# AutoCell

Generated by Doxygen 1.8.14

# Contents

1 Main Page 1

2 Presentation 3

3 Hierarchical Index 5

3.1 Class Hierarchy . . . . . 5

4 Class Index 7

4.1 Class List . . . . . 7

5 File Index 9

5.1 File List . . . . . 9

6 Class Documentation 11

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 getCellHandler() . . . . . 14

6.1.3.3 getRules() . . . . . 14

6.1.3.4 loadRules() . . . . . 14

6.1.3.5 run() . . . 15

6.1.3.6 saveAll() . . . 15

6.1.3.7 saveCells() . . . 15

6.1.3.8 saveRules() . . . 16

6.1.3.9 setRulePriority() . . . 16

6.1.4 Friends And Related Function Documentation . . . 16

6.1.4.1 AutomateHandler . . . 16

6.1.5 Member Data Documentation . . . 17

6.1.5.1 m_cellHandler . . . 17

6.1.5.2 m_rules . . . 17

6.2 AutomateHandler Class Reference . . . 17

6.2.1 Detailed Description . . . 18

6.2.2 Constructor & Destructor Documentation . . . 18

6.2.2.1 AutomateHandler() . . . 18

6.2.2.2 ∼AutomateHandler() . . . 18

6.2.3 Member Function Documentation . . . 18

6.2.3.1 deleteActiveAutomate() . . . 18

6.2.3.2 getActiveAutomate() . . . 19

6.2.3.3 operator=() . . . 19

6.2.3.4 setActiveAutomate() . . . 19

6.2.4 Member Data Documentation . . . 19

6.2.4.1 m_activeAutomate . . . 19

6.3 Cell Class Reference . . . 20

6.3.1 Detailed Description . . . 20

6.3.2 Constructor & Destructor Documentation . . . 21

6.3.2.1 Cell() . . . 21

6.3.3 Member Function Documentation . . . 21

6.3.3.1 addNeighbour() . . . 21

6.3.3.2 countNeighbours() [1/2] . . . 22

6.3.3.3 countNeighbours() [2/2] . . . 22

6.3.3.4 forceState() . . . 22

6.3.3.5 getNeighbour() . . . 22

6.3.3.6 getNeighbours() . . . 23

6.3.3.7 getRelativePosition() . . . 23

6.3.3.8 getState() . . . 23

6.3.3.9 setState() . . . 24

6.3.3.10 validState() . . . 24

6.3.4 Member Data Documentation . . . 24

6.3.4.1 m_neighbours . . . 24

6.3.4.2 m_nextState . . . 25

6.3.4.3 m_state . . . 25

6.4 CellHandler Class Reference . . . 25

6.4.1 Detailed Description . . . 27

6.4.2 Member Typedef Documentation . . . 27

6.4.2.1 const_iterator . . . 27

6.4.2.2 iterator . . . 27

6.4.3 Member Enumeration Documentation . . . 27

6.4.3.1 generationTypes . . . 27

6.4.4 Constructor & Destructor Documentation . . . 28

6.4.4.1 CellHandler() [1/3] . . . 28

6.4.4.2 CellHandler() [2/3] . . . 28

6.4.4.3 CellHandler() [3/3] . . . 29

6.4.4.4 ∼CellHandler() . . . 29

6.4.5 Member Function Documentation . . . 30

6.4.5.1 begin() [1/2] . . . 30

6.4.5.2 begin() [2/2] . . . 30

6.4.5.3 end() . . . 30

6.4.5.4 foundNeighbours() . . . 30

6.4.5.5 generate() . . . 30

6.4.5.6 getCell() . . . 31

6.4.5.7 getDimensions() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
6.4.5.8 getListNeighboursPositions() . . . . . . . . . . . . . . . . . . . . . . . . 31
6.4.5.9 getListNeighboursPositionsRecursive() . . . . . . . . . . . . . . . . . . . . 32
6.4.5.10 load() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
6.4.5.11 nextStates() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
6.4.5.12 positionIncrement() . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
6.4.5.13 print() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
6.4.5.14 save() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
6.4.6 Member Data Documentation . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
6.4.6.1 m_cells . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
6.4.6.2 m_dimensions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
6.5 CreationDialog Class Reference . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
6.5.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
6.5.2 Constructor & Destructor Documentation . . . . . . . . . . . . . . . . . . . . . 37
6.5.2.1 CreationDialog() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
6.5.3 Member Function Documentation . . . . . . . . . . . . . . . . . . . . . . . . . 37
6.5.3.1 createGenButtons() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
6.5.3.2 processSettings . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
6.5.3.3 settingsFilled . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
6.5.4 Member Data Documentation . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
6.5.4.1 m_densityBox . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
6.5.4.2 m_dimensionsEdit . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
6.5.4.3 m_doneBt . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39
6.5.4.4 m_empGen . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39
6.5.4.5 m_groupBox . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39
6.5.4.6 m_randGen . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39
6.5.4.7 m_stateMaxBox . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39
6.5.4.8 m_symGen . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
6.6 CellHandler::iteratorT< CellHandler_T, Cell_T > Class Template Reference . . . . . . . . . . . 40
6.6.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41

6.6.2 Constructor & Destructor Documentation . . . . . 41
6.6.2.1 iteratorT() . . . . . 41
6.6.3 Member Function Documentation . . . . . 41
6.6.3.1 changedDimension() . . . . . 42
6.6.3.2 operator"!=() . . . . . 42
6.6.3.3 operator∗() . . . . . 42
6.6.3.4 operator++() . . . . . 42
6.6.3.5 operator->() . . . . . 43
6.6.4 Friends And Related Function Documentation . . . . . 43
6.6.4.1 CellHandler . . . . . 43
6.6.5 Member Data Documentation . . . . . 43
6.6.5.1 m_changedDimension . . . . . 43
6.6.5.2 m_finished . . . . . 43
6.6.5.3 m_handler . . . . . 44
6.6.5.4 m_position . . . . . 44
6.6.5.5 m_zero . . . . . 44
6.7 MainWindow Class Reference . . . . . 44
6.7.1 Detailed Description . . . . . 46
6.7.2 Constructor & Destructor Documentation . . . . . 46
6.7.2.1 MainWindow() . . . . . 46
6.7.3 Member Function Documentation . . . . . 47
6.7.3.1 closeTab . . . . . 47
6.7.3.2 createActions() . . . . . 47
6.7.3.3 createBoard() . . . . . 47
6.7.3.4 createIcons() . . . . . 47
6.7.3.5 createTab() . . . . . 48
6.7.3.6 createTabs() . . . . . 48
6.7.3.7 createToolBar() . . . . . 48
6.7.3.8 forward . . . . . 48
6.7.3.9 getBoard() . . . . . 49

6.7.3.10 nextState() . . . . . 49
6.7.3.11 openCreationWindow . . . . . 49
6.7.3.12 openFile . . . . . 49
6.7.3.13 saveToFile . . . . . 50
6.7.3.14 setCellHandler . . . . . 50
6.7.3.15 updateBoard() . . . . . 50
6.7.4 Member Data Documentation . . . . . 50
6.7.4.1 m_boardHSize . . . . . 51
6.7.4.2 m_boardVSize . . . . . 51
6.7.4.3 m_cellHandlers . . . . . 51
6.7.4.4 m_cellSize . . . . . 51
6.7.4.5 m_fastBackward . . . . . 51
6.7.4.6 m_fastBackwardBt . . . . . 52
6.7.4.7 m_fastBackwardIcon . . . . . 52
6.7.4.8 m_fastForward . . . . . 52
6.7.4.9 m_fastForwardBt . . . . . 52
6.7.4.10 m_fastForwardIcon . . . . . 52
6.7.4.11 m_jumpSpeed . . . . . 53
6.7.4.12 m_newAutomaton . . . . . 53
6.7.4.13 m_newAutomatonBt . . . . . 53
6.7.4.14 m_newIcon . . . . . 53
6.7.4.15 m_nextState . . . . . 53
6.7.4.16 m_nextStateBt . . . . . 54
6.7.4.17 m_openAutomaton . . . . . 54
6.7.4.18 m_openAutomatonBt . . . . . 54
6.7.4.19 m_openIcon . . . . . 54
6.7.4.20 m_pauseIcon . . . . . 54
6.7.4.21 m_playIcon . . . . . 55
6.7.4.22 m_playPause . . . . . 55
6.7.4.23 m_playPauseBt . . . . . 55

6.7.4.24 m_previousState . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55
6.7.4.25 m_previousStateBt . . . . . . . . . . . . . . . . . . . . . . . . . . . 55
6.7.4.26 m_resetAutomaton . . . . . . . . . . . . . . . . . . . . . . . . . . . 56
6.7.4.27 m_resetBt . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56
6.7.4.28 m_resetIcon . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56
6.7.4.29 m_saveAutomaton . . . . . . . . . . . . . . . . . . . . . . . . . . . 56
6.7.4.30 m_saveAutomatonBt . . . . . . . . . . . . . . . . . . . . . . . . . . 56
6.7.4.31 m_saveIcon . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57
6.7.4.32 m_speedLabel . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57
6.7.4.33 m_tabs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57
6.7.4.34 m_toolBar . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57
6.8 MatrixRule Class Reference . . . . . . . . . . . . . . . . . . . . . . . . . . . 58
6.8.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . . . . . . . . 58
6.8.2 Constructor & Destructor Documentation . . . . . . . . . . . . . . . . . . 58
6.8.2.1 MatrixRule() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 58
6.8.3 Member Function Documentation . . . . . . . . . . . . . . . . . . . . . . 59
6.8.3.1 addNeighbourState() [1/2] . . . . . . . . . . . . . . . . . . . . . . . 59
6.8.3.2 addNeighbourState() [2/2] . . . . . . . . . . . . . . . . . . . . . . . 59
6.8.3.3 matchCell() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 59
6.8.3.4 toJson() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 60
6.8.4 Member Data Documentation . . . . . . . . . . . . . . . . . . . . . . . . 60
6.8.4.1 m_matrix . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 60
6.9 NeighbourRule Class Reference . . . . . . . . . . . . . . . . . . . . . . . . . 61
6.9.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61
6.9.2 Constructor & Destructor Documentation . . . . . . . . . . . . . . . . . . 62
6.9.2.1 NeighbourRule() . . . . . . . . . . . . . . . . . . . . . . . . . . . . 62
6.9.2.2 ∼NeighbourRule() . . . . . . . . . . . . . . . . . . . . . . . . . . . 62
6.9.3 Member Function Documentation . . . . . . . . . . . . . . . . . . . . . . 62
6.9.3.1 inInterval() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 62
6.9.3.2 matchCell() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 64

6.9.3.3 toJson() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 64
6.9.4 Member Data Documentation . . . . . . . . . . . . . . . . . . . . . . . . . . 64
6.9.4.1 m_neighbourInterval . . . . . . . . . . . . . . . . . . . . . . . . . . . . 64
6.9.4.2 m_neighbourPossibleValues . . . . . . . . . . . . . . . . . . . . . . . . 65
6.10 Rule Class Reference . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65
6.10.1 Detailed Description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65
6.10.2 Constructor & Destructor Documentation . . . . . . . . . . . . . . . . . . . . 66
6.10.2.1 Rule() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 66
6.10.3 Member Function Documentation . . . . . . . . . . . . . . . . . . . . . . . . 66
6.10.3.1 getCellOutputState() . . . . . . . . . . . . . . . . . . . . . . . . . . . 66
6.10.3.2 matchCell() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 66
6.10.3.3 toJson() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 67
6.10.4 Member Data Documentation . . . . . . . . . . . . . . . . . . . . . . . . . . 67
6.10.4.1 m_cellOutputState . . . . . . . . . . . . . . . . . . . . . . . . . . . . 67
6.10.4.2 m_currentCellPossibleValues . . . . . . . . . . . . . . . . . . . . . . . 67

**7 File Documentation 69**

7.1 automate.cpp File Reference . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
7.2 automate.cpp . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
7.3 automate.h File Reference . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 72
7.4 automate.h . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 72
7.5 automatehandler.cpp File Reference . . . . . . . . . . . . . . . . . . . . . . . . . 73
7.5.1 Variable Documentation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 73
7.5.1.1 m_activeAutomate . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 73
7.6 automatehandler.cpp . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 74
7.7 automatehandler.h File Reference . . . . . . . . . . . . . . . . . . . . . . . . . . 74
7.8 automatehandler.h . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 74
7.9 cell.cpp File Reference . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 75
7.10 cell.cpp . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 75
7.11 cell.h File Reference . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76
7.12 cell.h . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76

7.13 cellhandler.cpp File Reference . . . . . 76
7.14 cellhandler.cpp . . . . . 77
7.15 cellhandler.h File Reference . . . . . 82
7.16 cellhandler.h . . . . . 82
7.17 creationdialog.cpp File Reference . . . . . 83
7.18 creationdialog.cpp . . . . . 83
7.19 creationdialog.h File Reference . . . . . 85
7.20 creationdialog.h . . . . . 85
7.21 main.cpp File Reference . . . . . 85
7.21.1 Function Documentation . . . . . 86
7.21.1.1 main() . . . . . 86
7.22 main.cpp . . . . . 86
7.23 mainwindow.cpp File Reference . . . . . 86
7.24 mainwindow.cpp . . . . . 86
7.25 mainwindow.h File Reference . . . . . 90
7.26 mainwindow.h . . . . . 90
7.27 matrixrule.cpp File Reference . . . . . 91
7.27.1 Function Documentation . . . . . 91
7.27.1.1 fillInterval() . . . . . 92
7.28 matrixrule.cpp . . . . . 92
7.29 matrixrule.h File Reference . . . . . 93
7.29.1 Function Documentation . . . . . 93
7.29.1.1 fillInterval() . . . . . 94
7.30 matrixrule.h . . . . . 94
7.31 neighbourrule.cpp File Reference . . . . . 94
7.32 neighbourrule.cpp . . . . . 95
7.33 neighbourrule.h File Reference . . . . . 96
7.34 neighbourrule.h . . . . . 96
7.35 presentation.md File Reference . . . . . 96
7.36 presentation.md . . . . . 96
7.37 README.md File Reference . . . . . 96
7.38 README.md . . . . . 96
7.39 rule.cpp File Reference . . . . . 97
7.40 rule.cpp . . . . . 97
7.41 rule.h File Reference . . . . . 97
7.42 rule.h . . . . . 97

**Index** **99**

# Chapter 1

# Main Page

To generate the Documentation, go in Documentation directory and run `make`.

