AutoCell

Generated by Doxygen 1.8.14

Contents

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:

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Class Documentation

6.1 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.

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.

6.1.1 Detailed Description

Contains the state, the next state and the neighbours.

Definition at line 10 of file cell.h.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 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.

6.1.3 Member Function Documentation

6.1.3.1 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.

6.1.3.2 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.

6.1.3.3 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.

6.1.3.4 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.

6.1.3.5 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 91 of file cell.cpp.

Referenced by CellHandler::foundNeighbours().

6.1.3.6 getState()

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

Access current cell state.

Definition at line 48 of file cell.cpp.

References m_state.

6.1.3.7 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.

6.1.3.8 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.

6.1.4 Member Data Documentation

6.1.4.1 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 30 of file cell.h.

Referenced by addNeighbour(), getNeighbour(), and getNeighbours().

6.1.4.2 m_nextState

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

Temporary state, before validation.

Definition at line 28 of file cell.h.

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

6.1.4.3 m_state

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

Current state.

Definition at line 27 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

6.2 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 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.

6.2.1 Detailed Description

Cell container and cell generator.

Generate cells from a json file.

Definition at line 20 of file cellhandler.h.

6.2.2 Member Typedef Documentation

6.2.2.1 const_iterator

```
typedef iteratorT<const CellHandler, const Cell> CellHandler::const_iterator
```

Definition at line 64 of file cellhandler.h.

6.2.2.2 iterator

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

Definition at line 65 of file cellhandler.h.

6.2.3 Member Enumeration Documentation

6.2.3.1 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.

6.2.4 Constructor & Destructor Documentation

6.2.4.1 CellHandler() [1/2]

```
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().

6.2.4.2 CellHandler() [2/2]

```
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 65 of file cellhandler.cpp.

References empty, foundNeighbours(), generate(), m_cells, m_dimensions, and positionIncrement().

6.2.4.3 ~CellHandler()

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

Destroys all cells in the CellHandler.

Definition at line 97 of file cellhandler.cpp.

References m_cells.

6.2.5 Member Function Documentation

6.2.5.1 begin() [1/2]

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

Give the iterator which corresponds to the current CellHandler.

Definition at line 262 of file cellhandler.cpp.

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

6.2.5.2 begin() [2/2]

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

Give the iterator which corresponds to the current CellHandler.

Definition at line 255 of file cellhandler.cpp.

6.2.5.3 end()

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

End condition of the iterator.

See iterator::operator!=(bool finished) for further information.

Definition at line 271 of file cellhandler.cpp.

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

6.2.5.4 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 364 of file cellhandler.cpp.

References getListNeighboursPositions(), Cell::getRelativePosition(), m_cells, m_dimensions, and positionIncrement().

Referenced by CellHandler().

6.2.5.5 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 176 of file cellhandler.cpp.

References m_cells, m_dimensions, positionIncrement(), random, and symetric.

Referenced by CellHandler().

6.2.5.6 getCell()

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

Access the cell to the given position.

Definition at line 107 of file cellhandler.cpp.

References m_cells.

6.2.5.7 getDimensions()

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

Accessor of m_dimensions.

Definition at line 114 of file cellhandler.cpp.

References m_dimensions.

Referenced by MainWindow::openFile().

6.2.5.8 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 423 of file cellhandler.cpp.

References getListNeighboursPositionsRecursive().

Referenced by foundNeighbours().

6.2.5.9 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 464 of file cellhandler.cpp.

References m_dimensions.

Referenced by getListNeighboursPositions().

6.2.5.10 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 306 of file cellhandler.cpp.

References m_cells, m_dimensions, and positionIncrement().

Referenced by CellHandler().

6.2.5.11 nextStates()

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

Valid the state of all cells.

Definition at line 121 of file cellhandler.cpp.

References m_cells.

Referenced by MainWindow::nextState().

6.2.5.12 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 394 of file cellhandler.cpp.

References m_dimensions.

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

6.2.5.13 print()

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

Print in the given stream the CellHandler.

Parameters

stream	Stream to print into

Definition at line 241 of file cellhandler.cpp.

References begin(), and end().

6.2.5.14 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 136 of file cellhandler.cpp.

References begin(), end(), and m_dimensions.

Referenced by MainWindow::saveToFile().

6.2.6 Member Data Documentation

6.2.6.1 m_cells

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

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

Definition at line 100 of file cellhandler.h.

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

6.2.6.2 m_dimensions

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

Vector of x dimensions.

Definition at line 99 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:

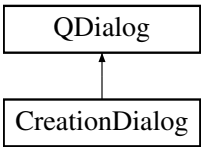
- cellhandler.h
- cellhandler.cpp

6.3 CreationDialog Class Reference

Automaton creation dialog box.

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

Inheritance diagram for CreationDialog:



Public Slots

- void processSettings ()

Signals

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

Public Member Functions

- CreationDialog (QWidget *parent=0)

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

6.3.1 Detailed Description

Automaton creation dialog box.

Allow the user to input settings to create an automaton

Definition at line 13 of file creationdialog.h.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 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().

6.3.3 Member Function Documentation

6.3.3.1 createGenButtons()

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

Creates radio buttons to select cell generation type.

Validates user settings and sends them to MainWindow.

Definition at line 49 of file creationdialog.cpp.

References m_empGen, m_groupBox, m_randGen, and m_symGen.

Referenced by CreationDialog().

6.3.3.2 processSettings

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

Definition at line 70 of file creationdialog.cpp.

References m_densityBox, m_dimensionsEdit, m_randGen, m_stateMaxBox, m_symGen, and settingsFilled().

Referenced by CreationDialog().

6.3.3.3 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().

6.3.4 Member Data Documentation

6.3.4.1 m_densityBox

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

Definition at line 30 of file creationdialog.h.

Referenced by CreationDialog(), and processSettings().

6.3.4.2 m_dimensionsEdit

```
QLineEdit* CreationDialog::m_dimensionsEdit  [private]
```

Definition at line 29 of file creationdialog.h.

Referenced by CreationDialog(), and processSettings().

6.3.4.3 m_doneBt

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

Definition at line 32 of file creationdialog.h.

Referenced by CreationDialog().

6.3.4.4 m_empGen

```
QRadioButton* CreationDialog::m_empGen  [private]
```

Definition at line 35 of file creationdialog.h.

Referenced by createGenButtons().

6.3.4.5 m_groupBox

```
QGroupBox* CreationDialog::m_groupBox  [private]
```

Definition at line 34 of file creationdialog.h.

Referenced by createGenButtons().

6.3.4.6 m_randGen

```
QRadioButton* CreationDialog::m_randGen  [private]
```

Definition at line 36 of file creationdialog.h.

Referenced by createGenButtons(), and processSettings().

6.3.4.7 m_stateMaxBox

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

Definition at line 31 of file creationdialog.h.

Referenced by CreationDialog(), and processSettings().

6.3.4.8 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

6.4 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

6.4.1 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.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 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 504 of file cellhandler.cpp.

References CellHandler::iteratorT< CellHandler_T, Cell_T >::m_position, and CellHandler::iteratorT< CellHandler_T, Cell_T >::m_z

6.4.3 Member Function Documentation

6.4.3.1 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 566 of file cellhandler.cpp.

