0,0);
00022         m_currentStatesEdit = new QLineEdit(this);
00023         QRegExp rgx("[0-9]+,)*");
00024         QRegExpValidator *v = new QRegExpValidator(rgx, this);
00025         m_currentStatesEdit->setValidator(v);
00026         rulesInputLayout->addWidget(m_currentStatesEdit,0,1);
00027
00028         rulesInputLayout->addWidget(new QLabel("Neighbour number lower bound :",this),1,0);
00029         m_lowerNeighbourBox = new QSpinBox(this);
00030         rulesInputLayout->addWidget(m_lowerNeighbourBox,1,1);
00031
00032         rulesInputLayout->addWidget(new QLabel("Neighbour number upper bound :",this),2,0);
00033         m_upperNeighbourBox = new QSpinBox(this);
00034         rulesInputLayout->addWidget(m_upperNeighbourBox,2,1);
00035
00036         rulesInputLayout->addWidget(new QLabel("Neighbour values :",this),3,0);
00037         m_neighbourStatesEdit = new QLineEdit(this);
00038         m_neighbourStatesEdit->setValidator(v);
00039         rulesInputLayout->addWidget(m_neighbourStatesEdit,3,1);
00040
00041         rulesInputLayout->addWidget(new QLabel("Output state :",this),4,0);
00042         m_outputStateBox = new QSpinBox(this);
00043         rulesInputLayout->addWidget(m_outputStateBox,4,1);
00044     }
00045     else
00046     {
00047         rulesInputLayout->addWidget(new QLabel("Automaton number :",this),0,0);
00048         m_automatonNumber = new QSpinBox(this);
00049         m_automatonNumber->setMaximum(255);
00050         m_automatonNumber->setMinimum(0);
00051         rulesInputLayout->addWidget(m_automatonNumber,0,1);
00052     }
00053
00054     hlayout->addLayout(rulesInputLayout);
00055     QVBoxLayout* mainLayout = new QVBoxLayout();
00056     QHBoxLayout* buttonLayout = new QHBoxLayout();
00057
00058     if (dimensions > 1)
00059     {
00060         m_addBt = new QPushButton("Add Rule",this);
00061         m_importBt = new QPushButton("Import Rule file",this);
00062         m_removeBt = new QPushButton("Remove Rule",this);
00063         buttonLayout->addWidget(m_importBt);
00064         buttonLayout->addWidget(m_addBt);
00065         buttonLayout->addWidget(m_removeBt);
00066     }
00067     m_doneBt = new QPushButton("Done !",this);
00068
00069
00070     buttonLayout->addWidget(m_doneBt);
00071
00072     mainLayout->addLayout(hlayout);
00073     mainLayout->addLayout(buttonLayout);
00074     setLayout(mainLayout);
00075
00076     if (dimensions > 1)
00077     {
00078         connect(m_addBt, SIGNAL(clicked(bool)), this, SLOT(addRule()));
00079         connect(m_importBt, SIGNAL(clicked(bool)), this, SLOT(
importFile()));
00080         connect(m_removeBt, SIGNAL(clicked(bool)), this, SLOT(
removeRule()));
00081     }
00082     connect(m_doneBt, SIGNAL(clicked(bool)), this, SLOT(sendRules()));
00083
00084 }

```

```

00085
00086
00089 void RuleEditor::addRule(){
00090     unsigned int outputState = m_outputStateBox->value();
00091     QVector<unsigned int> currentCellValues;
00092     QStringList valList = m_currentStatesEdit->text().split(",");
00093     for(int i = 0; i < valList.size(); i++) currentCellValues.append(valList.at(i).toInt());
00094
00095     QPair<unsigned int, unsigned int> neighbourInterval;
00096     neighbourInterval.first = m_lowerNeighbourBox->value();
00097     neighbourInterval.second = m_upperNeighbourBox->value();
00098
00099     QSet<unsigned int> neighbourValues;
00100     valList = m_neighbourStatesEdit->text().split(",");