It will generate html doc (in `output/html/index.html`) and latex doc (pdf output directely in Documentation directory (`docPdf.pdf`).

# Chapter 2

# Presentation

## What is AutoCell

The purpose of this project is to create a Cellular Automate Simulator.

# Chapter 3

# Hierarchical Index

## 3.1 Class Hierarchy

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

Automate . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
AutomateHandler . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
Cell . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
CellHandler . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
CellHandler::iteratorT< CellHandler_T, Cell_T > . . . . . . . . . . . . . . . . . . . . . . . . . 40
QDialog
    CreationDialog . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
QMainWindow
    MainWindow . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
Rule . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65
    MatrixRule . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 58
    NeighbourRule . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61

# Chapter 4

# Class Index

## 4.1 Class List

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

Automate . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
AutomateHandler
Implementation of singleton design pattern . . . . . . . . . . . . . . . . . . . . . . . . . . 17
Cell
Contains the state, the next state and the neighbours . . . . . . . . . . . . . . . . . . . . . . 20
CellHandler
Cell container and cell generator . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
CreationDialog
Automaton creation dialog box . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
CellHandler::iteratorT< CellHandler_T, Cell_T >
Implementation of iterator design pattern with a template to generate iterator and const_iterator at the same time . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
MainWindow
Simulation window . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
MatrixRule
Manage specific rules, about specific values of specific neighbour . . . . . . . . . . . . . . . 58
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 . . . . . . . . . . . . . . 61
Rule . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65

# Chapter 5

# File Index

## 5.1 File List

Here is a list of all files with brief descriptions:

automate.cpp . . . . . 69
automate.h . . . . . 72
automatehandler.cpp . . . . . 73
automatehandler.h . . . . . 74
cell.cpp . . . . . 75
cell.h . . . . . 76
cellhandler.cpp . . . . . 76
cellhandler.h . . . . . 82
creationdialog.cpp . . . . . 83
creationdialog.h . . . . . 85
main.cpp . . . . . 85
mainwindow.cpp . . . . . 86
mainwindow.h . . . . . 90
matrixrule.cpp . . . . . 91
matrixrule.h . . . . . 93
neighbourrule.cpp . . . . . 94
neighbourrule.h . . . . . 96
rule.cpp . . . . . 97
rule.h . . . . . 97

# Chapter 6

# Class Documentation

## 6.1 Automate Class Reference

```
#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 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

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**

| | |
|---|---|
| *cellHandlerFilename* | 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**

| | |
|---|---|
| *dimensions* | Dimensions of the CellHandler |
| *type* | Generation type, empty by default |
| *stateMax* | Generate states between 0 and stateMax |
| *density* | 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**

| | |
|---|---|
| *cellHandlerFilename* | File of the cellHandler |
| *ruleFilename* | 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 229 of file automate.cpp.

References m_rules.

#### 6.1.3.2 getCellHandler()

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

Accessor of m_cellHandler.

Definition at line 281 of file automate.cpp.

References m_cellHandler.

#### 6.1.3.3 getRules()

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

Return all the rules.

Definition at line 247 of file automate.cpp.

References m_rules.

#### 6.1.3.4 loadRules()

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

Load the rules of the json given.

**Returns**

Return false if something went wrong

**Parameters**

| | |
|---|---|
| *json* | 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 Automate().

#### 6.1.3.5 run()

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

Apply the rule on the cells grid nbSteps times.

**Parameters**

| | |
|---|---|
| *nbSteps* | number of iterations of the automate on the cell grid |

Definition at line 256 of file automate.cpp.

References CellHandler::begin(), CellHandler::end(), m_cellHandler, m_rules, and CellHandler::nextStates().

#### 6.1.3.6 saveAll()

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

Save both rules and cellHandler in the differents files.

Definition at line 222 of file automate.cpp.

References saveCells(), and saveRules().

#### 6.1.3.7 saveCells()

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

Save cellHandler.

Definition at line 213 of file automate.cpp.

References m_cellHandler, and CellHandler::save().

Referenced by saveAll().

#### 6.1.3.8 saveRules()

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

Save automate's rules in the file.

**Returns**

False if something went wrong

Definition at line 191 of file automate.cpp.

References m_rules.

Referenced by saveAll().

#### 6.1.3.9 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**

| | |
|---|---|
| *rule* | Rule to move |
| *newPlace* | New place of the rule |

Definition at line 240 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.

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

### Public Member Functions

- void setActiveAutomate (unsigned int activeAutomate)

  *Set the active automate.*

### Static Public Member Functions

- static Automate & getActiveAutomate ()

  *Get the unique running automate instance or create one if there is no instance running.*
- static void deleteActiveAutomate ()

  *Delete the unique running automate instance if it exists.*

## Private Member Functions

- AutomateHandler (const AutomateHandler &a)=delete
- AutomateHandler & operator= (const AutomateHandler &a)=delete
- ∼AutomateHandler ()

## Static Private Attributes

- static AutomateHandler ∗ m_activeAutomate

  *active automate if existing, nullptr else*

### 6.2.1 Detailed Description

Implementation of singleton design pattern.

Definition at line 10 of file automatehandler.h.

### 6.2.2 Constructor & Destructor Documentation

#### 6.2.2.1 AutomateHandler()

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

#### 6.2.2.2 ∼AutomateHandler()

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

Definition at line 7 of file automatehandler.cpp.

### 6.2.3 Member Function Documentation

#### 6.2.3.1 deleteActiveAutomate()

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

Delete the unique running automate instance if it exists.

Definition at line 26 of file automatehandler.cpp.

#### 6.2.3.2 getActiveAutomate()

```
Automate & AutomateHandler::getActiveAutomate ( )  [static]
```

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

**Returns**

the unique running automate instance

Definition at line 16 of file automatehandler.cpp.

#### 6.2.3.3 operator=()

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

#### 6.2.3.4 setActiveAutomate()

```
void AutomateHandler::setActiveAutomate (
             unsigned int activeAutomate )
```

Set the active automate.

Definition at line 35 of file automatehandler.cpp.

### 6.2.4 Member Data Documentation

#### 6.2.4.1 m_activeAutomate

```
AutomateHandler* AutomateHandler::m_activeAutomate  [static], [private]
```

active automate if existing, nullptr else

Definition at line 13 of file automatehandler.h.

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 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

- unsigned int m_state

  *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 10 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**

| | |
|---|---|
| *state* | Cell state, dead state by default |

Definition at line 7 of file cell.cpp.

## 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**

| | |
|---|---|
| *relativePosition* | Relative position of the new neighbour |
| *neighbour* | New neighbour |

**Returns**

False if the neighbour already exists

Definition at line 60 of file cell.cpp.

References m_neighbours.

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

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

Return the number of neighbour which have the given state.

Definition at line 87 of file cell.cpp.

References m_neighbours.

Referenced by NeighbourRule::matchCell().

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

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

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

Definition at line 100 of file cell.cpp.

References m_neighbours.

#### 6.3.3.4 forceState()

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

Force the state change.

Is equivalent to setState followed by validState

**Parameters**

| | |
|---|---|
| *state* | New state |

Definition at line 41 of file cell.cpp.

References m_nextState, and m_state.

#### 6.3.3.5 getNeighbour()

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

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

Definition at line 80 of file cell.cpp.

References m_neighbours.

Referenced by MatrixRule::matchCell().

#### 6.3.3.6 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 73 of file cell.cpp.

References m_neighbours.

#### 6.3.3.7 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**

| | |
|---|---|
| *QString* | Different size of position vectors |

**Parameters**

| | |
|---|---|
| *cellPosition* | Cell Position |
| *neighbourPosition* | Neighbour absolute position |

Definition at line 117 of file cell.cpp.

Referenced by CellHandler::foundNeighbours().

#### 6.3.3.8 getState()

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

Access current cell state.

Definition at line 48 of file cell.cpp.

References m_state.

Referenced by MatrixRule::matchCell(), and NeighbourRule::matchCell().

#### 6.3.3.9 setState()

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

Set temporary state.

To change current cell state, use setState(unsigned int state) then validState().

**Parameters**

| | |
|---|---|
| *state* | New state |

Definition at line 20 of file cell.cpp.

References m_nextState.

#### 6.3.3.10 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_state.

### 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 33 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 31 of file cell.h.

Referenced by forceState(), setState(), and validState().

#### 6.3.4.3 m_state

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

Current state.

Definition at line 30 of file cell.h.

Referenced by forceState(), getState(), 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 save (QString filename) const

  *Save the CellHandler current configuration in the file given.*
- void generate (generationTypes type, unsigned int stateMax=1, unsigned short density=50)

  *Replace Cell values by random values (symetric or not)*
- 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.*

### Private Member Functions

- bool load (const QJsonObject &json)

  *Load the config file in the CellHandler.*
- void foundNeighbours ()

  *Set the neighbours of each cells.*
- void positionIncrement (QVector< unsigned int > &pos, unsigned int value=1) const

  *Increment the QVector given by the value choosen.*
- 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.*

### Private 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 64 of file cellhandler.h.

### 6.4.2.2 iterator

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

Definition at line 65 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 69 of file cellhandler.h.

## 6.4.4 Constructor & 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,
         1,2,0,0,0,0,1,2,3,2]
}
```

**Parameters**

| | |
|---|---|
| *filename* | Json file which contains the description of all the cells |

**Exceptions**

| | |
|---|---|
| *QString* | Unreadable file |
| *QString* | Empty file |
| *QString* | 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**

|  |  |
|---|---|
| *json* | Json object which contains the description of all the cells |

**Exceptions**

|  |  |
|---|---|
| *QString* | 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**

|  |  |
|---|---|
| *dimensions* | Dimensions of the CellHandler |
| *type* | Generation type, empty by default |
| *stateMax* | Generate states between 0 and stateMax |
| *density* | 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 295 of file cellhandler.cpp.

Referenced by print(), Automate::run(), and save().

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

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

Give the iterator which corresponds to the current CellHandler.

Definition at line 288 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 304 of file cellhandler.cpp.

Referenced by print(), Automate::run(), save(), and MainWindow::updateBoard().

### 6.4.5.4 foundNeighbours()

```
void CellHandler::foundNeighbours ( )  [private]
```

Set the neighbours of each cells.

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

Definition at line 397 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 )
```

Replace Cell values by random values (symetric or not)

**Parameters**

| | |
|---|---|
| *type* | Type of random generation |
| *stateMax* | Generate states between 0 and stateMax |
| *density* | Average (%) of non-zeros |

Definition at line 209 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.

### 6.4.5.7 getDimensions()

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

Accessor of m_dimensions.

Definition at line 147 of file cellhandler.cpp.

References m_dimensions.

Referenced by Automate::loadRules(), and MainWindow::updateBoard().

### 6.4.5.8 getListNeighboursPositions()

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

Prepare the call of the recursive version of itself.

**Parameters**

| | |
|---|---|
| *position* | Position of the central cell (x1,x2,x3,..,xn) |

**Returns**

List of positions

Definition at line 456 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  [private]
```

Recursive function which browse the position possibilities tree.

Careful, the complexity is in O(3^dimension)
Piece of the tree:

```
            x_d -1
          /
x_(d-1)-1/_ x_d
         \
          \
            x_d +1

            x_d -1
          /
x_(d-1)  /_ x_d
         \
          \
            x_d +1

            x_d -1
          /
x_(d-1)+1/ x_d
         \
          \
            x_d +1
```

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

**Parameters**

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

**Returns**

List of position

Definition at line 497 of file cellhandler.cpp.

References m_dimensions.

Referenced by getListNeighboursPositions().

#### 6.4.5.10 load()

```
bool CellHandler::load (
              const QJsonObject & json )  [private]
```

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**

| | |
|---|---|
| *json* | Json Object which contains the grid configuration |

**Returns**

False if the Json Object is not correct

Definition at line 339 of file cellhandler.cpp.

References m_cells, m_dimensions, and positionIncrement().

Referenced by CellHandler().

#### 6.4.5.11 nextStates()

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

Valid the state of all cells.

Definition at line 154 of file cellhandler.cpp.

References m_cells.

Referenced by Automate::run().

#### 6.4.5.12 positionIncrement()

```
void CellHandler::positionIncrement (
            QVector< unsigned int > & pos,
            unsigned int value = 1 ) const  [private]
```

Increment the QVector given by the value choosen.

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

**Parameters**

| | |
|---|---|
| *pos* | Position to increment |
| *value* | Value to add, 1 by default |

Definition at line 427 of file cellhandler.cpp.

References m_dimensions.

Referenced by CellHandler(), foundNeighbours(), generate(), and load().

#### 6.4.5.13 print()

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

Print in the given stream the CellHandler.

**Parameters**

| | |
|---|---|
| *stream* | Stream to print into |

Definition at line 274 of file cellhandler.cpp.

References begin(), and end().

#### 6.4.5.14 save()

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

Save the CellHandler current configuration in the file given.