6.4.3.2 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.

6.4.3.3 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 555 of file cellhandler.cpp.

6.4.3.4 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 518 of file cellhandler.cpp.

6.4.3.5 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 546 of file cellhandler.cpp.

6.4.4 Friends And Related Function Documentation

6.4.4.1 CellHandler

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

Definition at line 43 of file cellhandler.h.

6.4.5 Member Data Documentation

6.4.5.1 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.

6.4.5.2 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!=().

6.4.5.3 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.

6.4.5.4 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().

6.4.5.5 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:

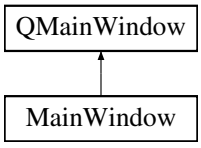
- cellhandler.h
- cellhandler.cpp

6.5 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.

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 ()

  Creates the Automaton board.
- void updateBoard ()

  Updates cells on the board with the cellHandler's cells states.
- void nextState (int n)

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

Private Attributes

- CellHandler * m_cellHandler
- 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
- QTableWidget * m_Board

  Toolbar containing the buttons.
- unsigned int m_boardHSize = 25

  Board showing the automaton's current state.
- unsigned int m_boardVSize = 25
- unsigned int m_cellSize = 30

6.5.1 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.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 MainWindow()

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

Definition at line 3 of file mainwindow.cpp.

References createActions(), createBoard(), createIcons(), createToolBar(), and m_cellHandler.

6.5.3 Member Function Documentation

6.5.3.1 createActions()

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

Creates and connects QActions and associated buttons for the MainWindow.

Definition at line 52 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().

6.5.3.2 createBoard()

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

Creates the Automaton board.

Definition at line 132 of file mainwindow.cpp.

References m_Board, m_boardHSize, m_boardVSize, and m_cellSize.

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

6.5.3.3 createIcons()

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

Creates Icons for the MainWindow.

Definition at line 22 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().

6.5.3.4 createToolBar()

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

Creates the toolBar for the MainWindow.

Definition at line 98 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().

6.5.3.5 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 276 of file mainwindow.cpp.

References m_jumpSpeed, and nextState().

Referenced by createActions().

6.5.3.6 nextState()

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

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

Definition at line 234 of file mainwindow.cpp.

References m_cellHandler, CellHandler::nextStates(), and updateBoard().

Referenced by forward().

6.5.3.7 openCreationWindow

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

Opens the automaton creation window.

Definition at line 201 of file mainwindow.cpp.

References setCellHandler().

Referenced by createActions().

6.5.3.8 openFile

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

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

Definition at line 160 of file mainwindow.cpp.

References createBoard(), CellHandler::getDimensions(), m_boardHSize, m_boardVSize, m_cellHandler, and updateBoard().

Referenced by createActions().

6.5.3.9 saveToFile

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

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

Definition at line 183 of file mainwindow.cpp.

References m_cellHandler, and CellHandler::save().

Referenced by createActions().

6.5.3.10 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 214 of file mainwindow.cpp.

References createBoard(), m_boardHSize, m_boardVSize, m_cellHandler, and updateBoard().

Referenced by openCreationWindow().

6.5.3.11 updateBoard()

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

Updates cells on the board with the cellHandler's cells states.

Definition at line 250 of file mainwindow.cpp.

References CellHandler::begin(), CellHandler::end(), m_Board, and m_cellHandler.

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

6.5.4 Member Data Documentation

6.5.4.1 m_Board

```
QTableWidget* MainWindow::m_Board  [private]
```

Toolbar containing the buttons.

Definition at line 60 of file mainwindow.h.

Referenced by createBoard(), and updateBoard().

6.5.4.2 m_boardHSize

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

Board showing the automaton's current state.

Board size settings

Definition at line 63 of file mainwindow.h.

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

6.5.4.3 m_boardVSize

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

Definition at line 64 of file mainwindow.h.

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

6.5.4.4 m_cellHandler

```
CellHandler* MainWindow::m_cellHandler  [private]
```

Definition at line 20 of file mainwindow.h.

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

6.5.4.5 m_cellSize

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

Definition at line 65 of file mainwindow.h.

Referenced by createBoard().

6.5.4.6 m_fastBackward

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

Definition at line 37 of file mainwindow.h.

Referenced by createActions().

6.5.4.7 m_fastBackwardBt

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

Definition at line 48 of file mainwindow.h.

Referenced by createActions(), and createToolBar().

6.5.4.8 m_fastBackwardIcon

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

Icons.

Definition at line 23 of file mainwindow.h.

Referenced by createActions(), and createIcons().

6.5.4.9 m_fastForward

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

Definition at line 36 of file mainwindow.h.

Referenced by createActions().

6.5.4.10 m_fastForwardBt

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

Definition at line 47 of file mainwindow.h.

Referenced by createActions(), and createToolBar().

6.5.4.11 m_fastForwardIcon

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

Definition at line 24 of file mainwindow.h.

Referenced by createActions(), and createIcons().

6.5.4.12 m_jumpSpeed

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

Definition at line 55 of file mainwindow.h.

Referenced by createToolBar(), and forward().

6.5.4.13 m_newAutomaton

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

Definition at line 40 of file mainwindow.h.

Referenced by createActions().

6.5.4.14 m_newAutomatonBt

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

Definition at line 51 of file mainwindow.h.

Referenced by createActions(), and createToolBar().

6.5.4.15 m_newIcon

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

Definition at line 27 of file mainwindow.h.

Referenced by createActions(), and createIcons().

6.5.4.16 m_nextState

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

Definition at line 34 of file mainwindow.h.

6.5.4.17 m_nextStateBt

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

Definition at line 45 of file mainwindow.h.

6.5.4.18 m_openAutomaton

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

Definition at line 38 of file mainwindow.h.

Referenced by createActions().

6.5.4.19 m_openAutomatonBt

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

Definition at line 49 of file mainwindow.h.

Referenced by createActions(), and createToolBar().

6.5.4.20 m_openIcon

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

Definition at line 29 of file mainwindow.h.

Referenced by createActions(), and createIcons().

6.5.4.21 m_pauseIcon

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

Definition at line 26 of file mainwindow.h.

Referenced by createIcons().

6.5.4.22 m_playIcon

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

Definition at line 25 of file mainwindow.h.

Referenced by createActions(), and createIcons().

6.5.4.23 m_playPause

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

Actions.

Definition at line 33 of file mainwindow.h.

Referenced by createActions().

6.5.4.24 m_playPauseBt

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

Buttons.

Definition at line 44 of file mainwindow.h.

Referenced by createActions(), and createToolBar().

6.5.4.25 m_previousState

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

Definition at line 35 of file mainwindow.h.

6.5.4.26 m_previousStateBt

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

Definition at line 46 of file mainwindow.h.

6.5.4.27 m_resetAutomaton

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

Definition at line 41 of file mainwindow.h.

Referenced by createActions().

6.5.4.28 m_resetBt

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

Definition at line 52 of file mainwindow.h.

Referenced by createActions(), and createToolBar().

6.5.4.29 m_resetIcon

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

Definition at line 30 of file mainwindow.h.

