

AutoCell

Generated by Doxygen 1.8.14

Contents

1	Main Page	1
2	Presentation	3
3	Class Index	5
3.1	Class List	5
4	File Index	7
4.1	File List	7
5	Class Documentation	9
5.1	Cell Class Reference	9
5.1.1	Detailed Description	10
5.1.2	Constructor & Destructor Documentation	10
5.1.2.1	Cell()	10
5.1.3	Member Function Documentation	10
5.1.3.1	addNeighbour()	10
5.1.3.2	forceState()	11
5.1.3.3	getNeighbour()	11
5.1.3.4	getNeighbours()	12
5.1.3.5	getRelativePosition()	12
5.1.3.6	getState()	12
5.1.3.7	setState()	12
5.1.3.8	validState()	13
5.1.4	Member Data Documentation	13

5.1.4.1	<code>m_neighbours</code>	13
5.1.4.2	<code>m_nextState</code>	13
5.1.4.3	<code>m_state</code>	14
5.2	CellHandler Class Reference	14
5.2.1	Detailed Description	15
5.2.2	Member Typedef Documentation	15
5.2.2.1	<code>const_iterator</code>	16
5.2.2.2	<code>iterator</code>	16
5.2.3	Member Enumeration Documentation	16
5.2.3.1	<code>generationTypes</code>	16
5.2.4	Constructor & Destructor Documentation	16
5.2.4.1	<code>CellHandler()</code> [1/2]	16
5.2.4.2	<code>CellHandler()</code> [2/2]	17
5.2.4.3	<code>~CellHandler()</code>	18
5.2.5	Member Function Documentation	18
5.2.5.1	<code>begin()</code> [1/2]	18
5.2.5.2	<code>begin()</code> [2/2]	18
5.2.5.3	<code>end()</code>	18
5.2.5.4	<code>foundNeighbours()</code>	19
5.2.5.5	<code>generate()</code>	19
5.2.5.6	<code>getCell()</code>	19
5.2.5.7	<code>getDimensions()</code>	20
5.2.5.8	<code>getListNeighboursPositions()</code>	20
5.2.5.9	<code>getListNeighboursPositionsRecursive()</code>	20
5.2.5.10	<code>load()</code>	21
5.2.5.11	<code>nextStates()</code>	22
5.2.5.12	<code>positionIncrement()</code>	22
5.2.5.13	<code>print()</code>	23
5.2.5.14	<code>save()</code>	23
5.2.6	Member Data Documentation	23

5.2.6.1	<code>m_cells</code>	24
5.2.6.2	<code>m_dimensions</code>	24
5.3	<code>CellHandler::iteratorT< T, R ></code> Class Template Reference	24
5.3.1	Detailed Description	25
5.3.2	Constructor & Destructor Documentation	25
5.3.2.1	<code>iteratorT()</code>	25
5.3.3	Member Function Documentation	26
5.3.3.1	<code>changedDimension()</code>	26
5.3.3.2	<code>operator!=()</code>	26
5.3.3.3	<code>operator*()</code>	26
5.3.3.4	<code>operator++()</code>	27
5.3.3.5	<code>operator->()</code>	27
5.3.4	Friends And Related Function Documentation	27
5.3.4.1	<code>CellHandler</code>	27
5.3.5	Member Data Documentation	27
5.3.5.1	<code>m_changedDimension</code>	27
5.3.5.2	<code>m_finished</code>	28
5.3.5.3	<code>m_handler</code>	28
5.3.5.4	<code>m_position</code>	28
5.3.5.5	<code>m_zero</code>	28
6	File Documentation	29
6.1	<code>cell.cpp</code> File Reference	29
6.2	<code>cell.cpp</code>	29
6.3	<code>cell.h</code> File Reference	30
6.4	<code>cell.h</code>	30
6.5	<code>cellhandler.cpp</code> File Reference	30
6.6	<code>cellhandler.cpp</code>	31
6.7	<code>cellhandler.h</code> File Reference	36
6.8	<code>cellhandler.h</code>	36
6.9	<code>main.cpp</code> File Reference	37
6.9.1	Function Documentation	37
6.9.1.1	<code>main()</code>	37
6.10	<code>main.cpp</code>	38
6.11	<code>presentation.md</code> File Reference	38
6.12	<code>presentation.md</code>	38
6.13	<code>README.md</code> File Reference	38
6.14	<code>README.md</code>	38
	Index	39

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 directly 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

Class Index

3.1 Class List

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

Cell	Contains the state, the next state and the neighbours	9
CellHandler	Cell container and cell generator	14
CellHandler::iteratorT< T, R >	Implementation of iterator design pattern with a template to generate iterator and const_iterator at the same time	24

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

cell.cpp	29
cell.h	30
cellhandler.cpp	30
cellhandler.h	36
main.cpp	37

Chapter 5

Class Documentation

5.1 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.