**Parameters**

| | |
|---|---|
| *filename* | Path to the file |

**Returns**

False if there was a problem

**Exceptions**

| | |
|---|---|
| *QString* | Impossible to open the file |

Definition at line 169 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  [private]
```

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

Definition at line 101 of file cellhandler.h.

Referenced by CellHandler(), foundNeighbours(), generate(), getCell(), load(), nextStates(), and ~CellHandler().

#### 6.4.6.2 m_dimensions

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

Vector of x dimensions.

Definition at line 100 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:

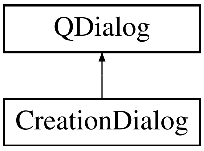
- cellhandler.h
- cellhandler.cpp

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

### Public Member Functions

- CreationDialog (QWidget *parent=0)

### 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 )
```

Definition at line 5 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 51 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 72 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

  *Return the number of dimensions we change.*

### 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**

| | |
|---|---|
| *handler* | CellHandler to browse |

Definition at line 537 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
```

Return the number of dimensions we change.

For example, if we were at the (3,4,4) cell, and we incremented the position, we are now at (4,0,0), and changed↩ Dimension return 2 (because of the 2 zeros).

Definition at line 599 of file cellhandler.cpp.

#### 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 51 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
```

Get the current cell.

Definition at line 588 of file cellhandler.cpp.

#### 6.6.3.4 operator++()

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

Increment the current position and handle dimension changes.

Definition at line 551 of file cellhandler.cpp.

#### 6.6.3.5 operator->()

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

Get the current cell.

Definition at line 579 of file cellhandler.cpp.

### 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 61 of file cellhandler.h.

#### 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 59 of file cellhandler.h.

Referenced by CellHandler::iteratorT< CellHandler_T, Cell_T >::operator!=().

#### 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 57 of file cellhandler.h.

#### 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 58 of file cellhandler.h.

Referenced by CellHandler::iteratorT< CellHandler_T, Cell_T >::iteratorT().

#### 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 60 of file cellhandler.h.

Referenced by CellHandler::iteratorT< CellHandler_T, Cell_T >::iteratorT().

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

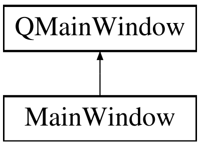
- cellhandler.h
- cellhandler.cpp

## 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 setCellHandler (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 forward ()

  *Skips the number of steps chosen by the user and sets the automaton on the last one.*
- void closeTab (int n)

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

## Public Member Functions

- MainWindow (QWidget ∗parent=nullptr)

## Private Member Functions

- void createIcons ()

  *Creates Icons for the MainWindow.*
- void createActions ()

  *Creates and connects QActions and associated 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 updateBoard (int index)

  *Updates cells on the board on the tab at the give index with the cellHandler's cells states.*
- void nextState (int n)

  *Shows the nth next state of the automaton on the board.*
- QTableWidget ∗ getBoard (int n)

## Private Attributes

- QTabWidget ∗ m_tabs
- QVector< CellHandler ∗ > m_cellHandlers
- QIcon m_fastBackwardIcon

  *Icons.*
- QIcon m_fastForwardIcon
- QIcon m_playIcon
- QIcon m_pauseIcon
- QIcon m_newIcon
- QIcon m_saveIcon

- QIcon m_openIcon
- QIcon m_resetIcon
- QAction ∗ m_playPause

  *Actions.*
- QAction ∗ m_nextState
- QAction ∗ m_previousState
- QAction ∗ m_fastForward
- QAction ∗ m_fastBackward
- QAction ∗ m_openAutomaton
- QAction ∗ m_saveAutomaton
- QAction ∗ m_newAutomaton
- QAction ∗ m_resetAutomaton
- QToolButton ∗ m_playPauseBt

  *Buttons.*
- QToolButton ∗ m_nextStateBt
- QToolButton ∗ m_previousStateBt
- QToolButton ∗ m_fastForwardBt
- QToolButton ∗ m_fastBackwardBt
- QToolButton ∗ m_openAutomatonBt
- QToolButton ∗ m_saveAutomatonBt
- QToolButton ∗ m_newAutomatonBt
- QToolButton ∗ m_resetBt
- QSpinBox ∗ m_jumpSpeed
- QLabel ∗ m_speedLabel

  *Simulation speed input.*
- QToolBar ∗ m_toolBar
- unsigned int m_boardHSize = 25

  *Toolbar containing the buttons.*
- 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 16 of file mainwindow.h.

## 6.7.2 Constructor & Destructor Documentation

### 6.7.2.1 MainWindow()

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

Definition at line 3 of file mainwindow.cpp.

References createActions(), createIcons(), createToolBar(), and m_tabs.

## 6.7.3 Member Function Documentation

### 6.7.3.1 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 324 of file mainwindow.cpp.

References m_tabs, and saveToFile().

Referenced by createTabs().

### 6.7.3.2 createActions()

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

Creates and connects QActions and associated buttons for the MainWindow.

Definition at line 51 of file mainwindow.cpp.

References forward(), m_fastBackward, m_fastBackwardBt, m_fastBackwardIcon, m_fastForward, m_fastForwardBt, m_fastForwardIcon, m_newAutomaton, m_newAutomatonBt, m_newIcon, m_openAutomaton, m_openAutomatonBt, m_openIcon, m_playIcon, m_playPause, m_playPauseBt, m_resetAutomaton, m_resetBt, m_resetIcon, m_saveAutomaton, m_saveAutomatonBt, m_saveIcon, openCreationWindow(), openFile(), and saveToFile().

Referenced by MainWindow().

### 6.7.3.3 createBoard()

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

### 6.7.3.4 createIcons()

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

Creates Icons for the MainWindow.

Definition at line 21 of file mainwindow.cpp.

References m_fastBackwardIcon, m_fastForwardIcon, m_newIcon, m_openIcon, m_pauseIcon, m_playIcon, m_resetIcon, and m_saveIcon.

Referenced by MainWindow().

#### 6.7.3.5 createTab()

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

Creates a new Tab with an empty board.

Definition at line 131 of file mainwindow.cpp.

References m_cellHandlers, and m_cellSize.

Referenced by openFile(), and setCellHandler().

#### 6.7.3.6 createTabs()

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

Creates a QTabWidget for the main window and displays it.

Definition at line 312 of file mainwindow.cpp.

References closeTab(), and m_tabs.

Referenced by openFile(), and setCellHandler().

#### 6.7.3.7 createToolBar()

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

Creates the toolBar for the MainWindow.

Definition at line 97 of file mainwindow.cpp.

References m_fastBackwardBt, m_fastForwardBt, m_jumpSpeed, m_newAutomatonBt, m_openAutomatonBt, m_playPauseBt, m_resetBt, m_saveAutomatonBt, m_speedLabel, and m_toolBar.

Referenced by MainWindow().

#### 6.7.3.8 forward

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

Skips the number of steps chosen by the user and sets the automaton on the last one.

Definition at line 300 of file mainwindow.cpp.

References m_jumpSpeed, and nextState().

Referenced by createActions().

#### 6.7.3.9 getBoard()

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

Definition at line 304 of file mainwindow.cpp.

References m_tabs.

Referenced by updateBoard().

#### 6.7.3.10 nextState()

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

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

Definition at line 243 of file mainwindow.cpp.

References m_cellHandlers, m_tabs, and updateBoard().

Referenced by forward().

#### 6.7.3.11 openCreationWindow

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

Opens the automaton creation window.

Definition at line 210 of file mainwindow.cpp.

References setCellHandler().

Referenced by createActions().

#### 6.7.3.12 openFile

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

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

Definition at line 176 of file mainwindow.cpp.

References createTab(), createTabs(), m_cellHandlers, m_tabs, and updateBoard().

Referenced by createActions().

#### 6.7.3.13 saveToFile

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

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

Definition at line 192 of file mainwindow.cpp.

References m_cellHandlers, and m_tabs.

Referenced by closeTab(), and createActions().

#### 6.7.3.14 setCellHandler

```
void MainWindow::setCellHandler (
             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 223 of file mainwindow.cpp.

References createTab(), createTabs(), m_cellHandlers, m_tabs, and updateBoard().

Referenced by openCreationWindow().

#### 6.7.3.15 updateBoard()

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

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

Definition at line 259 of file mainwindow.cpp.

References CellHandler::end(), getBoard(), CellHandler::getDimensions(), and m_cellHandlers.

Referenced by nextState(), openFile(), and setCellHandler().

### 6.7.4 Member Data Documentation

#### 6.7.4.1 m_boardHSize

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

Toolbar containing the buttons.

Board size settings

Definition at line 62 of file mainwindow.h.

#### 6.7.4.2 m_boardVSize

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

Definition at line 63 of file mainwindow.h.

#### 6.7.4.3 m_cellHandlers

```
QVector<CellHandler *> MainWindow::m_cellHandlers  [private]
```

Definition at line 21 of file mainwindow.h.

Referenced by createTab(), nextState(), openFile(), saveToFile(), setCellHandler(), and updateBoard().

#### 6.7.4.4 m_cellSize

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

Definition at line 64 of file mainwindow.h.

Referenced by createTab().

#### 6.7.4.5 m_fastBackward

```
QAction* MainWindow::m_fastBackward  [private]
```

Definition at line 38 of file mainwindow.h.

Referenced by createActions().

#### 6.7.4.6 m_fastBackwardBt

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

Definition at line 49 of file mainwindow.h.

Referenced by createActions(), and createToolBar().

#### 6.7.4.7 m_fastBackwardIcon

```
QIcon MainWindow::m_fastBackwardIcon  [private]
```

Icons.

Definition at line 24 of file mainwindow.h.

Referenced by createActions(), and createIcons().

#### 6.7.4.8 m_fastForward

```
QAction* MainWindow::m_fastForward  [private]
```

Definition at line 37 of file mainwindow.h.

Referenced by createActions().

#### 6.7.4.9 m_fastForwardBt

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

Definition at line 48 of file mainwindow.h.

Referenced by createActions(), and createToolBar().

#### 6.7.4.10 m_fastForwardIcon

```
QIcon MainWindow::m_fastForwardIcon  [private]
```

Definition at line 25 of file mainwindow.h.

Referenced by createActions(), and createIcons().

#### 6.7.4.11 m_jumpSpeed

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

Definition at line 56 of file mainwindow.h.

Referenced by createToolBar(), and forward().

#### 6.7.4.12 m_newAutomaton

```
QAction* MainWindow::m_newAutomaton  [private]
```

Definition at line 41 of file mainwindow.h.

Referenced by createActions().

#### 6.7.4.13 m_newAutomatonBt

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

Definition at line 52 of file mainwindow.h.

Referenced by createActions(), and createToolBar().

#### 6.7.4.14 m_newIcon

```
QIcon MainWindow::m_newIcon  [private]
```

Definition at line 28 of file mainwindow.h.

Referenced by createActions(), and createIcons().

#### 6.7.4.15 m_nextState

```
QAction* MainWindow::m_nextState  [private]
```

Definition at line 35 of file mainwindow.h.

#### 6.7.4.16 m_nextStateBt

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

Definition at line 46 of file mainwindow.h.

#### 6.7.4.17 m_openAutomaton

```
QAction* MainWindow::m_openAutomaton  [private]
```

Definition at line 39 of file mainwindow.h.

Referenced by createActions().

#### 6.7.4.18 m_openAutomatonBt

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

Definition at line 50 of file mainwindow.h.

Referenced by createActions(), and createToolBar().

#### 6.7.4.19 m_openIcon

```
QIcon MainWindow::m_openIcon  [private]
```

Definition at line 30 of file mainwindow.h.

Referenced by createActions(), and createIcons().

#### 6.7.4.20 m_pauseIcon

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

Definition at line 27 of file mainwindow.h.

Referenced by createIcons().

#### 6.7.4.21 m_playIcon

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

Definition at line 26 of file mainwindow.h.

Referenced by createActions(), and createIcons().

#### 6.7.4.22 m_playPause

```
QAction* MainWindow::m_playPause  [private]
```

Actions.

Definition at line 34 of file mainwindow.h.

Referenced by createActions().

#### 6.7.4.23 m_playPauseBt

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

Buttons.

Definition at line 45 of file mainwindow.h.

Referenced by createActions(), and createToolBar().

#### 6.7.4.24 m_previousState

```
QAction* MainWindow::m_previousState  [private]
```

Definition at line 36 of file mainwindow.h.

#### 6.7.4.25 m_previousStateBt

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

Definition at line 47 of file mainwindow.h.

#### 6.7.4.26 m_resetAutomaton

```
QAction* MainWindow::m_resetAutomaton [private]
```

Definition at line 42 of file mainwindow.h.

Referenced by createActions().

#### 6.7.4.27 m_resetBt

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

Definition at line 53 of file mainwindow.h.

Referenced by createActions(), and createToolBar().

#### 6.7.4.28 m_resetIcon

```
QIcon MainWindow::m_resetIcon [private]
```

Definition at line 31 of file mainwindow.h.

Referenced by createActions(), and createIcons().

#### 6.7.4.29 m_saveAutomaton

```
QAction* MainWindow::m_saveAutomaton [private]
```

Definition at line 40 of file mainwindow.h.

Referenced by createActions().

#### 6.7.4.30 m_saveAutomatonBt

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

Definition at line 51 of file mainwindow.h.

Referenced by createActions(), and createToolBar().

#### 6.7.4.31 m_saveIcon

```
QIcon MainWindow::m_saveIcon  [private]
```

Definition at line 29 of file mainwindow.h.

Referenced by createActions(), and createIcons().

#### 6.7.4.32 m_speedLabel

```
QLabel* MainWindow::m_speedLabel  [private]
```

Simulation speed input.

Definition at line 57 of file mainwindow.h.

Referenced by createToolBar().

#### 6.7.4.33 m_tabs

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

Definition at line 20 of file mainwindow.h.

Referenced by closeTab(), createTabs(), getBoard(), MainWindow(), nextState(), openFile(), saveToFile(), and setCellHandler().

#### 6.7.4.34 m_toolBar

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

Definition at line 59 of file mainwindow.h.

Referenced by createToolBar().

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

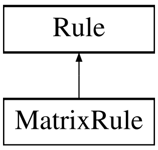
- mainwindow.h
- mainwindow.cpp

## 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.*
- void addNeighbourState (QVector< short > relativePosition, unsigned int matchState)

  *Add a possible state to a relative position.*
- 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.*

### Private Attributes

- QMap< QVector< short >, QVector< unsigned int > > m_matrix

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

### Additional Inherited Members

### 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**

| | |
|---|---|
| *finalState* | Final state if the rule match the cell |
| *currentStates* | 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 )
```

Add a possible state to a relative position.