00101     for(int i = 0; i < valList.size(); i++) neighbourValues << valList.at(i).toInt();
00102
00103     m_rules.append(new NeighbourRule(outputState,currentCellValues,neighbourInterval,
neighbourValues));
00104
00105     QString listLabel = m_currentStatesEdit->text()+" -> "+QString::number(
m_outputStateBox->value())
00106         +" if "+QString::number(m_lowerNeighbourBox->value())+" to "+
00107         QString::number(m_upperNeighbourBox->value())+" neighbours are
in states "+
00108         m_neighbourStatesEdit->text();
00109     m_rulesListWidget->addItem(listLabel);
00110 }
00111
00114 void RuleEditor::removeRule(){
00115     m_rules.removeAt(m_rulesListWidget->currentRow());
00116     delete m_rulesListWidget->takeItem(m_rulesListWidget->currentRow());
00117 }
00118
00121 void RuleEditor::sendRules(){
00122     if (m_dimensions == 1)
00123     {
00124         QList<const Rule*> ruleList = generateIDRules(
m_automatonNumber->value());
00125         for (const Rule* rule : ruleList) // C++11
00126         {
00127             m_rules.append(rule);
00128         }
00129     }
00130     emit rulesFilled(m_rules);
00131     this->close();
00132 }
00133 }
00134
00137 void RuleEditor::importFile(){
00138     QString fileName = QFileDialog::getOpenFileName(this, tr("Open Rule file"), ".",
tr("Automaton rule files (*.atr)"));
00139     if(!fileName.isEmpty()){
00140         emit fileImported(fileName);
00141         this->close();
00142     }
00143 }
00144 }

```

## 7.43 ruleeditor.h File Reference

```

#include <QtWidgets>
#include "neighbourrule.h"
#include "automate.h"

```

### Classes

- class [RuleEditor](#)

*Dialog for editing the rules.*

## 7.44 ruleeditor.h

```

00001 #ifndef RULEEDITOR_H
00002 #define RULEEDITOR_H
00003 #include <QtWidgets>
00004 #include "neighbourrule.h"
00005 #include "automate.h"
00006
00009 class RuleEditor : public QDialog
00010 {
00011     Q_OBJECT
00012     QList<const Rule*> m_rules;
00013     QListWidget* m_rulesListWidget;
00014     QTableWidgetItem* m_rulesTable;
00015
00016     QSpinBox* m_outputStateBox;
00017     QLineEdit* m_currentStatesEdit;
00018     QLineEdit* m_neighbourStatesEdit;
00019     QSpinBox* m_upperNeighbourBox;
00020     QSpinBox* m_lowerNeighbourBox;
00021     QSpinBox* m_automatonNumber;
00022
00023     QPushButton* m_addBt;
00024     QPushButton* m_doneBt;
00025     QPushButton* m_removeBt;
00026     QPushButton* m_importBt;
00027
00028     unsigned int m_selectedRule;
00029     unsigned int m_dimensions;
00030
00031
00032 public:
00033     explicit RuleEditor(unsigned int dimensions, QWidget *parent = nullptr);
00034
00035 signals:
00036     void rulesFilled(QList<const Rule*> rules);
00037     void fileImported(QString path);
00038
00039 public slots:
00040     void removeRule();
00041     void addRule();
00042     void importFile();
00043     void sendRules();
00044
00045
00046 };
00047
00048 #endif // RULEEDITOR_H

```





# Index

- ~Automate
  - Automate, [13](#)
- ~AutomateHandler
  - AutomateHandler, [19](#)
- ~CellHandler
  - CellHandler, [32](#)
- ~MainWindow
  - MainWindow, [51](#)
- ~NeighbourRule
  - NeighbourRule, [69](#)
- ~Rule
  - Rule, [72](#)
- addAutomate
  - AutomateHandler, [19](#)
- addAutomatonRuleFile
  - MainWindow, [51](#)
- addAutomatonRules
  - MainWindow, [52](#)
- addEmptyRow
  - MainWindow, [52](#)
- addNeighbour
  - Cell, [23](#)
- addNeighbourState
  - MatrixRule, [66](#)
- addRule
  - Automate, [13](#)
  - RuleEditor, [75](#)
- addRuleFile
  - Automate, [14](#)