Referenced by createActions(), and createIcons().

6.5.4.30 m_saveAutomaton

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

Definition at line 39 of file mainwindow.h.

Referenced by createActions().

6.5.4.31 m_saveAutomatonBt

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

Definition at line 50 of file mainwindow.h.

Referenced by createActions(), and createToolBar().

6.5.4.32 m_saveIcon

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

Definition at line 28 of file mainwindow.h.

Referenced by createActions(), and createIcons().

6.5.4.33 m_speedLabel

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

Simulation speed input.

Definition at line 56 of file mainwindow.h.

Referenced by createToolBar().

6.5.4.34 m_toolBar

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

Definition at line 58 of file mainwindow.h.

Referenced by createToolBar().

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

- mainwindow.h
- mainwindow.cpp

Chapter 7

File Documentation

7.1 cell.cpp File Reference

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

7.2 cell.cpp

```
00001 #include "cell.h"
00002
00007 Cell::Cell(unsigned int state):
00008     m_state(state), m_nextState(state)
00009 {
00010
00011 }
00012
00020 void Cell::setState(unsigned int state)
00021 {
00022     m_nextState = state;
00023 }
00024
00030 void Cell::validState()
00031 {
00032     m_state = m_nextState;
00033 }
00034
00041 void Cell::forceState(unsigned int state)
00042 {
00043     m_state = m_nextState = state;
00044 }
00045
00048 unsigned int Cell::getState() const
00049 {
00050     return m_state;
00051 }
00052
00060 bool Cell::addNeighbour(const Cell* neighbour, const QVector<short> relativePosition)
00061 {
00062     if (m_neighbours.count(relativePosition))
00063         return false;
00064
00065     m_neighbours.insert(relativePosition, neighbour);
00066     return true;
00067 }
00068
00073 QMap<QVector<short>, const Cell *> Cell::getNeighbours() const
00074 {
00075     return m_neighbours;
00076 }
00077
00080 const Cell *Cell::getNeighbour(QVector<short> relativePosition) const
00081 {
00082     return m_neighbours.value(relativePosition, nullptr);
```

```
00083 }
00084
00091 QVector<short> Cell::getRelativePosition(const QVector<unsigned int> cellPosition,
       const QVector<unsigned int> neighbourPosition)
00092 {
00093     if (cellPosition.size() != neighbourPosition.size())
00094     {
00095         throw QString(QObject::tr("Different size of position vectors"));
00096     }
00097     QVector<short> relativePosition;
00098     for (short i = 0; i < cellPosition.size(); i++)
00099         relativePosition.push_back(neighbourPosition.at(i) - cellPosition.at(i));
00100
00101     return relativePosition;
00102 }
```

7.3 cell.h File Reference

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

Classes

- class Cell

  Contains the state, the next state and the neighbours.

7.4 cell.h

```
00001 #ifndef CELL_H
00002 #define CELL_H
00003
00004 #include <QVector>
00005 #include <QDebug>
00006
00010 class Cell
00011 {
00012 public:
00013     Cell(unsigned int state = 0);
00014
00015     void setState(unsigned int state);
00016     void validState();
00017     void forceState(unsigned int state);
00018     unsigned int getState() const;
00019
00020     bool addNeighbour(const Cell* neighbour, const QVector<short> relativePosition);
00021     QMap<QVector<short>, const Cell*> getNeighbours() const;
00022     const Cell* getNeighbour(QVector<short> relativePosition) const;
00023
00024     static QVector<short> getRelativePosition(const QVector<unsigned int> cellPosition,
      const QVector<unsigned int> neighbourPosition);
00025
00026 private:
00027     unsigned int m_state;
00028     unsigned int m_nextState;
00029
00030     QMap<QVector<short>, const Cell*> m_neighbours;
00031 };
00032
00033 #endif // CELL_H
```