Definition at line 60 of file matrixrule.cpp.

References m_matrix.

Referenced by Automate::loadRules().

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

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

Add multiples possible states to existents states.

Definition at line 67 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**

| | |
|---|---|
| *cell* | Cell 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 75 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  [private]
```

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

Definition at line 26 of file matrixrule.h.

Referenced by addNeighbourState(), matchCell(), and toJson().

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

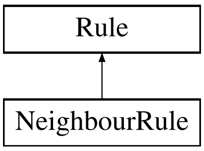
- matrixrule.h
- matrixrule.cpp

## 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.*
- virtual QJsonObject toJson () const

  *Return a QJsonObject to save the rule.*

### Private Member Functions

- bool inInterval (unsigned int matchingNeighbours) const

  *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.*

### Private 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.*

### Additional Inherited Members

### 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  [private]
```

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);
```

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

**Parameters**

| | |
|---|---|
| *matchingNeighbours* | 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**

| | |
|---|---|
| *c* | 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 147 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  [private]
```

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  [private]
```

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().

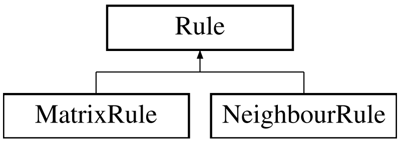
The documentation for this class was generated from the following files:

- 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)
- virtual QJsonObject toJson () const =0
- 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 12 of file rule.h.

## 6.10.2 Constructor & Destructor Documentation

### 6.10.2.1 Rule()

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

Definition at line 3 of file rule.cpp.

## 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 26 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**

| | |
|---|---|
| *c* | Cell to test |

Implemented in NeighbourRule, and MatrixRule.

#### 6.10.3.3 toJson()

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

Implemented in NeighbourRule, and MatrixRule.

Definition at line 9 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 16 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 15 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

# Chapter 7

# File Documentation

## 7.1 automate.cpp File Reference

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

## 7.2 automate.cpp

```
#include "automate.h"

bool Automate::loadRules(const QJsonArray &json)
{

    for (QJsonArray::const_iterator it = json.begin(); it != json.end(); ++it)
    {
        if (!it->isObject())
            return false;
        QJsonObject ruleJson = it->toObject();

        if (!ruleJson.contains("type") || !ruleJson["type"].isString())
            return false;
        if (!ruleJson.contains("finalState") || !ruleJson["finalState"].isDouble())
            return false;
        if (!ruleJson.contains("currentStates") || !ruleJson["currentStates"].isArray())
            return false;

        QVector<unsigned int> currentStates;
        QJsonArray statesJson = ruleJson["currentStates"].toArray();
        for (unsigned int i = 0; i < statesJson.size(); i++)
        {
            if (!statesJson.at(i).isDouble())
                return false;
            currentStates.push_back(statesJson.at(i).toInt());
        }

        if (!ruleJson["type"].toString().compare("neighbour", Qt::CaseInsensitive))
        {
            if (!ruleJson.contains("neighbourNumberMin") || !ruleJson["neighbourNumberMin"].isDouble())
                return false;
            if (!ruleJson.contains("neighbourNumberMax") || !ruleJson["neighbourNumberMax"].isDouble())
                return false;



            QPair<unsigned int, unsigned int> nbrNeighbourInterval(ruleJson["neighbourNumberMin"].toInt(), ruleJson["neighbourNumberMax"].toInt());
            NeighbourRule *newRule;
            if (ruleJson.contains("neighbourStates"))
            {
                if (!ruleJson["neighbourStates"].isArray())
                    return false;
                QSet<unsigned int> neighbourStates;

                QJsonArray statesJson = ruleJson["neighbourStates"].toArray();
                for (unsigned int i = 0; i < statesJson.size(); i++)
                {
                    if (!statesJson.at(i).isDouble())
                        return false;
                    neighbourStates.insert(statesJson.at(i).toInt());
                }
                newRule = new NeighbourRule((unsigned int)ruleJson["finalState"].toInt(), currentStates, nbrNeighbourInterval, neighbourStates);
            }
            else
                newRule = new NeighbourRule((unsigned int)ruleJson["finalState"].toInt(), currentStates, nbrNeighbourInterval);
            m_rules.push_back(newRule);
        }
        else if (!ruleJson["type"].toString().compare("matrix", Qt::CaseInsensitive))
        {
            MatrixRule *newRule = new MatrixRule((unsigned int)ruleJson["finalState"].toInt(), currentStates);
            if (ruleJson.contains("neighbours"))
            {
                if (!ruleJson["neighbours"].isArray())
                    return false;
                QJsonArray neighboursJson = ruleJson["neighbours"].toArray();
                for (unsigned int i = 0; i < neighboursJson.size(); i++)
                {
                    if (!neighboursJson.at(i).isObject())
                        return false;

                    if (!neighboursJson.at(i).toObject().contains("relativePosition") || !neighboursJson.at(i).toObject()["relativePosition"].isArray())
                        return false;
                    if (!neighboursJson.at(i).toObject().contains("neighbourStates") || !neighboursJson.at(i).toObject()["neighbourStates"].isArray())
                        return false;

                    QVector<unsigned int> neighbourStates;


                    QJsonArray statesJson = neighboursJson.at(i).toObject()["neighbourStates"].toArray();
                    for (unsigned int j = 0; j < statesJson.size(); j++)
                    {
                        if (!statesJson.at(j).isDouble())
                            return false;
                        neighbourStates.push_back(statesJson.at(j).toInt());
                    }

                    QVector<short> relativePosition;
                    QJsonArray positionJson = neighboursJson.at(i).toObject()["relativePosition"].toArray();
                    for (unsigned int j = 0; j < positionJson.size(); j++)
                    {
                        if (!positionJson.at(j).isDouble())
                            return false;
                        relativePosition.push_back(positionJson.at(j).toInt());
                    }
                    if (relativePosition.size() != m_cellHandler->getDimensions().size())
                        return false;
                    newRule->addNeighbourState(relativePosition, neighbourStates);
                }

            }
            m_rules.push_back(newRule);


        }
        else
            return false;

    }
    return true;
}

Automate::Automate(QString cellHandlerFilename)
{
    m_cellHandler = new CellHandler(cellHandlerFilename);

}

Automate::Automate(const QVector<unsigned int> dimensions, CellHandler::generationTypes type, unsigned int stateMax, unsigned int density)
{
    m_cellHandler = new CellHandler(dimensions, type, stateMax, density);

}

```

```
Automate::Automate(QString cellHandlerFilename, QString ruleFilename)
{
    m_cellHandler = new CellHandler(cellHandlerFilename);

    QFile ruleFile(ruleFilename);
    if (!ruleFile.open(QIODevice::ReadOnly | QIODevice::Text)) {
        qWarning("Couldn't open given file.");
        throw QString(QObject::tr("Couldn't open given file"));
    }

    QJsonParseError parseErr;
    QJsonDocument loadDoc(QJsonDocument::fromJson(ruleFile.readAll(), &parseErr));

    ruleFile.close();


    if (loadDoc.isNull() || loadDoc.isEmpty())
    {
        qWarning() << "Could not read data : ";
        qWarning() << parseErr.errorString();
        throw QString(parseErr.errorString());
    }

    if (!loadDoc.isArray())
    {
        qWarning() << "We need an array of rules !";
        throw QString(QObject::tr("We need an array of rules!"));
    }

    loadRules(loadDoc.array());

}

Automate::~Automate()
{
    delete m_cellHandler;
    for (QList<const Rule*>::iterator it = m_rules.begin(); it != m_rules.end(); ++it)
    {
        delete *it;
    }
}

bool Automate::saveRules(QString filename) const
{
    QFile ruleFile(filename);
    if (!ruleFile.open(QIODevice::WriteOnly | QIODevice::Text)) {
        qWarning("Couldn't open given file.");
        throw QString(QObject::tr("Couldn't open given file"));
    }

    QJsonArray array;

    for (QList<const Rule*>::const_iterator it = m_rules.cbegin(); it != m_rules.cend(); ++it)
        array.append((*it)->toJson());

    QJsonDocument doc(array);

    ruleFile.write(doc.toJson());

    return true;
}

bool Automate::saveCells(QString filename) const
{
    if (m_cellHandler != nullptr)
        return m_cellHandler->save(filename);
    return false;
}

bool Automate::saveAll(QString cellHandlerFilename, QString rulesFilename) const
{
    return saveRules(rulesFilename) && saveCells(cellHandlerFilename);
}

void Automate::addRule(const Rule *newRule)
{
    m_rules.push_back(newRule);
}

void Automate::setRulePriority(const Rule *rule, unsigned int newPlace)
{
    m_rules.move(m_rules.indexOf(rule), newPlace);
}

const QList<const Rule *> &Automate::getRules() const
{
    return m_rules;
}

bool Automate::run(unsigned int nbSteps) //void instead ?
{
    for(unsigned int i = 0; i<nbSteps; ++i)
    {
        for (CellHandler::iterator it = m_cellHandler->
begin(); it != m_cellHandler->end(); ++it)
        {
            for (QList<const Rule*>::iterator rule = m_rules.begin(); rule !=
m_rules.end() ; ++rule)
            {
                if((*rule)->matchCell(*it)) //if the cell matches with the rule, its state is changed
                {
                    it->setState((*rule)->getCellOutputState());
                    break;
                }
            }


        }
        m_cellHandler->nextStates(); //apply the changes to all the cells
 simultaneously
    }
    return true;

}

const CellHandler &Automate::getCellHandler() const
{
    return *m_cellHandler;
}
```

## 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

## 7.4 automate.h

```
#ifndef AUTOMATE_H
#define AUTOMATE_H
#include <QVector>
#include <QList>

#include "cellhandler.h"
#include "rule.h"
#include "neighbourrule.h"
#include "matrixrule.h"


class Automate
{
private:
    CellHandler* m_cellHandler = nullptr;
    QList<const Rule*> m_rules;
    friend class AutomateHandler;

    bool loadRules(const QJsonArray &json);
public:
    Automate(QString filename);
    Automate(const QVector<unsigned int> dimensions, CellHandler::generationTypes type = CellHandler::empty, unsigned int stateMax = 1, unsigned int density = 20);
    Automate(QString cellHandlerFilename, QString ruleFilename);
    virtual ~Automate();

    bool saveRules(QString filename) const ;
    bool saveCells(QString filename) const ;
    bool saveAll(QString cellHandlerFilename, QString rulesFilename)const ;

    void addRule(const Rule* newRule);
    void setRulePriority(const Rule* rule, unsigned int newPlace);
    const QList<const Rule *> &getRules() const;



public:
    bool run(unsigned int nbSteps = 1);
    const CellHandler& getCellHandler() const;

};

#endif // AUTOMATE_H
```

## 7.5 automatehandler.cpp File Reference

```
#include "automatehandler.h"
```

### Variables

- AutomateHandler ∗ m_activeAutomate = nullptr

  *Initialization of the static value.*

### 7.5.1 Variable Documentation

#### 7.5.1.1 m_activeAutomate

```
AutomateHandler* m_activeAutomate = nullptr
```

Initialization of the static value.

Definition at line 5 of file automatehandler.cpp.

## 7.6 automatehandler.cpp

```
#include "automatehandler.h"

AutomateHandler * m_activeAutomate = nullptr;

AutomateHandler::~AutomateHandler()
{

}

Automate & AutomateHandler::getActiveAutomate()
{
   /* if(!m_activeAutomate)
        m_activeAutomate = new Automate();
    return *m_activeAutomate;*/
}


void AutomateHandler::deleteActiveAutomate()
{
    /*if(m_activeAutomate)
        delete m_activeAutomate;
    m_activeAutomate = nullptr;*/
}

void AutomateHandler::setActiveAutomate(unsigned int activeAutomate)
{

}
```

## 7.7 automatehandler.h File Reference

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

### Classes

- class AutomateHandler

  *Implementation of singleton design pattern.*

## 7.8 automatehandler.h

```
#ifndef AUTOMATEHANDLER_H
#define AUTOMATEHANDLER_H

#include "automate.h"


class AutomateHandler
{
private:
    static AutomateHandler * m_activeAutomate;
    AutomateHandler(const AutomateHandler & a) = delete;
    AutomateHandler & operator=(const AutomateHandler & a) = delete;
    ~AutomateHandler();
public:
    static Automate & getActiveAutomate();
    static void deleteActiveAutomate();
    void setActiveAutomate(unsigned int activeAutomate);
};

#endif // AUTOMATEHANDLER_H
```

## 7.9 cell.cpp File Reference

```
#include "cell.h"
```

## 7.10 cell.cpp

```
#include "cell.h"

Cell::Cell(unsigned int state):
    m_state(state), m_nextState(state)
{

}

void Cell::setState(unsigned int state)
{
    m_nextState = state;
}

void Cell::validState()
{
    m_state = m_nextState;
}

void Cell::forceState(unsigned int state)
{
    m_state = m_nextState = state;
}

unsigned int Cell::getState() const
{
    return m_state;
}

bool Cell::addNeighbour(const Cell* neighbour, const QVector<short> relativePosition)
{
    if (m_neighbours.count(relativePosition))
        return false;

    m_neighbours.insert(relativePosition, neighbour);
    return true;
}

QMap<QVector<short>, const Cell *> Cell::getNeighbours() const
{
    return m_neighbours;
}

const Cell *Cell::getNeighbour(QVector<short> relativePosition) const
{
    return m_neighbours.value(relativePosition, nullptr);
}

unsigned int Cell::countNeighbours(unsigned int filterState) const
{
    unsigned int count = 0;
    for (QMap<QVector<short>, const Cell*>::const_iterator it = m_neighbours.begin(); it != m_neighbours.end(); ++it)
    {
        if ((*it)->getState() == filterState)
            count++;
    }
    return count;
}

unsigned int Cell::countNeighbours() const
{
    unsigned int count = 0;
    for (QMap<QVector<short>, const Cell*>::const_iterator it = m_neighbours.begin(); it != m_neighbours.end(); ++it)
    {
        if ((*it)->getState() != 0)
            count++;
    }
    return count;
}