- Automate, [11](#)
  - ~Automate, [13](#)
  - addRule, [13](#)
  - addRuleFile, [14](#)
  - Automate, [12](#), [13](#)
  - AutomateHandler, [16](#)
  - getCellHandler, [14](#)
  - getRules, [14](#)
  - loadRules, [14](#)
  - m\_cellHandler, [17](#)
  - m\_rules, [17](#)
  - run, [15](#)
  - saveAll, [15](#)
  - saveCells, [15](#)
  - saveRules, [15](#)
  - setRulePriority, [16](#)
- automate.cpp, [81](#), [82](#)
  - generate1DRules, [81](#)
  - getRuleFromNumber, [82](#)
- automate.h, [86](#), [88](#)
  - generate1DRules, [87](#)
  - getRuleFromNumber, [87](#)
- AutomateHandler, [17](#)
  - ~AutomateHandler, [19](#)
  - addAutomate, [19](#)
  - Automate, [16](#)
  - AutomateHandler, [18](#)
  - deleteAutomate, [19](#)
  - deleteAutomateHandler, [20](#)
  - getAutomate, [20](#)
  - getAutomateHandler, [20](#)
  - getNumberAutomates, [21](#)
  - m\_ActiveAutomates, [22](#)
  - m\_activeAutomateHandler, [21](#)
  - operator=, [21](#)
- automatehandler.cpp, [88](#), [89](#)
- automatehandler.h, [89](#), [90](#)
- back
  - Cell, [24](#)
- backward
  - MainWindow, [52](#)
- begin
  - CellHandler, [33](#)
- Cell, [22](#)
  - addNeighbour, [23](#)
  - back, [24](#)
  - Cell, [23](#)
  - countNeighbours, [24](#)
  - forceState, [25](#)
  - getNeighbour, [25](#)
  - getNeighbours, [25](#)
  - getRelativePosition, [26](#)
  - getState, [26](#)
  - m\_neighbours, [27](#)
  - m\_nextState, [28](#)
  - m\_states, [28](#)
  - reset, [26](#)
  - setState, [27](#)
  - validState, [27](#)
- cell.cpp, [90](#)
- cell.h, [91](#), [92](#)
- CellHandler, [28](#)
  - ~CellHandler, [32](#)
  - begin, [33](#)
  - CellHandler, [31](#), [32](#)
  - CellHandler::iteratorT, [47](#)
  - const\_iterator, [30](#)
  - end, [33](#)

- foundNeighbours, 33
- generate, 34
- generationTypes, 30
- getCell, 34
- getDimensions, 34
- getListNeighboursPositions, 35
- getListNeighboursPositionsRecursive, 35
- getMaxState, 36
- iterator, 30
- load, 36
- m\_cells, 39
- m\_dimensions, 40
- nextStates, 37
- positionIncrement, 37
- previousStates, 38
- print, 38
- reset, 39
- save, 39
- CellHandler::iteratorT< CellHandler\_T, Cell\_T >, 44
- CellHandler::iteratorT
  - CellHandler, 47
  - changedDimension, 45
  - iteratorT, 45
  - m\_changedDimension, 47
  - m\_finished, 47
  - m\_handler, 47
  - m\_position, 48
  - m\_zero, 48
  - operator!=, 46
  - operator\*, 46
  - operator++, 46
  - operator->, 46
- cellPressed
  - MainWindow, 53
- cellhandler.cpp, 92
- cellhandler.h, 97, 98
- changeCellValue
  - MainWindow, 53
- changedDimension
  - CellHandler::iteratorT, 45
- closeTab
  - MainWindow, 53
- const\_iterator
  - CellHandler, 30
- countNeighbours
  - Cell, 24
- createBoard
  - MainWindow, 54
- createButtons
  - MainWindow, 54
- createGenButtons
  - CreationDialog, 41
- createTab
  - MainWindow, 54
- createTabs
  - MainWindow, 54
- createToolBar
  - MainWindow, 55
- CreationDialog, 40
  - createGenButtons, 41
  - CreationDialog, 41
  - m\_densityBox, 42
  - m\_dimensionsEdit, 42
  - m\_doneBt, 43
  - m\_empGen, 43
  - m\_groupBox, 43
  - m\_randGen, 43
  - m\_stateMaxBox, 43
  - m\_symGen, 44
  - processSettings, 42
  - settingsFilled, 42
- creationdialog.cpp, 99
- creationdialog.h, 100, 101
- deleteAutomate
  - AutomateHandler, 19
- deleteAutomateHandler
  - AutomateHandler, 20
- end
  - CellHandler, 33
- fileImported
  - RuleEditor, 76
- fillInterval
  - matrixrule.cpp, 112
  - matrixrule.h, 114
- forceState
  - Cell, 25
- forward
  - MainWindow, 55
- foundNeighbours