7.5 cellhandler.cpp File Reference

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

7.6 cellhandler.cpp

```
00001 #include <iostream>
00002 #include "cellhandler.h"
00003
00025 CellHandler::CellHandler(const QString filename)
00026 {
00027     QFile loadFile(filename);
00028     if (!loadFile.open(QIODevice::ReadOnly | QIODevice::Text)) {
00029         qWarning("Couldn't open given file.");
00030         throw QString(QObject::tr("Couldn't open given file"));
00031     }
00032
00033     QJsonParseError parseErr;
00034     QJsonDocument loadDoc(QJsonDocument::fromJson(loadFile.readAll(), &parseErr));
00035
00036
00037
00038     if (loadDoc.isNull() || loadDoc.isEmpty()) {
00039         qWarning() << "Could not read data : ";
00040         qWarning() << parseErr.errorString();
00041         throw QString(parseErr.errorString());
00042     }
00043
00044     // Loadding of the json file
00045     if (!load(loadDoc.object()))
00046     {
00047         qWarning("File not valid");
00048         throw QString(QObject::tr("File not valid"));
00049     }
00050
00051     foundNeighbours();
00052
00053
00054 }
00055
00065 CellHandler::CellHandler(const QVector<unsigned int> dimensions,
      generationTypes type, unsigned int stateMax, unsigned int density)
00066 {
00067     m_dimensions = dimensions;
00068     QVector<unsigned int> position;
00069     unsigned int size = 1;
00070
00071     // Set position vector to 0
00072
00073     for (unsigned short i = 0; i < m_dimensions.size(); i++)
00074     {
00075         position.push_back(0);
00076         size *= m_dimensions.at(i);
00077     }
00078
00079
00080     // Creation of cells
00081     for (unsigned int j = 0; j < size; j++)
00082     {
00083         m_cells.insert(position, new Cell(0));
00084
00085         positionIncrement(position);
00086     }
00087
00088     foundNeighbours();
00089
00090     if (type != empty)
00091         generate(type, stateMax, density);
00092
00093 }
00094
00097 CellHandler::~CellHandler()
00098 {
00099     for (QMap<QVector<unsigned int>, Cell* >::iterator it = m_cells.begin(); it !=
      m_cells.end(); ++it)
00100     {
00101         delete it.value();
00102     }
00103 }
00104
00107 Cell *CellHandler::getCell(const QVector<unsigned int> position) const
00108 {
00109     return m_cells.value(position);
00110 }
00111
00114 QVector<unsigned int> CellHandler::getDimensions() const
00115 {
00116     return m_dimensions;
00117 }
00118
```

```
00121 void CellHandler::nextStates() const
00122 {
00123     for (QMap<QVector<unsigned int>, Cell* >::const_iterator it =
      m_cells.begin(); it != m_cells.end(); ++it)
00124     {
00125         it.value()->validState();
00126     }
00127 }
00128
00136 bool CellHandler::save(QString filename) const
00137 {
00138     QFile saveFile(filename);
00139     if (!saveFile.open(QIODevice::WriteOnly)) {
00140         qWarning("Couldn't create or open given file.");
00141         throw QString(QObject::tr("Couldn't create or open given file"));
00142     }
00143
00144     QJsonObject json;
00145     QString stringDimension;
00146     // Creation of the dimension string
00147     for (int i = 0; i < m_dimensions.size(); i++)
00148     {
00149         if (i != 0)
00150             stringDimension.push_back("x");
00151         stringDimension.push_back(QString::number(m_dimensions.at(i)));
00152     }
00153     json["dimensions"] = QJsonValue(stringDimension);
00154
00155     QJsonArray cells;
00156     for (CellHandler::const_iterator it = begin(); it !=
      end(); ++it)
00157     {
00158         cells.append(QJsonValue((int)it->getState()));
00159     }
00160     json["cells"] = cells;
00161
00162
00163     QJsonDocument saveDoc(json);
00164     saveFile.write(saveDoc.toJson());
00165
00166     saveFile.close();
00167     return true;
00168 }
00169
00176 void CellHandler::generate(CellHandler::generationTypes
      type, unsigned int stateMax, unsigned short density)
00177 {
00178     if (type == random)
00179     {
00180         QVector<unsigned int> position;
00181         for (unsigned short i = 0; i < m_dimensions.size(); i++)
00182         {
00183             position.push_back(0);
00184         }
00185         QRandomGenerator generator((float)qrand()*(float)time_t()/RAND_MAX);
00186         for (int j = 0; j < m_cells.size(); j++)
00187         {
00188             unsigned int state = 0;
00189             // 0 have (1-density)% of chance of being generate
00190             if (generator.generateDouble()*100.0 < density)
00191                 state = (float)(generator.generateDouble()*stateMax) +1;
00192             if (state > stateMax)
00193                 state = stateMax;
00194             m_cells.value(position)->forceState(state);
00195
00196             positionIncrement(position);
00197         }
00198     }
00199     else if (type == symetric)
00200     {
00201         QVector<unsigned int> position;
00202         for (short i = 0; i < m_dimensions.size(); i++)
00203         {
00204             position.push_back(0);
00205         }
00206
00207         QRandomGenerator generator((float)qrand()*(float)time_t()/RAND_MAX);
00208         QVector<unsigned int> savedStates;
00209         for (int j = 0; j < m_cells.size(); j++)
00210         {
00211             if (j % m_dimensions.at(0) == 0)
00212                 savedStates.clear();
00213             if (j % m_dimensions.at(0) < (m_dimensions.at(0)+1) / 2)
00214             {
00215                 unsigned int state = 0;
00216                 // 0 have (1-density)% of chance of being generate
00217                 if (generator.generateDouble()*100.0 < density)
```

```
00218                     state = (float)(generator.generateDouble()*stateMax) +1;
00219                 if (state > stateMax)
00220                     state = stateMax;
00221                 savedStates.push_back(state);
00222                 m_cells.value(position)->forceState(state);
00223             }
00224             else
00225             {
00226                 unsigned int i = savedStates.size() - (j % m_dimensions.at(0) - (
      m_dimensions.at(0)-1)/2 + (m_dimensions.at(0) % 2 == 0 ? 0 : 1));
00227                 m_cells.value(position)->forceState(savedStates.at(i));
00228             }
00229             positionIncrement(position);
00230 
00231 
00232         }
00233 
00234     }
00235 }
00236 
00241 void CellHandler::print(std::ostream &stream) const
00242 {
00243     for (const_iterator it = begin(); it != end(); ++it)
00244     {
00245         for (unsigned int d = 0; d < it.changedDimension(); d++)
00246             stream << std::endl;
00247         stream << it->getState() << " ";
00248 
00249     }
00250 
00251 }
00252 
00255 CellHandler::iterator CellHandler::begin()
00256 {
00257     return iterator(this);
00258 }
00259 
00262 CellHandler::const_iterator CellHandler::begin() const
00263 {
00264     return const_iterator(this);
00265 }
00266 
00271 bool CellHandler::end() const
00272 {
00273     return true;
00274 }
00275 
00306 bool CellHandler::load(const QJsonObject &json)
00307 {
00308     if (!json.contains("dimensions") || !json["dimensions"].isString())
00309         return false;
00310 
00311     // RegExp to validate dimensions field format : "10x10"
00312     QRegExpValidator dimensionValidator(QRegExp("([0-9]*x?)*"));
00313     QString stringDimensions = json["dimensions"].toString();
00314     int pos= 0;
00315     if (dimensionValidator.validate(stringDimensions, pos) != QRegExpValidator::Acceptable)
00316         return false;
00317 
00318     // Split of dimensions field : "10x10" => "10", "10"
00319     QRegExp rx("x");
00320     QStringList list = json["dimensions"].toString().split(rx, QString::SkipEmptyParts);
00321 
00322     int product = 1;
00323     // Dimensions construction
00324     for (int i = 0; i < list.size(); i++)
00325     {
00326         product = product * list.at(i).toInt();
00327         m_dimensions.push_back(list.at(i).toInt());
00328     }
00329     if (!json.contains("cells") || !json["cells"].isArray())
00330         return false;
00331 
00332     QJsonArray cells = json["cells"].toArray();
00333     if (cells.size() != product)
00334         return false;
00335 
00336     QVector<unsigned int> position;
00337     // Set position vector to 0
00338     for (unsigned short i = 0; i < m_dimensions.size(); i++)
00339     {
00340         position.push_back(0);
00341     }
00342 
00343     // Creation of cells
00344     for (int j = 0; j < cells.size(); j++)
00345     {
```

```
00346         if (!cells.at(j).isDouble())
00347             return false;
00348         if (cells.at(j).toDouble() < 0)
00349             return false;
00350         m_cells.insert(position, new Cell(cells.at(j).toDouble()));
00351 
00352         positionIncrement(position);
00353     }
00354 
00355     return true;
00356 
00357 }
00358 
00364 void CellHandler::foundNeighbours()
00365 {
00366     QVector<unsigned int> currentPosition;
00367     // Set position vector to 0
00368     for (unsigned short i = 0; i < m_dimensions.size(); i++)
00369     {
00370         currentPosition.push_back(0);
00371     }
00372     // Modification of all the cells
00373     for (int j = 0; j < m_cells.size(); j++)
00374     {
00375         // Get the list of the neighbours positions
00376         // This function is recursive
00377         QVector<QVector<unsigned int> > listPosition(getListNeighboursPositions(
    currentPosition));
00378 
00379         // Adding neighbours
00380         for (int i = 0; i < listPosition.size(); i++)
00381             m_cells.value(currentPosition)->addNeighbour(m_cells.value(listPosition.at(i)),
    Cell::getRelativePosition(currentPosition, listPosition.at(i)));
00382         positionIncrement(currentPosition);
00383     }
00384 
00385 }
00386 
00394 void CellHandler::positionIncrement(QVector<unsigned int> &pos, unsigned int
    value) const
00395 {
00396     pos.replace(0, pos.at(0) + value); // adding the value to the first digit
00397 
00398     // Carry management
00399     for (unsigned short i = 0; i < m_dimensions.size(); i++)
00400     {
00401         if (pos.at(i) >= m_dimensions.at(i) && pos.at(i) <
    m_dimensions.at(i)*2)
00402         {
00403             pos.replace(i, 0);
00404             if (i + 1 != m_dimensions.size())
00405                 pos.replace(i+1, pos.at(i+1)+1);
00406         }
00407         else if (pos.at(i) >= m_dimensions.at(i))
00408         {
00409             pos.replace(i, pos.at(i) - m_dimensions.at(i));
00410             if (i + 1 != m_dimensions.size())
00411                 pos.replace(i+1, pos.at(i+1)+1);
00412             i--;
00413         }
00414 
00415     }
00416 }
00417 
00423 QVector<QVector<unsigned int> >& CellHandler::getListNeighboursPositions
    (const QVector<unsigned int> position) const
00424 {
00425     QVector<QVector<unsigned int> > *list = getListNeighboursPositionsRecursive
    (position, position.size(), position);
00426     // We remove the position of the cell
00427     list->removeAll(position);
00428     return *list;
00429 }
00430 
00464 QVector<QVector<unsigned int> >*
    CellHandler::getListNeighboursPositionsRecursive(const
    QVector<unsigned int> position, unsigned int dimension, QVector<unsigned int> lastAdd) const
00465 {
00466     if (dimension == 0) // Stop condition
00467     {
00468         QVector<QVector<unsigned int> > *list = new QVector<QVector<unsigned int> >;
00469         return list;
00470     }
00471     QVector<QVector<unsigned int> > *listPositions = new QVector<QVector<unsigned int> >;
00472 
00473     QVector<unsigned int> modifiedPosition(lastAdd);
00474 
```

```
00475     // "x_d - 1" tree
00476     if (modifiedPosition.at(dimension-1) != 0) // Avoid "negative" position
00477         modifiedPosition.replace(dimension-1, position.at(dimension-1) - 1);
00478     listPositions->append(*getListNeighboursPositionsRecursive(position,
      dimension -1, modifiedPosition));
00479     if (!listPositions->count(modifiedPosition))
00480         listPositions->push_back(modifiedPosition);
00481
00482     // "x_d" tree
00483     modifiedPosition.replace(dimension-1, position.at(dimension-1));
00484     listPositions->append(*getListNeighboursPositionsRecursive(position,
      dimension -1, modifiedPosition));
00485     if (!listPositions->count(modifiedPosition))
00486         listPositions->push_back(modifiedPosition);
00487
00488     // "x_d + 1" tree
00489     if (modifiedPosition.at(dimension -1) + 1 < m_dimensions.at(dimension-1)) // Avoid position
      out of the cell space
00490         modifiedPosition.replace(dimension-1, position.at(dimension-1) +1);
00491     listPositions->append(*getListNeighboursPositionsRecursive(position,
      dimension -1, modifiedPosition));
00492     if (!listPositions->count(modifiedPosition))
00493         listPositions->push_back(modifiedPosition);
00494
00495     return listPositions;
00496
00497 }
00498
00503 template<typename CellHandler_T, typename Cell_T>
00504 CellHandler::iteratorT<CellHandler_T,Cell_T>::iteratorT
     (CellHandler_T *handler):
00505         m_handler(handler), m_changedDimension(0)
00506 {
00507     // Initialisation of m_position
00508     for (unsigned short i = 0; i < handler->m_dimensions.size(); i++)
00509     {
00510         m_position.push_back(0);
00511     }
00512     m_zero = m_position;
00513 }
00514
00517 template<typename CellHandler_T, typename Cell_T>
00518 CellHandler::iteratorT<CellHandler_T,Cell_T> &
     CellHandler::iteratorT<CellHandler_T,Cell_T>::operator++
     ()
00519 {
00520     m_position.replace(0, m_position.at(0) + 1); // adding the value to the first digit
00521
00522     m_changedDimension = 0;
00523     // Carry management
00524     for (unsigned short i = 0; i < m_handler->m_dimensions.size(); i++)
00525     {
00526         if (m_position.at(i) >= m_handler->m_dimensions.at(i))
00527         {
00528             m_position.replace(i, 0);
00529             m_changedDimension++;
00530             if (i + 1 != m_handler->m_dimensions.size())
00531                 m_position.replace(i+1, m_position.at(i+1)+1);
00532         }
00533
00534     }
00535     // If we return to zero, we have finished
00536     if (m_position == m_zero)
00537         m_finished = true;
00538
00539     return *this;
00540
00541 }
00542
00545 template<typename CellHandler_T, typename Cell_T>
00546 Cell_T* CellHandler::iteratorT<CellHandler_T,Cell_T>::operator->
     () const
00547 {
00548     return m_handler->m_cells.value(m_position);
00549 }
00550
00551
00554 template<typename CellHandler_T, typename Cell_T>
00555 Cell_T *CellHandler::iteratorT<CellHandler_T,Cell_T>::operator*
     () const
00556 {
00557     return m_handler->m_cells.value(m_position);
00558 }
00559
00565 template<typename CellHandler_T, typename Cell_T>
00566 unsigned int CellHandler::iteratorT<CellHandler_T,Cell_T>::changedDimension
     () const
```

```
00567 {
00568     return m_changedDimension;
00569 }
00570
```