QVector<short> Cell::getRelativePosition(const QVector<unsigned int> cellPosition,
const QVector<unsigned int> neighbourPosition)
{
    if (cellPosition.size() != neighbourPosition.size())
    {
        throw QString(QObject::tr("Different size of position vectors"));
    }
    QVector<short> relativePosition;
    for (short i = 0; i < cellPosition.size(); i++)
        relativePosition.push_back(neighbourPosition.at(i) - cellPosition.at(i));

    return relativePosition;
}
```

## 7.11 cell.h File Reference

```
#include <QVector>
#include <QDebug>
```

### Classes

- class Cell

  *Contains the state, the next state and the neighbours.*

## 7.12 cell.h

```
#ifndef CELL_H
#define CELL_H

#include <QVector>
#include <QDebug>

class Cell
{
public:
    Cell(unsigned int state = 0);

    void setState(unsigned int state);
    void validState();
    void forceState(unsigned int state);
    unsigned int getState() const;

    bool addNeighbour(const Cell* neighbour, const QVector<short> relativePosition);
    QMap<QVector<short>, const Cell*> getNeighbours() const;
    const Cell* getNeighbour(QVector<short> relativePosition) const;

    unsigned int countNeighbours(unsigned int filterState) const;
    unsigned int countNeighbours() const;

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

private:
    unsigned int m_state;
    unsigned int m_nextState;

    QMap<QVector<short>, const Cell*> m_neighbours;
};

#endif // CELL_H
```

## 7.13 cellhandler.cpp File Reference

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

## 7.14 cellhandler.cpp

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

CellHandler::CellHandler(const QString filename)
{
    QFile loadFile(filename);
    if (!loadFile.open(QIODevice::ReadOnly | QIODevice::Text)) {
        qWarning("Couldn't open given file.");
        throw QString(QObject::tr("Couldn't open given file"));
    }

    QJsonParseError parseErr;
    QJsonDocument loadDoc(QJsonDocument::fromJson(loadFile.readAll(), &parseErr));

    loadFile.close();


    if (loadDoc.isNull() || loadDoc.isEmpty()) {
        qWarning() << "Could not read data : ";
        qWarning() << parseErr.errorString();
        throw QString(parseErr.errorString());
    }

    // Loadding of the json file
    if (!load(loadDoc.object()))
    {
        qWarning("File not valid");
        throw QString(QObject::tr("File not valid"));
    }

    foundNeighbours();


}

CellHandler::CellHandler(const QJsonObject& json)
{
    if (!load(json))
    {
        qWarning("Json not valid");
        throw QString(QObject::tr("Json not valid"));
    }

    foundNeighbours();

}


CellHandler::CellHandler(const QVector<unsigned int> dimensions, generationTypes type, unsigned int stateMax, unsigned int density)
{
    m_dimensions = dimensions;
    QVector<unsigned int> position;
    unsigned int size = 1;

    // Set position vector to 0

    for (unsigned short i = 0; i < m_dimensions.size(); i++)
    {
        position.push_back(0);
        size *= m_dimensions.at(i);
    }


    // Creation of cells
    for (unsigned int j = 0; j < size; j++)
    {
        m_cells.insert(position, new Cell(0));

        positionIncrement(position);
    }

    foundNeighbours();

    if (type != empty)
        generate(type, stateMax, density);

}

CellHandler::~CellHandler()
{
    for (QMap<QVector<unsigned int>, Cell* >::iterator it = m_cells.begin(); it != m_cells.end(); ++it)
    {
        delete it.value();
    }
}

Cell *CellHandler::getCell(const QVector<unsigned int> position) const
{
    return m_cells.value(position);
}

QVector<unsigned int> CellHandler::getDimensions() const
{
    return m_dimensions;
}

void CellHandler::nextStates() const
{
    for (QMap<QVector<unsigned int>, Cell* >::const_iterator it = m_cells.begin(); it != m_cells.end(); ++it)
    {
        it.value()->validState();
    }
}

bool CellHandler::save(QString filename) const
{
    QFile saveFile(filename);
    if (!saveFile.open(QIODevice::WriteOnly)) {
        qWarning("Couldn't create or open given file.");
        throw QString(QObject::tr("Couldn't create or open given file"));
    }

    QJsonObject json;
    QString stringDimension;
    // Creation of the dimension string
    for (int i = 0; i < m_dimensions.size(); i++)
    {
        if (i != 0)
            stringDimension.push_back("x");
        stringDimension.push_back(QString::number(m_dimensions.at(i)));
    }
    json["dimensions"] = QJsonValue(stringDimension);

    QJsonArray cells;
    for (CellHandler::const_iterator it = begin(); it != end(); ++it)
    {
        cells.append(QJsonValue((int)it->getState()));
    }
    json["cells"] = cells;


    QJsonDocument saveDoc(json);
    saveFile.write(saveDoc.toJson());

    saveFile.close();
    return true;
}

void CellHandler::generate(CellHandler::generationTypes type, unsigned int stateMax, unsigned short density)
{
    if (type == random)
    {
        QVector<unsigned int> position;
        for (unsigned short i = 0; i < m_dimensions.size(); i++)
        {
            position.push_back(0);
        }
        QRandomGenerator generator((float)qrand()*(float)time_t()/RAND_MAX);
        for (int j = 0; j < m_cells.size(); j++)
        {
            unsigned int state = 0;
            // 0 have (1-density)% of chance of being generate
            if (generator.generateDouble()*100.0 < density)
                 state = (float)(generator.generateDouble()*stateMax) +1;
            if (state > stateMax)
                state = stateMax;
            m_cells.value(position)->forceState(state);

            positionIncrement(position);
        }
    }
    else if (type == symetric)
    {
        QVector<unsigned int> position;
        for (short i = 0; i < m_dimensions.size(); i++)
        {
            position.push_back(0);
        }

        QRandomGenerator generator((float)qrand()*(float)time_t()/RAND_MAX);
        QVector<unsigned int> savedStates;
        for (int j = 0; j < m_cells.size(); j++)
        {
            if (j % m_dimensions.at(0) == 0)
                savedStates.clear();
            if (j % m_dimensions.at(0) < (m_dimensions.at(0)+1) / 2)
            {
                unsigned int state = 0;
                // 0 have (1-density)% of chance of being generate
                if (generator.generateDouble()*100.0 < density)
                    state = (float)(generator.generateDouble()*stateMax) +1;
                if (state > stateMax)
                    state = stateMax;
                savedStates.push_back(state);
                m_cells.value(position)->forceState(state);
            }
            else
            {
                unsigned int i = savedStates.size() - (j % m_dimensions.at(0) - (m_dimensions.at(0)-1)/2 + (m_dimensions.at(0) % 2 == 0 ? 0 : 1));
                m_cells.value(position)->forceState(savedStates.at(i));
            }
            positionIncrement(position);


        }

    }
}

void CellHandler::print(std::ostream &stream) const
{
    for (const_iterator it = begin(); it != end(); ++it)
    {
        for (unsigned int d = 0; d < it.changedDimension(); d++)
            stream << std::endl;
        stream << it->getState() << " ";

    }

}

CellHandler::iterator CellHandler::begin()
{
    return iterator(this);
}

CellHandler::const_iterator CellHandler::begin() const
{
    return const_iterator(this);
}

bool CellHandler::end() const
{
    return true;
}

bool CellHandler::load(const QJsonObject &json)
{
    if (!json.contains("dimensions") || !json["dimensions"].isString())
        return false;

    // RegExp to validate dimensions field format : "10x10"
    QRegExpValidator dimensionValidator(QRegExp("([0-9]*x?)*"));
    QString stringDimensions = json["dimensions"].toString();
    int pos= 0;
    if (dimensionValidator.validate(stringDimensions, pos) != QRegExpValidator::Acceptable)
        return false;

    // Split of dimensions field : "10x10" => "10", "10"
    QRegExp rx("x");
    QStringList list = json["dimensions"].toString().split(rx, QString::SkipEmptyParts);

    int product = 1;
    // Dimensions construction
    for (int i = 0; i < list.size(); i++)
    {
        product = product * list.at(i).toInt();
        m_dimensions.push_back(list.at(i).toInt());
    }
    if (!json.contains("cells") || !json["cells"].isArray())
        return false;

    QJsonArray cells = json["cells"].toArray();
    if (cells.size() != product)
        return false;

    QVector<unsigned int> position;
    // Set position vector to 0
    for (unsigned short i = 0; i < m_dimensions.size(); i++)
    {
        position.push_back(0);
    }

    // Creation of cells
    for (int j = 0; j < cells.size(); j++)
    {
        if (!cells.at(j).isDouble())
            return false;
        if (cells.at(j).toDouble() < 0)
            return false;
        m_cells.insert(position, new Cell(cells.at(j).toDouble()));

        positionIncrement(position);
    }

    return true;

}

void CellHandler::foundNeighbours()
{
    QVector<unsigned int> currentPosition;
    // Set position vector to 0
    for (unsigned short i = 0; i < m_dimensions.size(); i++)
    {
        currentPosition.push_back(0);
    }
    // Modification of all the cells
    for (int j = 0; j < m_cells.size(); j++)
    {
        // Get the list of the neighbours positions
        // This function is recursive
        QVector<QVector<unsigned int> > listPosition(getListNeighboursPositions(currentPosition));

        // Adding neighbours
        for (int i = 0; i < listPosition.size(); i++)
            m_cells.value(currentPosition)->addNeighbour(m_cells.value(listPosition.at(i)), Cell::getRelativePosition(currentPosition, listPosition.at(i)));
        positionIncrement(currentPosition);
    }

}

void CellHandler::positionIncrement(QVector<unsigned int> &pos, unsigned int value) const
{
    pos.replace(0, pos.at(0) + value); // adding the value to the first digit

    // Carry management
    for (unsigned short i = 0; i < m_dimensions.size(); i++)
    {
        if (pos.at(i) >= m_dimensions.at(i) && pos.at(i) < m_dimensions.at(i)*2)
        {
            pos.replace(i, 0);
            if (i + 1 != m_dimensions.size())
                pos.replace(i+1, pos.at(i+1)+1);
        }
        else if (pos.at(i) >= m_dimensions.at(i))
        {
            pos.replace(i, pos.at(i) - m_dimensions.at(i));
            if (i + 1 != m_dimensions.size())
                pos.replace(i+1, pos.at(i+1)+1);
            i--;
        }

    }
}

QVector<QVector<unsigned int> >& CellHandler::getListNeighboursPositions(const QVector<unsigned int> position) const
{
    QVector<QVector<unsigned int> > *list = getListNeighboursPositionsRecursive(position, position.size(), position);
    // We remove the position of the cell
    list->removeAll(position);
    return *list;
}
```

```

QVector<QVector<unsigned int> >*
CellHandler::getListNeighboursPositionsRecursive(const
QVector<unsigned int> position, unsigned int dimension, QVector<unsigned int> lastAdd) const
{
    if (dimension == 0) // Stop condition
    {
        QVector<QVector<unsigned int> > *list = new QVector<QVector<unsigned int> >;
        return list;
    }
    QVector<QVector<unsigned int> > *listPositions = new QVector<QVector<unsigned int> >;

    QVector<unsigned int> modifiedPosition(lastAdd);

    // "x_d - 1" tree
    if (modifiedPosition.at(dimension-1) != 0) // Avoid "negative" position
        modifiedPosition.replace(dimension-1, position.at(dimension-1) - 1);
    listPositions->append(*getListNeighboursPositionsRecursive(position,
 dimension -1, modifiedPosition));
    if (!listPositions->count(modifiedPosition))
        listPositions->push_back(modifiedPosition);

    // "x_d" tree
    modifiedPosition.replace(dimension-1, position.at(dimension-1));
    listPositions->append(*getListNeighboursPositionsRecursive(position,
 dimension -1, modifiedPosition));
    if (!listPositions->count(modifiedPosition))
        listPositions->push_back(modifiedPosition);

    // "x_d + 1" tree
    if (modifiedPosition.at(dimension -1) + 1 < m_dimensions.at(dimension-1)) // Avoid position
 out of the cell space
        modifiedPosition.replace(dimension-1, position.at(dimension-1) +1);
    listPositions->append(*getListNeighboursPositionsRecursive(position,
 dimension -1, modifiedPosition));
    if (!listPositions->count(modifiedPosition))
        listPositions->push_back(modifiedPosition);

    return listPositions;

}

template<typename CellHandler_T, typename Cell_T>
CellHandler::iteratorT<CellHandler_T,Cell_T>::iteratorT
(CellHandler_T *handler):
        m_handler(handler), m_changedDimension(0)
{
    // Initialisation of m_position
    for (unsigned short i = 0; i < handler->m_dimensions.size(); i++)
    {
        m_position.push_back(0);
    }
    m_zero = m_position;
}

template<typename CellHandler_T, typename Cell_T>
CellHandler::iteratorT<CellHandler_T,Cell_T> &
CellHandler::iteratorT<CellHandler_T,Cell_T>::operator++
()
{
    m_position.replace(0, m_position.at(0) + 1); // adding the value to the first digit

    m_changedDimension = 0;
    // Carry management
    for (unsigned short i = 0; i < m_handler->m_dimensions.size(); i++)
    {
        if (m_position.at(i) >= m_handler->m_dimensions.at(i))
        {
            m_position.replace(i, 0);
            m_changedDimension++;
            if (i + 1 != m_handler->m_dimensions.size())
                m_position.replace(i+1, m_position.at(i+1)+1);
        }

    }
    // If we return to zero, we have finished
    if (m_position == m_zero)
        m_finished = true;

    return *this;

}

template<typename CellHandler_T, typename Cell_T>
Cell_T* CellHandler::iteratorT<CellHandler_T,Cell_T>::operator->
() const
{
    return m_handler->m_cells.value(m_position);
}


template<typename CellHandler_T, typename Cell_T>
Cell_T *CellHandler::iteratorT<CellHandler_T,Cell_T>::operator*
() const
{
    return m_handler->m_cells.value(m_position);
}

template<typename CellHandler_T, typename Cell_T>
unsigned int CellHandler::iteratorT<CellHandler_T,Cell_T>::changedDimension
() const
{
    return m_changedDimension;
}