  - CellHandler, 33
- generate
  - CellHandler, 34
- generate1DRules
  - automate.cpp, 81
  - automate.h, 87
- generationTypes
  - CellHandler, 30
- getAutomate
  - AutomateHandler, 20
- getAutomateHandler
  - AutomateHandler, 20
- getBoard
  - MainWindow, 55
- getCell
  - CellHandler, 34
- getCellHandler
  - Automate, 14
- getCellOutputState
  - Rule, 72
- getColor
  - MainWindow, 55
- getDimensions
  - CellHandler, 34

- getListNeighboursPositions
  - CellHandler, 35
- getListNeighboursPositionsRecursive
  - CellHandler, 35
- getMaxState
  - CellHandler, 36
- getNeighbour
  - Cell, 25
- getNeighbours
  - Cell, 25
- getNumberAutomates
  - AutomateHandler, 21
- getRelativePosition
  - Cell, 26
- getRuleFromNumber
  - automate.cpp, 82
  - automate.h, 87
- getRules
  - Automate, 14
- getState
  - Cell, 26
- handlePlayPause
  - MainWindow, 56
- handleTabChanged
  - MainWindow, 56
- hookMoreColor
  - MainWindow, 56
- importFile
  - RuleEditor, 76
- inInterval
  - NeighbourRule, 69
- iterator
  - CellHandler, 30
- iteratorT
  - CellHandler::iteratorT, 45
- load
  - CellHandler, 36
- loadRules
  - Automate, 14
- m\_ActiveAutomates
  - AutomateHandler, 22
- m\_activeAutomateHandler
  - AutomateHandler, 21
- m\_addBt
  - RuleEditor, 77
- m\_automatonNumber
  - RuleEditor, 77
- m\_boardHSize
  - MainWindow, 60
- m\_boardVSize
  - MainWindow, 60
- m\_cellHandler
  - Automate, 17
- m\_cellOutputState
  - Rule, 73
- m\_cellSetter
  - MainWindow, 60
- m\_cellSize
  - MainWindow, 61
- m\_cells
  - CellHandler, 39
- m\_changedDimension
  - CellHandler::iteratorT, 47
- m\_currentCellPossibleValues
  - Rule, 73
- m\_currentCellX
  - MainWindow, 61
- m\_currentCellY
  - MainWindow, 61
- m\_currentStatesEdit
  - RuleEditor, 77
- m\_densityBox
  - CreationDialog, 42
- m\_dimensions
  - CellHandler, 40
  - RuleEditor, 78
- m\_dimensionsEdit
  - CreationDialog, 42
- m\_doneBt
  - CreationDialog, 43
  - RuleEditor, 78
- m\_empGen
  - CreationDialog, 43
- m\_finished
  - CellHandler::iteratorT, 47
- m\_groupBox
  - CreationDialog, 43
- m\_handler
  - CellHandler::iteratorT, 47
- m\_importBt
  - RuleEditor, 78
- m\_lowerNeighbourBox
  - RuleEditor, 78
- m\_matrix
  - MatrixRule, 67
- m\_neighbourInterval
  - NeighbourRule, 70
- m\_neighbourPossibleValues
  - NeighbourRule, 70
- m\_neighbourStatesEdit
  - RuleEditor, 78
- m\_neighbours
  - Cell, 27
- m\_newAutomatonBt
  - MainWindow, 61
- m\_nextState
  - Cell, 28
- m\_nextStateBt
  - MainWindow, 61
- m\_openAutomatonBt
  - MainWindow, 62
- m\_outputStateBox
  - RuleEditor, 79

- m\_pauselcon
  - MainWindow, 62
- m\_playIcon
  - MainWindow, 62
- m\_playPauseBt
  - MainWindow, 62
- m\_position
  - CellHandler::iteratorT, 48
- m\_previousStateBt
  - MainWindow, 62
- m\_randGen
  - CreationDialog, 43
- m\_removeBt
  - RuleEditor, 79
- m\_resetBt
  - MainWindow, 63
- m\_rules
  - Automate, 17
  - RuleEditor, 79
- m\_rulesListWidget
  - RuleEditor, 79
- m\_rulesTable
  - RuleEditor, 79
- m\_running
  - MainWindow, 63
- m\_saveAutomatonBt
  - MainWindow, 63
- m\_selectedRule
  - RuleEditor, 80
- m\_stateMaxBox
  - CreationDialog, 43
- m\_states
  - Cell, 28
- m\_symGen
  - CreationDialog, 44
- m\_tabs
  - MainWindow, 63
- m\_timeStep
  - MainWindow, 64
- m\_timer
  - MainWindow, 63
- m\_toolBar
  - MainWindow, 64
- m\_upperNeighbourBox
  - RuleEditor, 80
- m\_zero
  - CellHandler::iteratorT, 48
- m\_zoom
  - MainWindow, 64
- main
  - main.cpp, 101
- main.cpp, 101, 102
  - main, 101
- MainWindow, 48
  - ~MainWindow, 51