7.7 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.

7.8 cellhandler.h

```
00001 #ifndef CELLHANDLER_H
00002 #define CELLHANDLER_H
00003
00004 #include <QString>
00005 #include <QFile>
00006 #include <QJsonDocument>
00007 #include <QtWidgets>
00008 #include <QMap>
00009 #include <QRegExpValidator>
00010 #include <QDebug>
00011
00012 #include "cell.h"
00013
00014
00015
00020 class CellHandler
00021 {
00022
00040     template <typename CellHandler_T, typename Cell_T>
00041     class iteratorT
00042     {
00043         friend class CellHandler;
00044     public:
00045         iteratorT(CellHandler_T* handler);
00046
00047         iteratorT& operator++();
00048         Cell_T* operator->() const;
00049         Cell_T* operator*() const;
00050
00051         bool operator!=(bool finished) const { return (m_finished != finished); }
00052         unsigned int changedDimension() const;
00053
00054
00055
00056     private:
00057         CellHandler_T *m_handler;
00058         QVector<unsigned int> m_position;
00059         bool m_finished = false;
00060         QVector<unsigned int> m_zero;
```

```
00061         unsigned int m_changedDimension;
00062     };
00063 public:
00064     typedef iteratorT<const CellHandler, const Cell>
      const_iterator;
00065     typedef iteratorT<CellHandler, Cell> iterator;
00066
00069     enum generationTypes {
00070         empty,
00071         random,
00072         symetric
00073     };
00074
00075     CellHandler(const QString filename);
00076     CellHandler(const QVector<unsigned int> dimensions,
      generationTypes type = empty, unsigned int stateMax = 1, unsigned int density = 20);
00077     virtual ~CellHandler();
00078
00079     Cell* getCell(const QVector<unsigned int> position) const;
00080     QVector<unsigned int> getDimensions() const;
00081     void nextStates() const;
00082
00083     bool save(QString filename) const;
00084
00085     void generate(generationTypes type, unsigned int stateMax = 1, unsigned short
      density = 50);
00086     void print(std::ostream &stream) const;
00087
00088     const_iterator begin() const;
00089     iterator begin();
00090     bool end() const;
00091
00092 private:
00093     bool load(const QJsonObject &json);
00094     void foundNeighbours();
00095     void positionIncrement(QVector<unsigned int> &pos, unsigned int value = 1) const;
00096     QVector<QVector<unsigned int> > *getListNeighboursPositionsRecursive
      (const QVector<unsigned int> position, unsigned int dimension, QVector<unsigned int> lastAdd) const;
00097     QVector<QVector<unsigned int> > &getListNeighboursPositions(const
      QVector<unsigned int> position) const;
00098
00099     QVector<unsigned int> m_dimensions;
00100     QMap<QVector<unsigned int>, Cell* > m_cells;
00101 };
00102
00103 template class CellHandler::iteratorT<CellHandler, Cell>;
00104 template class CellHandler::iteratorT<const CellHandler, const Cell>
      ;
00105
00106 #endif // CELLHANDLER_H
```