```

## 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

```
#ifndef CELLHANDLER_H
#define CELLHANDLER_H

#include <QString>
#include <QFile>
#include <QJsonDocument>
#include <QtWidgets>
#include <QMap>
#include <QRegExpValidator>
#include <QDebug>

#include "cell.h"



class CellHandler
{

    template <typename CellHandler_T, typename Cell_T>
    class iteratorT
    {
        friend class CellHandler;
    public:
        iteratorT(CellHandler_T* handler);

        iteratorT& operator++();
        Cell_T* operator->() const;
        Cell_T* operator*() const;

        bool operator!=(bool finished) const { return (m_finished != finished); }
        unsigned int changedDimension() const;



    private:
        CellHandler_T *m_handler;
        QVector<unsigned int> m_position;
        bool m_finished = false;
        QVector<unsigned int> m_zero;
        unsigned int m_changedDimension;
    };
public:
    typedef iteratorT<const CellHandler, const Cell> const_iterator;
    typedef iteratorT<CellHandler, Cell> iterator;

    enum generationTypes {
        empty,
        random,
        symetric
    };

    CellHandler(const QString filename);
    CellHandler(const QJsonObject &json);
    CellHandler(const QVector<unsigned int> dimensions, generationTypes type = empty, unsigned int stateMax = 1, unsigned int density = 20);
    virtual ~CellHandler();

    Cell* getCell(const QVector<unsigned int> position) const;
    QVector<unsigned int> getDimensions() const;
    void nextStates() const;

    bool save(QString filename) const;

    void generate(generationTypes type, unsigned int stateMax = 1, unsigned short density = 50);
    void print(std::ostream &stream) const;

    const_iterator begin() const;
    iterator begin();
    bool end() const;

private:
    bool load(const QJsonObject &json);
    void foundNeighbours();
    void positionIncrement(QVector<unsigned int> &pos, unsigned int value = 1) const;
    QVector<QVector<unsigned int> > *getListNeighboursPositionsRecursive(const QVector<unsigned int> position, unsigned int dimension, QVector<unsigned int> lastAdd) const;
    QVector<QVector<unsigned int> > &getListNeighboursPositions(const QVector<unsigned int> position) const;

    QVector<unsigned int> m_dimensions;
    QMap<QVector<unsigned int>, Cell* > m_cells;
};

template class CellHandler::iteratorT<CellHandler, Cell>;
template class CellHandler::iteratorT<const CellHandler, const Cell>;

#endif // CELLHANDLER_H
```

## 7.17 creationdialog.cpp File Reference

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

## 7.18 creationdialog.cpp

```
#include "creationdialog.h"
```

```
#include <iostream>


CreationDialog::CreationDialog(QWidget *parent)
{
    QLabel *m_dimLabel= new QLabel(tr("Write your dimensions and their size, separated by a comma.\n"
                           "For instance, ’25,25 ’ will create a 2-dimensional 25x25 Automaton. "));
    QLabel *m_densityLabel = new QLabel(tr("Density :"));
    QLabel *m_stateMaxLabel = new QLabel(tr("Max state :"));
    m_densityBox = new QSpinBox();
    m_densityBox->setValue(20);
    m_stateMaxBox = new QSpinBox();
    m_stateMaxBox->setValue(1);

    QHBoxLayout *densityLayout = new QHBoxLayout();
    densityLayout->addWidget(m_densityLabel);
    densityLayout->addWidget(m_densityBox);

    QHBoxLayout *stateMaxLayout = new QHBoxLayout();
    stateMaxLayout->addWidget(m_stateMaxLabel);
    stateMaxLayout->addWidget(m_stateMaxBox);

    m_dimensionsEdit = new QLineEdit;
    QRegExp rgx("([0-9]+,)*");
    QRegExpValidator *v = new QRegExpValidator(rgx, this);
    m_dimensionsEdit->setValidator(v);
    m_doneBt = new QPushButton(tr("Done !"));

    QVBoxLayout *layout = new QVBoxLayout;

    QGroupBox *grpBox = createGenButtons();

    layout->addWidget(m_dimLabel);
    layout->addWidget(m_dimensionsEdit);
    layout->addLayout(densityLayout);
    layout->addLayout(stateMaxLayout);
    layout->addWidget(grpBox);
    layout->addWidget(m_doneBt);
    setLayout(layout);

    connect(m_doneBt, SIGNAL(clicked(bool)), this, SLOT(processSettings()));

}

QGroupBox *CreationDialog::createGenButtons(){
    m_groupBox = new QGroupBox(tr("Cell generation settings"));
    m_empGen = new QRadioButton(tr("&Empty Board"));
    m_randGen = new QRadioButton(tr("&Random"));
    m_symGen = new QRadioButton(tr("&Symmetrical"));

    QVBoxLayout *layout = new QVBoxLayout;
    layout->addWidget(m_empGen);
    layout->addWidget(m_randGen);
    layout->addWidget(m_symGen);

    m_groupBox->setLayout(layout);

    return m_groupBox;
}

void CreationDialog::processSettings(){
    QString dimensions = m_dimensionsEdit->text();
    if(dimensions.length() == 0){
        QMessageBox messageBox;
        messageBox.critical(0,"Error","You must specify valid dimensions !");
        messageBox.setFixedSize(500,200);
    }
    else{
        CellHandler::generationTypes genType;
        if(m_symGen->isChecked()) genType = CellHandler::generationTypes::symetric;
        else if(m_randGen->isChecked()) genType = CellHandler::generationTypes::random;
        else genType = CellHandler::generationTypes::empty;
        QStringList dimList = m_dimensionsEdit->text().split(",");
        QVector<unsigned int> dimensions;
        for(int i = 0; i < dimList.size(); i++) dimensions.append(dimList.at(i).toInt());

        emit settingsFilled(dimensions, genType, m_stateMaxBox->value(),
m_densityBox->value());
        this->close();
    }

}

```

## 7.19 creationdialog.h File Reference

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

### Classes

- class CreationDialog

  *Automaton creation dialog box.*

## 7.20 creationdialog.h

```
#ifndef CREATIONDIALOG_H
#define CREATIONDIALOG_H

#include <QtWidgets>
#include "cellhandler.h"

class CreationDialog : public QDialog
{
    Q_OBJECT

public:
    CreationDialog(QWidget *parent = 0);

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

public slots:
    void processSettings();

private:
    QLineEdit *m_dimensionsEdit;
    QSpinBox *m_densityBox;
    QSpinBox *m_stateMaxBox;
    QPushButton *m_doneBt;

    QGroupBox *m_groupBox;
    QRadioButton *m_empGen;
    QRadioButton *m_randGen;
    QRadioButton *m_symGen;

    QGroupBox *createGenButtons();






};

#endif // CREATEDIALOG_H
```

## 7.21 main.cpp File Reference

```
#include <QApplication>
#include <QDebug>
#include "cell.h"
#include "mainwindow.h"
```

## 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 6 of file main.cpp.

## 7.22 main.cpp

```
00001 #include <QApplication>
00002 #include <QDebug>
00003 #include "cell.h"
00004 #include "mainwindow.h"
00005
00006 int main(int argc, char * argv[])
00007 {
00008     QApplication app(argc, argv);
00009     QApplication::setAttribute(Qt::AA_UseHighDpiPixmaps);
00010     MainWindow w;
00011     w.show();
00012     return app.exec();
00013
00014 }
```