  - addAutomatonRuleFile, 51
  - addAutomatonRules, 52
  - addEmptyRow, 52
  - backward, 52
  - cellPressed, 53
  - changeCellValue, 53
  - closeTab, 53
  - createBoard, 54
  - createButtons, 54
  - createTab, 54
  - createTabs, 54
  - createToolBar, 55
  - forward, 55
  - getBoard, 55
  - getColor, 55
  - handlePlayPause, 56
  - handleTabChanged, 56
  - hookMoreColor, 56
  - m\_boardHSize, 60
  - m\_boardVSize, 60
  - m\_cellSetter, 60
  - m\_cellSize, 61
  - m\_currentCellX, 61
  - m\_currentCellY, 61
  - m\_newAutomatonBt, 61
  - m\_nextStateBt, 61
  - m\_openAutomatonBt, 62
  - m\_pauselcon, 62
  - m\_playIcon, 62
  - m\_playPauseBt, 62
  - m\_previousStateBt, 62
  - m\_resetBt, 63
  - m\_running, 63
  - m\_saveAutomatonBt, 63
  - m\_tabs, 63
  - m\_timeStep, 64
  - m\_timer, 63
  - m\_toolBar, 64
  - m\_zoom, 64
  - MainWindow, 51
  - nextState, 57
  - openCreationWindow, 57
  - openFile, 57
  - receiveCellHandler, 57
  - reset, 58
  - runAutomaton, 58
  - saveToFile, 58
  - setSize, 59
  - updateBoard, 60
- mainwindow.cpp, 102
- mainwindow.h, 110
- matchCell
  - MatrixRule, 66
  - NeighbourRule, 69
  - Rule, 72
- MatrixRule, 65
  - addNeighbourState, 66
  - m\_matrix, 67
  - matchCell, 66
  - MatrixRule, 65
  - toJson, 67

- matrixrule.cpp, 112
  - fillInterval, 112
- matrixrule.h, 114
  - fillInterval, 114
- NeighbourRule, 68
  - ~NeighbourRule, 69
  - inInterval, 69
  - m\_neighbourInterval, 70
  - m\_neighbourPossibleValues, 70
  - matchCell, 69
  - NeighbourRule, 68
  - toJson, 70
- neighbourrule.cpp, 115
- neighbourrule.h, 116
- nextState
  - MainWindow, 57
- nextStates
  - CellHandler, 37
- openCreationWindow
  - MainWindow, 57
- openFile
  - MainWindow, 57
- operator!=
  - CellHandler::iteratorT, 46
- operator\*
  - CellHandler::iteratorT, 46
- operator++
  - CellHandler::iteratorT, 46
- operator->
  - CellHandler::iteratorT, 46
- operator=
  - AutomateHandler, 21
- positionIncrement
  - CellHandler, 37
- presentation.md, 117
- previousStates
  - CellHandler, 38
- print
  - CellHandler, 38
- processSettings
  - CreationDialog, 42
- receiveCellHandler
  - MainWindow, 57
- removeRule
  - RuleEditor, 76
- reset
  - Cell, 26
  - CellHandler, 39
  - MainWindow, 58
- Rule, 71
  - ~Rule, 72
  - getCellOutputState, 72
  - m\_cellOutputState, 73
  - m\_currentCellPossibleValues, 73
  - matchCell, 72
  - Rule, 72
  - toJson, 73
- rule.cpp, 117
- rule.h, 118
- RuleEditor, 74
  - addRule, 75
  - fileImported, 76
  - importFile, 76
  - m\_addBt, 77
  - m\_automatonNumber, 77
  - m\_currentStatesEdit, 77
  - m\_dimensions, 78
  - m\_doneBt, 78
  - m\_importBt, 78
  - m\_lowerNeighbourBox, 78
  - m\_neighbourStatesEdit, 78
  - m\_outputStateBox, 79
  - m\_removeBt, 79
  - m\_rules, 79
  - m\_rulesListWidget, 79
  - m\_rulesTable, 79
  - m\_selectedRule, 80
  - m\_upperNeighbourBox, 80
  - removeRule, 76
  - RuleEditor, 75
  - rulesFilled, 76
  - sendRules, 77
- ruleeditor.cpp, 118, 119
- ruleeditor.h, 120, 121
- rulesFilled
  - RuleEditor, 76
- run
  - Automate, 15
- runAutomaton
  - MainWindow, 58
- save
  - CellHandler, 39
- saveAll
  - Automate, 15
- saveCells
  - Automate, 15
- saveRules
  - Automate, 15
- saveToFile
  - MainWindow, 58
- sendRules
  - RuleEditor, 77
- setRulePriority
  - Automate, 16
- setSize
  - MainWindow, 59
- setState
  - Cell, 27
- settingsFilled
  - CreationDialog, 42
- toJson
  - MatrixRule, 67

NeighbourRule, [70](#)  
Rule, [73](#)

updateBoard  
  MainWindow, [60](#)

validState  
  Cell, [27](#)