7.9 creationdialog.cpp File Reference

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

7.10 creationdialog.cpp

```
00001 #include "creationdialog.h"
00002 #include <iostream>
00003
00004
00005 CreationDialog::CreationDialog(QWidget *parent)
00006 {
00007     QLabel *m_dimLabel= new QLabel(tr("Write your dimensions and their size, separated by a comma.\n"
00008                             "For instance, '25,25 ' will create a 2-dimensional 25x25 Automaton. "));
00009     QLabel *m_densityLabel = new QLabel(tr("Density :"));
00010     QLabel *m_stateMaxLabel = new QLabel(tr("Max state :"));
00011     m_densityBox = new QSpinBox();
00012     m_stateMaxBox = new QSpinBox();
00013
00014     QHBoxLayout *densityLayout = new QHBoxLayout();
00015     densityLayout->addWidget(m_densityLabel);
00016     densityLayout->addWidget(m_densityBox);
```

```
00017
00018     QHBoxLayout *stateMaxLayout = new QHBoxLayout();
00019     stateMaxLayout->addWidget(m_stateMaxLabel);
00020     stateMaxLayout->addWidget(m_stateMaxBox);
00021
00022     m_dimensionsEdit = new QLineEdit;
00023     QRegExp rgx("([0-9]+,)*");
00024     QRegExpValidator *v = new QRegExpValidator(rgx, this);
00025     m_dimensionsEdit->setValidator(v);
00026     m_doneBt = new QPushButton(tr("Done !"));
00027
00028     QVBoxLayout *layout = new QVBoxLayout;
00029
00030     QGroupBox *grpBox = createGenButtons();
00031
00032     layout->addWidget(m_dimLabel);
00033     layout->addWidget(m_dimensionsEdit);
00034     layout->addLayout(densityLayout);
00035     layout->addLayout(stateMaxLayout);
00036     layout->addWidget(grpBox);
00037     layout->addWidget(m_doneBt);
00038     setLayout(layout);
00039
00040     connect(m_doneBt, SIGNAL(clicked(bool)), this, SLOT(processSettings()));
00041
00042 }
00043
00049 QGroupBox *CreationDialog::createGenButtons(){
00050     m_groupBox = new QGroupBox(tr("Cell generation settings"));
00051     m_empGen = new QRadioButton(tr("&Empty Board"));
00052     m_randGen = new QRadioButton(tr("&Random"));
00053     m_symGen = new QRadioButton(tr("&Symmetrical"));
00054
00055     QVBoxLayout *layout = new QVBoxLayout;
00056     layout->addWidget(m_empGen);
00057     layout->addWidget(m_randGen);
00058     layout->addWidget(m_symGen);
00059
00060     m_groupBox->setLayout(layout);
00061
00062     return m_groupBox;
00063 }
00064
00070 void CreationDialog::processSettings(){
00071     QString dimensions = m_dimensionsEdit->text();
00072     if(dimensions.length() == 0){
00073         QMessageBox messageBox;
00074         messageBox.critical(0,"Error","You must specify valid dimensions !");
00075         messageBox.setFixedSize(500,200);
00076     }
00077     else{
00078         CellHandler::generationTypes genType;
00079         if(m_randGen == NULL)std::cout << "Radio button null line 68 \n" << std::flush;
00080         if(m_symGen->isChecked()) genType = CellHandler::generationTypes::symetric;
00081         else if(m_randGen->isChecked()) genType = CellHandler::generationTypes::random;
00082         else genType = CellHandler::generationTypes::empty;
00083         QStringList dimList = m_dimensionsEdit->text().split(",");
00084         QVector<unsigned int> dimensions;
00085         for(int i = 0; i < dimList.size(); i++) dimensions.append(dimList.at(i).toInt());
00086
00087         emit settingsFilled(dimensions, genType, m_stateMaxBox->value(),
      m_densityBox->value());
00088         this->close();
00089     }
00090
00091 }
00092
```

7.11 creationdialog.h File Reference

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

Classes

- class CreationDialog

  Automaton creation dialog box.

7.12 creationdialog.h

```
00001 #ifndef CREATIONDIALOG_H
00002 #define CREATIONDIALOG_H
00003
00004 #include <QtWidgets>
00005 #include "cellhandler.h"
00006
00013 class CreationDialog : public QDialog
00014 {
00015     Q_OBJECT
00016
00017 public:
00018     CreationDialog(QWidget *parent = 0);
00019
00020 signals:
00021     void settingsFilled(const QVector<unsigned int> dimensions,
00022                         CellHandler::generationTypes type =
      CellHandler::generationTypes::empty,
00023                         unsigned int stateMax = 1, unsigned int density = 20);
00024
00025 public slots:
00026     void processSettings();
00027
00028 private:
00029     QLineEdit *m_dimensionsEdit;
00030     QSpinBox *m_densityBox;
00031     QSpinBox *m_stateMaxBox;
00032     QPushButton *m_doneBt;
00033
00034     QGroupBox *m_groupBox;
00035     QRadioButton *m_empGen;
00036     QRadioButton *m_randGen;
00037     QRadioButton *m_symGen;
00038
00039     QGroupBox *createGenButtons();
00040
00041
00042
00043
00044
00045
00046 };
00047
00048 #endif // CREATEDIALOG_H
```

7.13 main.cpp File Reference

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

Functions

- int main (int argc, char *argv[ ])

7.13.1 Function Documentation

7.13.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Definition at line 6 of file main.cpp.