## 7.23 mainwindow.cpp File Reference

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

## 7.24 mainwindow.cpp

```
00001 #include "mainwindow.h"
00002 #include <iostream>
00003 MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent)
00004 {
00005     createIcons();
00006     createActions();
00007     createToolBar();
00008
00009
00010     setMinimumSize(500,500);
00011     setWindowTitle("AutoCell");
00012
00013     m_tabs = NULL;
00014 }
00015
00021 void MainWindow::createIcons(){
    QPixmap fastBackwardPm(":/icons/icons/fast-backward.svg");
    QPixmap fastBackwardHoveredPm(":/icons/icons/fast-backward-full.svg");
    QPixmap fastForwardPm(":/icons/icons/fast-forward.svg");
    QPixmap fastForwardHoveredPm(":/icons/icons/fast-forward-full.svg");
    QPixmap playPm(":/icons/icons/play.svg");
    QPixmap playHoveredPm(":/icons/icons/play-full.svg");
    QPixmap newPm(":/icons/icons/new.svg");
    QPixmap openPm(":/icons/icons/open.svg");
    QPixmap savePm(":/icons/icons/save.svg");
    QPixmap pausePm(":/icons/icons/pause.svg");
    QPixmap resetPm(":/icons/icons/reset.svg");

    m_fastBackwardIcon.addPixmap(fastBackwardPm, QIcon::Normal, QIcon::Off);
    m_fastBackwardIcon.addPixmap(fastBackwardHoveredPm, QIcon::Active, QIcon::Off);
    m_fastForwardIcon.addPixmap(fastForwardPm, QIcon::Normal, QIcon::Off);
    m_fastForwardIcon.addPixmap(fastForwardHoveredPm, QIcon::Active, QIcon::Off);
    m_playIcon.addPixmap(playPm, QIcon::Normal, QIcon::Off);
    m_playIcon.addPixmap(playHoveredPm, QIcon::Active, QIcon::Off);
    m_pauseIcon.addPixmap(pausePm, QIcon::Normal, QIcon::Off);
    m_newIcon.addPixmap(newPm, QIcon::Normal, QIcon::Off);
    m_saveIcon.addPixmap(savePm, QIcon::Normal, QIcon::Off);
    m_openIcon.addPixmap(openPm, QIcon::Normal, QIcon::Off);
    m_resetIcon.addPixmap(resetPm, QIcon::Normal, QIcon::Off);
}

void MainWindow::createActions(){
    m_fastBackward = new QAction(m_fastBackwardIcon, tr("&fast backward"), this);
    m_fastForward = new QAction(m_fastForwardIcon, tr("&fast forward"), this);
    m_playPause = new QAction(m_playIcon, tr("Play"), this);
    m_saveAutomaton = new QAction(m_saveIcon, tr("Save automaton"), this);
    m_newAutomaton = new QAction(m_newIcon, tr("New automaton"), this);
    m_openAutomaton = new QAction(m_openIcon, tr("Open automaton"), this);
    m_resetAutomaton = new QAction(m_resetIcon, tr("Reset automaton"), this);



    m_fastBackwardBt = new QToolButton(this);
    m_fastForwardBt = new QToolButton(this);
    m_playPauseBt = new QToolButton(this);
    m_saveAutomatonBt = new QToolButton(this);
    m_newAutomatonBt = new QToolButton(this);
    m_openAutomatonBt = new QToolButton(this);
    m_resetBt = new QToolButton(this);

    m_fastBackwardBt->setDefaultAction(m_fastBackward);
    m_fastForwardBt->setDefaultAction(m_fastForward);
    m_playPauseBt->setDefaultAction(m_playPause);
    m_saveAutomatonBt->setDefaultAction(m_saveAutomaton);
    m_newAutomatonBt->setDefaultAction(m_newAutomaton);
    m_openAutomatonBt->setDefaultAction(m_openAutomaton);
    m_resetBt->setDefaultAction(m_resetAutomaton);

    m_fastBackwardBt->setIconSize(QSize(30,30));
    m_fastForwardBt->setIconSize(QSize(30,30));
    m_playPauseBt->setIconSize(QSize(30,30));
    m_saveAutomatonBt->setIconSize(QSize(30,30));
    m_newAutomatonBt->setIconSize(QSize(30,30));
    m_openAutomatonBt->setIconSize(QSize(30,30));
    m_resetBt->setIconSize(QSize(30,30));

    connect(m_openAutomatonBt, SIGNAL(clicked(bool)), this, SLOT(openFile()));
    connect(m_newAutomatonBt, SIGNAL(clicked(bool)), this, SLOT(openCreationWindow()));
    connect(m_saveAutomatonBt, SIGNAL(clicked(bool)), this, SLOT(saveToFile()));
    connect(m_fastForwardBt, SIGNAL(clicked(bool)), this, SLOT(forward()));

}

void MainWindow::createToolBar(){
    m_toolBar = new QToolBar(this);
    QLabel *m_speedLabel = new QLabel(tr("Speed : "),this);
    m_jumpSpeed = new QSpinBox(this);
    m_jumpSpeed->setValue(1);
    m_speedLabel->setFixedWidth(50);
    m_jumpSpeed->setFixedWidth(40);
    m_toolBar->setMovable(false);

    QHBoxLayout *tbLayout = new QHBoxLayout(this);
    tbLayout->addWidget(m_newAutomatonBt, Qt::AlignCenter);
    tbLayout->addWidget(m_openAutomatonBt, Qt::AlignCenter);
    tbLayout->addWidget(m_saveAutomatonBt, Qt::AlignCenter);
    tbLayout->addWidget(m_fastBackwardBt, Qt::AlignCenter);
    tbLayout->addWidget(m_playPauseBt, Qt::AlignCenter);
    tbLayout->addWidget(m_fastForwardBt, Qt::AlignCenter);
    tbLayout->addWidget(m_speedLabel, Qt::AlignCenter);
    tbLayout->addWidget(m_jumpSpeed, Qt::AlignCenter);
    tbLayout->addWidget(m_resetBt, Qt::AlignCenter);


    tbLayout->setAlignment(Qt::AlignCenter);
    QWidget* wrapper = new QWidget(this);
    wrapper->setLayout(tbLayout);
    m_toolBar->addWidget(wrapper);
    addToolBar(m_toolBar);


}

QWidget* MainWindow::createTab(){
    QWidget *tab = new QWidget(this);
    QVBoxLayout *layout = new QVBoxLayout(this);

    QVector<unsigned int> dimensions = m_cellHandlers.last()->getDimensions();
    int boardVSize = 0;
    int boardHSize = 0;
    if(dimensions.size() > 1){
        boardVSize = dimensions[0];
        boardHSize = dimensions[1];
    }
    else{
        boardVSize = 1;
        boardHSize = dimensions[0];
    }

    QTableWidget* board = new QTableWidget(boardVSize, boardHSize, this);
        board->setFixedSize(boardHSize*m_cellSize,boardVSize*m_cellSize);
        //setMinimumSize(m_boardHSize*m_cellSize,100+m_boardVSize*m_cellSize);
        board->horizontalHeader()->setVisible(false);
        board->verticalHeader()->setVisible(false);
        board->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
        board->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
        board->setEditTriggers(QAbstractItemView::NoEditTriggers);
        for(unsigned int col = 0; col < boardHSize; ++col)
            board->setColumnWidth(col, m_cellSize);
        for(unsigned int row = 0; row < boardVSize; ++row) {
            board->setRowHeight(row, m_cellSize);
            for(unsigned int col = 0; col < boardHSize; ++col) {
                board->setItem(row, col, new QTableWidgetItem(""));
                board->item(row, col)->setBackgroundColor("white");
                board->item(row, col)->setTextColor("black");
            }
        }
     QScrollArea *scrollArea = new QScrollArea(this);
     scrollArea->setWidget(board);
     layout->setContentsMargins(0,0,0,0);
     layout->addWidget(scrollArea);
     tab->setLayout(layout);
     return tab;
}

void MainWindow::openFile(){
    QString fileName = QFileDialog::getOpenFileName(this, tr("Open Cell file"), ".",
                                                    tr("Automaton cell files (*.atc)"));
    if(!fileName.isEmpty()){
        m_cellHandlers.append(new CellHandler(fileName));
        std::cout << "m_cellHandlers size :" <<m_cellHandlers.size() << std::endl<<std::flush;
        if(m_tabs == NULL) createTabs();
        m_tabs->addTab(createTab(), "Automaton "+ QString::number(m_cellHandlers.size()));
        updateBoard(m_cellHandlers.size()-1);
    }
}


void MainWindow::saveToFile(){
    if(m_cellHandlers.size() > 0){
        QString fileName = QFileDialog::getSaveFileName(this, tr("Save Automaton"),
                                                        ".", tr("Automaton Cells file (*.atc"));
        m_cellHandlers[m_tabs->currentIndex()]->save(fileName+".atc");

    }
    else{
        QMessageBox msgBox;
        msgBox.critical(0,"Error","Please create or import an Automaton first !");
        msgBox.setFixedSize(500,200);
    }
}
```

```
void MainWindow::openCreationWindow(){
    CreationDialog *window = new CreationDialog(this);
    connect(window, SIGNAL(settingsFilled(QVector<uint>, CellHandler::generationTypes,uint,uint)),
            this, SLOT(setCellHandler(QVector<uint>, CellHandler::generationTypes,uint,uint)));
    window->show();
}

void MainWindow::setCellHandler(const QVector<unsigned int> dimensions,
                                CellHandler::generationTypes type,
                                unsigned int stateMax, unsigned int density){
    CellHandler* newCellHandler = new CellHandler(dimensions, type, stateMax, density);

    if(m_tabs == NULL) createTabs();

    m_cellHandlers.append(newCellHandler);
    std::cout << "m_cellHandlers size :" <<m_cellHandlers.size() << std::endl<<std::flush;
    QWidget* newTab = createTab();
    m_tabs->addTab(newTab, "Automaton "+ QString::number(m_cellHandlers.size()));
    m_tabs->setCurrentWidget(newTab);
    updateBoard(m_cellHandlers.size()-1);

}

void MainWindow::nextState(int n){
    if(m_cellHandlers.size() == 0){
        QMessageBox msgBox;
        msgBox.critical(0,"Error","Please create or import an Automaton first !");
        msgBox.setFixedSize(500,200);
    }
    else{
        for(unsigned int i = 0; i < n; i++) m_cellHandlers[m_tabs->currentIndex()]->nextStates();
        updateBoard(m_tabs->currentIndex());
    }
}

void MainWindow::updateBoard(int index){
    if(m_cellHandlers.size()==0){
        QMessageBox msgBox;
        msgBox.critical(0,"Error","Please create or import an Automaton first !");
        msgBox.setFixedSize(500,200);
    }
    else{

        CellHandler* cellHandler = m_cellHandlers[index];
        QVector<unsigned int> dimensions = cellHandler->getDimensions();
        QTableWidget* board = getBoard(index);
        if(dimensions.size() > 1){
            int i = 0;
            int j = 0;
            for (CellHandler::const_iterator it = CellHandler::const_iterator(cellHandler); it != cellHandler->end() && it.changedDimension() < 2; ++it){
                    if(it.changedDimension() > 0){
                        i = 0;
                        j++;
                        std::cout << std::endl;
                    }
                    board->item(i,j)->setText(QString::number(it->getState()));
                    i++;
            }
        }
        else{
            int i = 0;
            int j = 0;
            for (CellHandler::const_iterator it = CellHandler::const_iterator(cellHandler); it != cellHandler->end() && it.changedDimension() < 1; ++it){
                    board->item(i,j)->setText(QString::number(it->getState()));
                    j++;
            }
        }

    }

}

void MainWindow::forward(){
    nextState(m_jumpSpeed->value());
}

QTableWidget* MainWindow::getBoard(int n){
    return m_tabs->widget(n)->findChild<QTableWidget *>();
}

void MainWindow::createTabs(){
    m_tabs = new QTabWidget(this);
    m_tabs->setMovable(true);
    m_tabs->setTabsClosable(true);
    setCentralWidget(m_tabs);
    connect(m_tabs, SIGNAL(tabCloseRequested(int)), this, SLOT(closeTab(int)));
}

void MainWindow::closeTab(int n){
    m_tabs->setCurrentIndex(n);
    saveToFile();
    m_tabs->removeTab(n);
}
```

## 7.25 mainwindow.h File Reference

```
#include <QMainWindow>
#include <QtWidgets>
#include "cellhandler.h"
#include "creationdialog.h"
```

### Classes

- class MainWindow

  *Simulation window.*

## 7.26 mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QtWidgets>
#include "cellhandler.h"
#include "creationdialog.h"


class MainWindow : public QMainWindow
{
    Q_OBJECT

    QTabWidget *m_tabs; //Tabs for the main window
    QVector<CellHandler *> m_cellHandlers; //QVector containing each tab's cellHandler

    QIcon m_fastBackwardIcon;
    QIcon m_fastForwardIcon;
    QIcon m_playIcon;
    QIcon m_pauseIcon;
    QIcon m_newIcon;
    QIcon m_saveIcon;
    QIcon m_openIcon;
    QIcon m_resetIcon;

    QAction *m_playPause;
    QAction *m_nextState;
    QAction *m_previousState;
    QAction *m_fastForward;
    QAction *m_fastBackward;
    QAction *m_openAutomaton;
    QAction *m_saveAutomaton;
    QAction *m_newAutomaton;
    QAction *m_resetAutomaton;

    QToolButton *m_playPauseBt;
    QToolButton *m_nextStateBt;
    QToolButton *m_previousStateBt;
    QToolButton *m_fastForwardBt;
    QToolButton *m_fastBackwardBt;
    QToolButton *m_openAutomatonBt;
    QToolButton *m_saveAutomatonBt;
    QToolButton *m_newAutomatonBt;
    QToolButton *m_resetBt;


    QSpinBox *m_jumpSpeed;
    QLabel *m_speedLabel;

    QToolBar *m_toolBar;

    unsigned int m_boardHSize = 25;
    unsigned int m_boardVSize = 25;
    unsigned int m_cellSize = 30;

    void createIcons();
    void createActions();
    void createToolBar();
    void createBoard();
    QWidget* createTab();
    void createTabs();


    void updateBoard(int index);
    void nextState(int n);
    QTableWidget* getBoard(int n);


public:
    explicit MainWindow(QWidget *parent = nullptr);


signals:

public slots:
    void openFile();
    void saveToFile();
    void openCreationWindow();
    void setCellHandler(const QVector<unsigned int> dimensions,
                        CellHandler::generationTypes type =
CellHandler::generationTypes::empty,
                        unsigned int stateMax = 1, unsigned int density = 20);
    void forward();
    void closeTab(int n);

};

#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**

| | |
|---|---|
| *min* | Minimal value (included) |
| *max* | 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     Rule(currentStates, finalState)
00023 {
00024 }
00025
00030 bool MatrixRule::matchCell(const Cell *cell) const
00031 {
00032     // Check cell state
00033     if (!m_currentCellPossibleValues.contains(cell->
      getState()))
00034     {
00035         return false;
00036     }
00037
00038     // Check neighbours
00039     bool matched = true;
00040     // Rappel : QMap<relativePosition, possibleStates>
00041     for (QMap<QVector<short>,  QVector<unsigned int> >::const_iterator it =
      m_matrix.begin() ; it != m_matrix.end(); ++it)
00042     {
00043         if (cell->getNeighbour(it.key()) == nullptr) // Border management
00044         {
00045             matched = false;
00046             break;
00047         }
00048         if (! it.value().contains(cell->getNeighbour(it.key())->getState()))
00049         {
00050             matched = false;
00051             break;
00052         }
00053     }
00054
00055     return matched;
00056 }
00057
```

```
void MatrixRule::addNeighbourState(QVector<short> relativePosition, unsigned int matchState)
{
    m_matrix[relativePosition].push_back(matchState);
}

void MatrixRule::addNeighbourState(QVector<short> relativePosition, QVector<unsigned int> matchStates)
{
    for (QVector<unsigned int>::const_iterator it = matchStates.begin(); it != matchStates.end(); ++it)
        m_matrix[relativePosition].push_back(*it);
}

QJsonObject MatrixRule::toJson() const
{
    QJsonObject object(Rule::toJson());

    object.insert("type", QJsonValue("matrix"));

    QJsonArray neighbours;
    for (QMap<QVector<short>,  QVector<unsigned int> >::const_iterator it = m_matrix.begin(); it != m_matrix.end(); ++it)
    {
        QJsonObject aNeighbour;
        QJsonArray relativePosition;
        for (unsigned int i = 0; i < it.key().size(); i++)
        {
            relativePosition.append(QJsonValue((int)it.key().at(i)));
        }
        aNeighbour.insert("relativePosition", relativePosition);

        QJsonArray neighbourStates;
        for (unsigned int i = 0; i < it.value().size(); i++)
        {
            neighbourStates.append(QJsonValue((int)it.value().at(i)));
        }
        aNeighbour.insert("neighbourStates", neighbourStates);

        neighbours.append(aNeighbour);
    }
    object.insert("neighbours", neighbours);

    return object;
}
```

## 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**

| | |
|---|---|
| *min* | Minimal value (included) |
| *max* | Maximal value (included) |

Definition at line 8 of file matrixrule.cpp.

## 7.30 matrixrule.h

```
#ifndef MATRIXRULE_H
#define MATRIXRULE_H

#include <QVector>
#include <QMap>
#include "cell.h"
#include "rule.h"

QVector<unsigned int> fillInterval(unsigned int min, unsigned int max);

class MatrixRule : public Rule
{
    public:
        MatrixRule(unsigned int finalState, QVector<unsigned int> currentStates = QVector<unsigned int>());


        virtual bool matchCell(const Cell* cell) const;
        void addNeighbourState(QVector<short> relativePosition, unsigned int matchState);
        void addNeighbourState(QVector<short> relativePosition, QVector<unsigned int> matchStates);

        QJsonObject toJson() const;
private:

        QMap<QVector<short>,  QVector<unsigned int> > m_matrix;
};

#endif // MATRIXRULE_H
```

## 7.31 neighbourrule.cpp File Reference

```
#include "neighbourrule.h"
```

## 7.32 neighbourrule.cpp

```
#include "neighbourrule.h"

bool NeighbourRule::inInterval(unsigned int matchingNeighbours)const
{
    if(matchingNeighbours >= m_neighbourInterval.first && matchingNeighbours<= m_neighbourInterval.second)
        return true;
    else
        return false;
}

NeighbourRule::NeighbourRule(unsigned int outputState, QVector<unsigned int> currentCellValues, QPair<unsigned int, unsigned int> intervalNbrNeighbour, QSet<unsigned int> neighbourValues) :
        Rule(currentCellValues, outputState), m_neighbourInterval(intervalNbrNeighbour), m_neighbourPossibleValues(neighbourValues)
{
    if (m_neighbourInterval.second == 0)
        throw QString(QObject::tr("Low value of the number of neighbour interval can't be 0"));
    if (m_neighbourInterval.first > m_neighbourInterval.second)
        throw QString(QObject::tr("The interval must be (x,y) with x <= y"));
}

NeighbourRule::~NeighbourRule()
{

}

bool NeighbourRule::matchCell(const Cell *c)const
{
    unsigned int matchingNeighbours = 0;

    if (!m_currentCellPossibleValues.contains(c->getState()))
        return false;

   // QSet<unsigned int> set;
    //QSet<unsigned int> m_neighbourPossibleValues;
    //set<<3<<2<<5<<9;
    QSet<unsigned int>::const_iterator i = m_neighbourPossibleValues.constBegin();
    if (i == m_neighbourPossibleValues.constEnd()) // All possibles values (except 0)
    {
        matchingNeighbours = c->countNeighbours();
    }
    else
    {
        while (i != m_neighbourPossibleValues.constEnd()) {
            //std::cout<<*i;
            matchingNeighbours += c->countNeighbours(*i);
            ++i;
        }
    }
    if(!inInterval(matchingNeighbours))
        return false; //the rule cannot be applied to the cell

    return true; //the rule can be applied to the cell

}

QJsonObject NeighbourRule::toJson() const
{
    QJsonObject object(Rule::toJson());

    object.insert("type", QJsonValue("neighbour"));
    object.insert("neighbourNumberMin", QJsonValue((int)m_neighbourInterval.first));
    object.insert("neighbourNumberMax", QJsonValue((int)m_neighbourInterval.second));

    QJsonArray neighbourState;
    for (QSet<unsigned int>::const_iterator it = m_neighbourPossibleValues.begin(); it != m_neighbourPossibleValues.end(); ++it)
    {
        neighbourState.append(QJsonValue((int)*it));
    }
    object.insert("neighbourStates", neighbourState);

    return object;
}
```

## 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

```
#ifndef NEIGHBOURRULE_H
#define NEIGHBOURRULE_H

#include <QPair>
#include <QSet>
#include "cell.h"
#include "rule.h"

class NeighbourRule : public Rule
{
private:
    QPair<unsigned int , unsigned int> m_neighbourInterval;
    //ATTENTION check that first is lower than second
    QSet<unsigned int> m_neighbourPossibleValues;
    bool inInterval(unsigned int matchingNeighbours)const;
    //bool load(const QJsonObject &json);
public:
    NeighbourRule(unsigned int outputState, QVector<unsigned int> currentCellValues, QPair<unsigned int, unsigned int> intervalNbrNeighbour, QSet<unsigned int> neighbourValues = QSet<unsigned int>());
    ~NeighbourRule();
    bool matchCell(const Cell * c)const;

    virtual QJsonObject toJson() const;
};

#endif // NEIGHBOURRULE_H
```

## 7.35 presentation.md File Reference

## 7.36 presentation.md

```
\page Presentation
# What is AutoCell
The purpose of this project is to create a Cellular Automate Simulator.