7.14 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 }
```

7.15 mainwindow.cpp File Reference

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

7.16 mainwindow.cpp

```
00001 #include "mainwindow.h"
00002 #include <iostream>
00003 MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent)
00004 {
00005     createIcons();
00006     createActions();
00007     createToolBar();
00008     createBoard();
00009
00010
00011     setMinimumSize(500,500);
00012     setWindowTitle("AutoCell");
00013
00014     m_cellHandler = NULL;
00015 }
00016
00022 void MainWindow::createIcons(){
00023     QPixmap fastBackwardPm(":/icons/icons/fast-backward.svg");
00024     QPixmap fastBackwardHoveredPm(":/icons/icons/fast-backward-full.svg");
00025     QPixmap fastForwardPm(":/icons/icons/fast-forward.svg");
00026     QPixmap fastForwardHoveredPm(":/icons/icons/fast-forward-full.svg");
00027     QPixmap playPm(":/icons/icons/play.svg");
00028     QPixmap playHoveredPm(":/icons/icons/play-full.svg");
00029     QPixmap newPm(":/icons/icons/new.svg");
00030     QPixmap openPm(":/icons/icons/open.svg");
00031     QPixmap savePm(":/icons/icons/save.svg");
00032     QPixmap pausePm(":/icons/icons/pause.svg");
00033     QPixmap resetPm(":/icons/icons/reset.svg");
00034
00035     m_fastBackwardIcon.addPixmap(fastBackwardPm, QIcon::Normal, QIcon::Off);
00036     m_fastBackwardIcon.addPixmap(fastBackwardHoveredPm, QIcon::Active, QIcon::Off);
00037     m_fastForwardIcon.addPixmap(fastForwardPm, QIcon::Normal, QIcon::Off);
```

```
00038     m_fastForwardIcon.addPixmap(fastForwardHoveredPm, QIcon::Active, QIcon::Off);
00039     m_playIcon.addPixmap(playPm, QIcon::Normal, QIcon::Off);
00040     m_playIcon.addPixmap(playHoveredPm, QIcon::Active, QIcon::Off);
00041     m_pauseIcon.addPixmap(pausePm, QIcon::Normal, QIcon::Off);
00042     m_newIcon.addPixmap(newPm, QIcon::Normal, QIcon::Off);
00043     m_saveIcon.addPixmap(savePm, QIcon::Normal, QIcon::Off);
00044     m_openIcon.addPixmap(openPm, QIcon::Normal, QIcon::Off);
00045     m_resetIcon.addPixmap(resetPm, QIcon::Normal, QIcon::Off);
00046 }
00047
00052 void MainWindow::createActions(){
00053     m_fastBackward = new QAction(m_fastBackwardIcon, tr("&fast backward"),
      this);
00054     m_fastForward = new QAction(m_fastForwardIcon, tr("&fast forward"), this)
      ;
00055     m_playPause = new QAction(m_playIcon, tr("Play"), this);
00056     m_saveAutomaton = new QAction(m_saveIcon, tr("Save automaton"), this);
00057     m_newAutomaton = new QAction(m_newIcon, tr("New automaton"), this);
00058     m_openAutomaton = new QAction(m_openIcon, tr("Open automaton"), this);
00059     m_resetAutomaton = new QAction(m_resetIcon, tr("Reset automaton"), this);
00060
00061
00062
00063     m_fastBackwardBt = new QToolButton();
00064     m_fastForwardBt = new QToolButton();
00065     m_playPauseBt = new QToolButton();
00066     m_saveAutomatonBt = new QToolButton();
00067     m_newAutomatonBt = new QToolButton();
00068     m_openAutomatonBt = new QToolButton();
00069     m_resetBt = new QToolButton();
00070
00071     m_fastBackwardBt->setDefaultAction(m_fastBackward);
00072     m_fastForwardBt->setDefaultAction(m_fastForward);
00073     m_playPauseBt->setDefaultAction(m_playPause);
00074     m_saveAutomatonBt->setDefaultAction(m_saveAutomaton);
00075     m_newAutomatonBt->setDefaultAction(m_newAutomaton);
00076     m_openAutomatonBt->setDefaultAction(m_openAutomaton);
00077     m_resetBt->setDefaultAction(m_resetAutomaton);
00078
00079     m_fastBackwardBt->setIconSize(QSize(30,30));
00080     m_fastForwardBt->setIconSize(QSize(30,30));
00081     m_playPauseBt->setIconSize(QSize(30,30));
00082     m_saveAutomatonBt->setIconSize(QSize(30,30));
00083     m_newAutomatonBt->setIconSize(QSize(30,30));
00084     m_openAutomatonBt->setIconSize(QSize(30,30));
00085     m_resetBt->setIconSize(QSize(30,30));
00086
00087     connect(m_openAutomatonBt, SIGNAL(clicked(bool)), this, SLOT(
      openFile()));
00088     connect(m_newAutomatonBt, SIGNAL(clicked(bool)), this, SLOT(
      openCreationWindow()));
00089     connect(m_saveAutomatonBt, SIGNAL(clicked(bool)), this, SLOT(
      saveToFile()));
00090     connect(m_fastForwardBt, SIGNAL(clicked(bool)), this, SLOT(
      forward()));
00091
00092 }
00093
00098 void MainWindow::createToolBar(){
00099     m_toolBar = new QToolBar(this);
00100     QLabel *m_speedLabel = new QLabel(tr("Speed : "));
00101     m_jumpSpeed = new QSpinBox(this);
00102     m_jumpSpeed->setValue(1);
00103     m_speedLabel->setFixedWidth(50);
00104     m_jumpSpeed->setFixedWidth(40);
00105     m_toolBar->setMovable(false);
00106
00107     QHBoxLayout *tbLayout = new QHBoxLayout(this);
00108     tbLayout->addWidget(m_newAutomatonBt, Qt::AlignCenter);
00109     tbLayout->addWidget(m_openAutomatonBt, Qt::AlignCenter);
00110     tbLayout->addWidget(m_saveAutomatonBt, Qt::AlignCenter);
00111     tbLayout->addWidget(m_fastBackwardBt, Qt::AlignCenter);
00112     tbLayout->addWidget(m_playPauseBt, Qt::AlignCenter);
00113     tbLayout->addWidget(m_fastForwardBt, Qt::AlignCenter);
00114     tbLayout->addWidget(m_speedLabel, Qt::AlignCenter);
00115     tbLayout->addWidget(m_jumpSpeed, Qt::AlignCenter);
00116     tbLayout->addWidget(m_resetBt, Qt::AlignCenter);
00117
00118
00119     tbLayout->setAlignment(Qt::AlignCenter);
00120     QWidget* wrapper = new QWidget();
00121     wrapper->setLayout(tbLayout);
00122     m_toolBar->addWidget(wrapper);
00123     addToolBar(m_toolBar);
00124
00125
00126 }
```