\includedoc CellHandler
```

## 7.37 README.md File Reference

## 7.38 README.md

```
\mainpage

To generate the Documentation, go in Documentation directory and run ‘make‘.

It will generate html doc (in ‘output/html/index.html‘) and latex doc (pdf output directely in Documentation directory (‘docPdf.pdf‘).
```

## 7.39 rule.cpp File Reference

```
#include "rule.h"
```

## 7.40 rule.cpp

```
#include "rule.h"

Rule::Rule(QVector<unsigned int> currentCellValues, unsigned int outputState):
        m_currentCellPossibleValues(currentCellValues), m_cellOutputState(outputState)
{

}

QJsonObject Rule::toJson() const
{
    QJsonObject object;
    object.insert("finalState", QJsonValue((int)m_cellOutputState));

    QJsonArray currentStates;
    for (unsigned int i = 0; i < m_currentCellPossibleValues.size(); i++)
    {
        currentStates.append(QJsonValue((int)m_currentCellPossibleValues.at(i)));
    }
    object.insert("currentStates", currentStates);

    return object;
}

unsigned int Rule::getCellOutputState()const
{
        return m_cellOutputState;
}

```

## 7.41 rule.h File Reference

```
#include <QVector>
#include <QJsonObject>
#include <QJsonArray>
#include "cell.h"
```

### Classes

- class Rule

## 7.42 rule.h

```
#ifndef RULE_H
#define RULE_H

#include <QVector>
#include <QJsonObject>
#include <QJsonArray>
#include "cell.h"

class Rule
{
```

```
protected:
    QVector<unsigned int> m_currentCellPossibleValues;
    unsigned int m_cellOutputState;
public:
    Rule(QVector<unsigned int> currentCellValues, unsigned int outputState);

    virtual QJsonObject toJson() const = 0;

    virtual bool matchCell(const Cell * c)const = 0;
    unsigned int getCellOutputState() const;
};

#endif // RULE_H
```

# Index

∼Automate
    Automate, 13
∼AutomateHandler
    AutomateHandler, 18
∼CellHandler
    CellHandler, 29
∼NeighbourRule
    NeighbourRule, 62

addNeighbour
    Cell, 21
addNeighbourState
    MatrixRule, 59
addRule
    Automate, 13
Automate, 11
    ∼Automate, 13
    addRule, 13
    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, 69
automate.h, 72
AutomateHandler, 17
    ∼AutomateHandler, 18
    Automate, 16
    AutomateHandler, 18
    deleteActiveAutomate, 18
    getActiveAutomate, 18
    m_activeAutomate, 19
    operator=, 19
    setActiveAutomate, 19
automatehandler.cpp, 73, 74
    m_activeAutomate, 73
automatehandler.h, 74

begin
    CellHandler, 30

Cell, 20
    addNeighbour, 21
    Cell, 21
    countNeighbours, 21, 22
    forceState, 22
    getNeighbour, 22
    getNeighbours, 23
    getRelativePosition, 23
    getState, 23
    m_neighbours, 24
    m_nextState, 24
    m_state, 25
    setState, 24
    validState, 24
cell.cpp, 75
cell.h, 76
CellHandler, 25
    ∼CellHandler, 29
    begin, 30
    CellHandler, 28, 29
    CellHandler::iteratorT, 43
    const_iterator, 27
    end, 30
    foundNeighbours, 30
    generate, 30
    generationTypes, 27
    getCell, 31
    getDimensions, 31
    getListNeighboursPositions, 31
    getListNeighboursPositionsRecursive, 32
    iterator, 27
    load, 33
    m_cells, 35
    m_dimensions, 35
    nextStates, 33
    positionIncrement, 34
    print, 34
    save, 35
CellHandler::iteratorT< CellHandler_T, Cell_T >, 40
CellHandler::iteratorT
    CellHandler, 43
    changedDimension, 41
    iteratorT, 41
    m_changedDimension, 43
    m_finished, 43
    m_handler, 43
    m_position, 44
    m_zero, 44
    operator!=, 42
    operator∗, 42
    operator++, 42

operator->, 42
cellhandler.cpp, 76, 77
cellhandler.h, 82
changedDimension
CellHandler::iteratorT, 41
closeTab
MainWindow, 47
const_iterator
CellHandler, 27
countNeighbours
Cell, 21, 22
createActions
MainWindow, 47
createBoard
MainWindow, 47
createGenButtons
CreationDialog, 37
createIcons
MainWindow, 47
createTab
MainWindow, 47
createTabs
MainWindow, 48
createToolBar
MainWindow, 48
CreationDialog, 36
createGenButtons, 37
CreationDialog, 37
m_densityBox, 38
m_dimensionsEdit, 38
m_doneBt, 38
m_empGen, 39
m_groupBox, 39
m_randGen, 39
m_stateMaxBox, 39
m_symGen, 39
processSettings, 37
settingsFilled, 38
creationdialog.cpp, 83
creationdialog.h, 85

deleteActiveAutomate
AutomateHandler, 18

end
CellHandler, 30

fillInterval
matrixrule.cpp, 91
matrixrule.h, 93
forceState
Cell, 22
forward
MainWindow, 48
foundNeighbours
CellHandler, 30

generate
CellHandler, 30
generationTypes
CellHandler, 27
getActiveAutomate
AutomateHandler, 18
getBoard
MainWindow, 48
getCell
CellHandler, 31
getCellHandler
Automate, 14
getCellOutputState
Rule, 66
getDimensions
CellHandler, 31
getListNeighboursPositions
CellHandler, 31
getListNeighboursPositionsRecursive
CellHandler, 32
getNeighbour
Cell, 22
getNeighbours
Cell, 23
getRelativePosition
Cell, 23
getRules
Automate, 14
getState
Cell, 23

inInterval
NeighbourRule, 62
iterator
CellHandler, 27
iteratorT
CellHandler::iteratorT, 41

load
CellHandler, 33
loadRules
Automate, 14

m_activeAutomate
AutomateHandler, 19
automatehandler.cpp, 73
m_boardHSize
MainWindow, 50
m_boardVSize
MainWindow, 51
m_cellHandler
Automate, 17
m_cellHandlers
MainWindow, 51
m_cellOutputState
Rule, 67
m_cellSize
MainWindow, 51
m_cells
CellHandler, 35
m_changedDimension

CellHandler::iteratorT, 43
m_currentCellPossibleValues
Rule, 67
m_densityBox
CreationDialog, 38
m_dimensions
CellHandler, 35
m_dimensionsEdit
CreationDialog, 38
m_doneBt
CreationDialog, 38
m_empGen
CreationDialog, 39
m_fastBackward
MainWindow, 51
m_fastBackwardBt
MainWindow, 51
m_fastBackwardIcon
MainWindow, 52
m_fastForward
MainWindow, 52
m_fastForwardBt
MainWindow, 52
m_fastForwardIcon
MainWindow, 52
m_finished
CellHandler::iteratorT, 43
m_groupBox
CreationDialog, 39
m_handler
CellHandler::iteratorT, 43
m_jumpSpeed
MainWindow, 52
m_matrix
MatrixRule, 60
m_neighbourInterval
NeighbourRule, 64
m_neighbourPossibleValues
NeighbourRule, 64
m_neighbours
Cell, 24
m_newAutomaton
MainWindow, 53
m_newAutomatonBt
MainWindow, 53
m_newIcon
MainWindow, 53
m_nextState
Cell, 24
MainWindow, 53
m_nextStateBt
MainWindow, 53
m_openAutomaton
MainWindow, 54
m_openAutomatonBt
MainWindow, 54
m_openIcon
MainWindow, 54
m_pauseIcon
MainWindow, 54
m_playIcon
MainWindow, 54
m_playPause
MainWindow, 55
m_playPauseBt
MainWindow, 55
m_position
CellHandler::iteratorT, 44
m_previousState
MainWindow, 55
m_previousStateBt
MainWindow, 55
m_randGen
CreationDialog, 39
m_resetAutomaton
MainWindow, 55
m_resetBt
MainWindow, 56
m_resetIcon
MainWindow, 56
m_rules
Automate, 17
m_saveAutomaton
MainWindow, 56
m_saveAutomatonBt
MainWindow, 56
m_saveIcon
MainWindow, 56
m_speedLabel
MainWindow, 57
m_state
Cell, 25
m_stateMaxBox
CreationDialog, 39
m_symGen
CreationDialog, 39
m_tabs
MainWindow, 57
m_toolBar
MainWindow, 57
m_zero
CellHandler::iteratorT, 44
main
main.cpp, 86
main.cpp, 85, 86
main, 86
MainWindow, 44
closeTab, 47
createActions, 47
createBoard, 47
createIcons, 47
createTab, 47
createTabs, 48
createToolBar, 48
forward, 48
getBoard, 48

  - m_boardHSize, 50
  - m_boardVSize, 51
  - m_cellHandlers, 51
  - m_cellSize, 51
  - m_fastBackward, 51
  - m_fastBackwardBt, 51
  - m_fastBackwardIcon, 52
  - m_fastForward, 52
  - m_fastForwardBt, 52
  - m_fastForwardIcon, 52
  - m_jumpSpeed, 52
  - m_newAutomaton, 53
  - m_newAutomatonBt, 53
  - m_newIcon, 53
  - m_nextState, 53
  - m_nextStateBt, 53
  - m_openAutomaton, 54
  - m_openAutomatonBt, 54
  - m_openIcon, 54
  - m_pauseIcon, 54
  - m_playIcon, 54
  - m_playPause, 55
  - m_playPauseBt, 55
  - m_previousState, 55
  - m_previousStateBt, 55
  - m_resetAutomaton, 55
  - m_resetBt, 56
  - m_resetIcon, 56
  - m_saveAutomaton, 56
  - m_saveAutomatonBt, 56
  - m_saveIcon, 56
  - m_speedLabel, 57
  - m_tabs, 57
  - m_toolBar, 57
  - MainWindow, 46
  - nextState, 49
  - openCreationWindow, 49
  - openFile, 49
  - saveToFile, 49
  - setCellHandler, 50
  - updateBoard, 50
- mainwindow.cpp, 86
- mainwindow.h, 90
- matchCell
  - MatrixRule, 59
  - NeighbourRule, 63
  - Rule, 66
- MatrixRule, 58
  - addNeighbourState, 59
  - m_matrix, 60
  - matchCell, 59
  - MatrixRule, 58
  - toJson, 60
- matrixrule.cpp, 91, 92
  - fillInterval, 91
- matrixrule.h, 93, 94
  - fillInterval, 93

- NeighbourRule, 61
  - ∼NeighbourRule, 62
  - inInterval, 62
  - m_neighbourInterval, 64
  - m_neighbourPossibleValues, 64
  - matchCell, 63
  - NeighbourRule, 62
  - toJson, 64
- neighbourrule.cpp, 94, 95
- neighbourrule.h, 96
- nextState
  - MainWindow, 49
- nextStates
  - CellHandler, 33

- openCreationWindow
  - MainWindow, 49
- openFile
  - MainWindow, 49
- operator!=
  - CellHandler::iteratorT, 42
- operator∗
  - CellHandler::iteratorT, 42
- operator++
  - CellHandler::iteratorT, 42
- operator->
  - CellHandler::iteratorT, 42
- operator=
  - AutomateHandler, 19

- positionIncrement
  - CellHandler, 34
- presentation.md, 96
- print
  - CellHandler, 34
- processSettings
  - CreationDialog, 37

- README.md, 96
- Rule, 65
  - getCellOutputState, 66
  - m_cellOutputState, 67
  - m_currentCellPossibleValues, 67
  - matchCell, 66
  - Rule, 66
  - toJson, 66
- rule.cpp, 97
- rule.h, 97
- run
  - Automate, 15

- save
  - CellHandler, 35
- saveAll
  - Automate, 15
- saveCells
  - Automate, 15
- saveRules
  - Automate, 15
- saveToFile

MainWindow, 49
setActiveAutomate
AutomateHandler, 19
setCellHandler
MainWindow, 50
setRulePriority
Automate, 16
setState
Cell, 24
settingsFilled
CreationDialog, 38

toJson
MatrixRule, 60
NeighbourRule, 64
Rule, 66

updateBoard
MainWindow, 50

validState
Cell, 24