```
00127
00132 void MainWindow::createBoard(){
00133     m_Board = new QTableWidget(m_boardVSize, m_boardHSize, this);
00134         m_Board->setFixedSize(m_boardHSize*m_cellSize,
      m_boardVSize*m_cellSize);
00135         //setMinimumSize(m_boardHSize*m_cellSize,100+m_boardVSize*m_cellSize);
00136         m_Board->horizontalHeader()->setVisible(false);
00137         m_Board->verticalHeader()->setVisible(false);
00138         m_Board->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
00139         m_Board->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
00140         m_Board->setEditTriggers(QAbstractItemView::NoEditTriggers);
00141         for(unsigned int col = 0; col < m_boardHSize; ++col)
00142             m_Board->setColumnWidth(col, m_cellSize);
00143         for(unsigned int row = 0; row < m_boardVSize; ++row) {
00144             m_Board->setRowHeight(row, m_cellSize);
00145             for(unsigned int col = 0; col < m_boardHSize; ++col) {
00146                 m_Board->setItem(row, col, new QTableWidgetItem(""));
00147                 m_Board->item(row, col)->setBackgroundColor("white");
00148                 m_Board->item(row, col)->setTextColor("black");
00149             }
00150         }
00151      QScrollArea *scrollArea = new QScrollArea(this);
00152      scrollArea->setWidget(m_Board);
00153      setCentralWidget(scrollArea);
00154 }
00155
00156
00160 void MainWindow::openFile(){
00161     QString fileName = QFileDialog::getOpenFileName(this, tr("Open Cell file"), ".",
00162                                                     tr("Automaton cell files (*.atc)"));
00163     if(!fileName.isEmpty()){
00164         m_cellHandler = new CellHandler(fileName);
00165         QVector<unsigned int> dimensions = m_cellHandler->
      getDimensions();
00166         if(dimensions.size() > 1){
00167             m_boardVSize = dimensions[0];
00168             m_boardHSize = dimensions[1];
00169         }
00170         else{
00171             m_boardVSize = 1;
00172             m_boardHSize = dimensions[0];
00173         }
00174         createBoard();
00175         updateBoard();
00176     }
00177 }
00178
00179
00183 void MainWindow::saveToFile(){
00184     if(m_cellHandler != NULL){
00185         QString fileName = QFileDialog::getSaveFileName(this, tr("Save Automaton"),
00186                                                         ".", tr("Automaton Cells file (*.atc"));
00187         m_cellHandler->save(fileName);
00188
00189     }
00190     else{
00191         QMessageBox msgBox;
00192         msgBox.critical(0,"Error","Please create or import an Automaton first !");
00193         msgBox.setFixedSize(500,200);
00194     }
00195 }
00196
00201 void MainWindow::openCreationWindow(){
00202     CreationDialog *window = new CreationDialog(this);
00203     connect(window, SIGNAL(settingsFilled(QVector<uint>,
      CellHandler::generationTypes,uint,uint)),
00204             this, SLOT(setCellHandler(QVector<uint>,
      CellHandler::generationTypes,uint,uint)));
00205     window->show();
00206 }
00207
00214 void MainWindow::setCellHandler(const QVector<unsigned int> dimensions,
00215                                 CellHandler::generationTypes type,
00216                                 unsigned int stateMax, unsigned int density){
00217     m_cellHandler = new CellHandler(dimensions, type, stateMax, density);
00218     if(dimensions.size() > 1){
00219         m_boardVSize = dimensions[0];
00220         m_boardHSize = dimensions[1];
00221     }
00222     else{
00223         m_boardVSize = 1;
00224         m_boardHSize = dimensions[0];
00225     }
00226     createBoard();
00227     updateBoard();
00228 }
00229
```

```
00234 void MainWindow::nextState(int n){
00235     if(m_cellHandler == NULL){
00236         QMessageBox msgBox;
00237         msgBox.critical(0,"Error","Please create or import an Automaton first !");
00238         msgBox.setFixedSize(500,200);
00239     }
00240     else{
00241         for(unsigned int i = 0; i < n; i++) m_cellHandler->
      nextStates();
00242         updateBoard();
00243     }
00244 }
00245
00250 void MainWindow::updateBoard(){
00251     if(m_cellHandler == NULL){
00252         QMessageBox msgBox;
00253         msgBox.critical(0,"Error","Please create or import an Automaton first !");
00254         msgBox.setFixedSize(500,200);
00255     }
00256     else{
00257         int i = 0;
00258         int j = 0;
00259         for (CellHandler::iterator it = m_cellHandler->
      begin(); it != m_cellHandler->end() && it.changedDimension() < 2; ++it){
00260                 if(it.changedDimension() > 0){
00261                     i = 0;
00262                     j++;
00263                     std::cout << std::endl;
00264                 }
00265                 m_Board->item(i,j)->setText(QString::number(it->getState()));
00266                 i++;
00267         }
00268     }
00269
00270 }
00271
00276 void MainWindow::forward(){
00277     nextState(m_jumpSpeed->value());
00278 }
```

7.17 mainwindow.h File Reference

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

Classes

- class MainWindow

  Simulation window.

7.18 mainwindow.h

```
00001 #ifndef MAINWINDOW_H
00002 #define MAINWINDOW_H
00003
00004 #include <QMainWindow>
00005 #include <QtWidgets>
00006 #include "cellhandler.h"
00007 #include "creationdialog.h"
00008
00009
00016 class MainWindow : public QMainWindow
00017 {
00018     Q_OBJECT
00019
00020     CellHandler *m_cellHandler;
```

```
00021
00023     QIcon m_fastBackwardIcon;
00024     QIcon m_fastForwardIcon;
00025     QIcon m_playIcon;
00026     QIcon m_pauseIcon;
00027     QIcon m_newIcon;
00028     QIcon m_saveIcon;
00029     QIcon m_openIcon;
00030     QIcon m_resetIcon;
00031
00033     QAction *m_playPause;
00034     QAction *m_nextState;
00035     QAction *m_previousState;
00036     QAction *m_fastForward;
00037     QAction *m_fastBackward;
00038     QAction *m_openAutomaton;
00039     QAction *m_saveAutomaton;
00040     QAction *m_newAutomaton;
00041     QAction *m_resetAutomaton;
00042
00044     QToolButton *m_playPauseBt;
00045     QToolButton *m_nextStateBt;
00046     QToolButton *m_previousStateBt;
00047     QToolButton *m_fastForwardBt;
00048     QToolButton *m_fastBackwardBt;
00049     QToolButton *m_openAutomatonBt;
00050     QToolButton *m_saveAutomatonBt;
00051     QToolButton *m_newAutomatonBt;
00052     QToolButton *m_resetBt;
00053
00054
00055     QSpinBox *m_jumpSpeed;
00056     QLabel *m_speedLabel;
00057
00058     QToolBar *m_toolBar;
00059
00060     QTableWidget *m_Board;
00061
00063     unsigned int m_boardHSize = 25;
00064     unsigned int m_boardVSize = 25;
00065     unsigned int m_cellSize = 30;
00066
00067     void createIcons();
00068     void createActions();
00069     void createToolBar();
00070     void createBoard();
00071
00072
00073     void updateBoard();
00074     void nextState(int n);
00075
00076
00077 public:
00078     explicit MainWindow(QWidget *parent = nullptr);
00079
00080
00081 signals:
00082
00083 public slots:
00084     void openFile();
00085     void saveToFile();
00086     void openCreationWindow();
00087     void setCellHandler(const QVector<unsigned int> dimensions,
00088                         CellHandler::generationTypes type =
      CellHandler::generationTypes::empty,
00089                         unsigned int stateMax = 1, unsigned int density = 20);
00090     void forward();
00091
00092 };
00093
00094 #endif // MAINWINDOW_H
```

7.19 presentation.md File Reference

7.20 presentation.md

```
00001 \page Presentation
00002 # What is AutoCell
00003 The purpose of this project is to create a Cellular Automate Simulator.
00004
00005 \includedoc CellHandler
```

7.21 README.md File Reference

7.22 README.md

```
00001 \mainpage
00002
00003 To generate the Documentation, go in Documentation directory and run `make`.
00004
00005 It will generate html doc (in `output/html/index.html`) and latex doc (pdf output directely in
       Documentation directory (`docPdf.pdf`).
```

Index