5.1.1 Detailed Description

Contains the state, the next state and the neighbours.

Definition at line 10 of file `cell.h`.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 `Cell()`

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

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

Parameters

<code>state</code>	<code>Cell</code> state, dead state by default
--------------------	--

Definition at line 7 of file `cell.cpp`.

5.1.3 Member Function Documentation

5.1.3.1 `addNeighbour()`

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

Add a new neighbour to the `Cell`.

Parameters

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

Returns

False if the neighbour already exists

Definition at line 60 of file [cell.cpp](#).

References [m_neighbours](#).

5.1.3.2 forceState()

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

Force the state change.

Is equivalent to setState followed by validState

Parameters

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

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

References [m_nextState](#), and [m_state](#).

5.1.3.3 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](#).

5.1.3.4 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](#).

5.1.3.5 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

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

Parameters

<i>cellPosition</i>	Cell Position
<i>neighbourPosition</i>	Neighbour absolute position

Definition at line 91 of file [cell.cpp](#).

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

5.1.3.6 getState()

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

Access current cell state.

Definition at line 48 of file [cell.cpp](#).

References [m_state](#).

5.1.3.7 setState()

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

Set temporary state.

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

Parameters

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

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

References [m_nextState](#).

5.1.3.8 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](#).

5.1.4 Member Data Documentation

5.1.4.1 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\(\)](#).

5.1.4.2 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\(\)](#).

5.1.4.3 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](#)

5.2 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.

5.2.1 Detailed Description

[Cell](#) container and cell generator.

Generate cells from a json file.

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

5.2.2 Member Typedef Documentation

5.2.2.1 const_iterator

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

Definition at line 67 of file [cellhandler.h](#).

5.2.2.2 iterator

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

Definition at line 68 of file [cellhandler.h](#).

5.2.3 Member Enumeration Documentation

5.2.3.1 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 72 of file [cellhandler.h](#).

5.2.4 Constructor & Destructor Documentation

5.2.4.1 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

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

Exceptions

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

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

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

5.2.4.2 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

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

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

References [empty](#), [foundNeighbours\(\)](#), [generate\(\)](#), [m_cells](#), [m_dimensions](#), and [positionIncrement\(\)](#).

5.2.4.3 `~CellHandler()`

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

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

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

References [m_cells](#).

5.2.5 Member Function Documentation

5.2.5.1 `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\(\)](#), and [save\(\)](#).

5.2.5.2 `begin()` [2/2]

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

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

Definition at line 255 of file [cellhandler.cpp](#).

5.2.5.3 `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\(\)](#), and [save\(\)](#).

5.2.5.4 foundNeighbours()

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

Set the neighbours of each cells.

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

Definition at line 364 of file [cellhandler.cpp](#).

References [getListNeighboursPositions\(\)](#), [Cell::getRelativePosition\(\)](#), [m_cells](#), [m_dimensions](#), and [positionIncrement\(\)](#).

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

5.2.5.5 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

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

Definition at line 176 of file [cellhandler.cpp](#).

References [m_cells](#), [m_dimensions](#), [positionIncrement\(\)](#), [random](#), and [symetric](#).

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

5.2.5.6 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](#).

5.2.5.7 `getDimensions()`

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

Accessor of `m_dimensions`.

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

References [m_dimensions](#).

5.2.5.8 `getListNeighboursPositions()`

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

Prepare the call of the recursive version of itself.

Parameters

<i>position</i>	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\(\)](#).

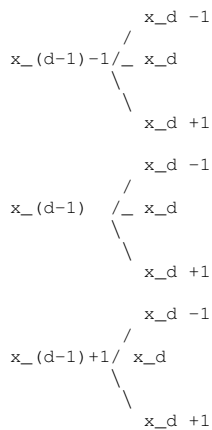
5.2.5.9 `getListNeighboursPositionsRecursive()`

```
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 $O(3^{\text{dimension}})$

Piece of the tree:



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

Parameters

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

Returns

List of position

Definition at line 464 of file [cellhandler.cpp](#).

References [m_dimensions](#).

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

5.2.5.10 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

<i>json</i>	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\(\)](#).

5.2.5.11 nextStates()

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

Valid the state of all cells.

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

References [m_cells](#).

5.2.5.12 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

<i>pos</i>	Position to increment
<i>value</i>	Value to add, 1 by default

Definition at line 394 of file [cellhandler.cpp](#).

References [m_dimensions](#).

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

5.2.5.13 print()

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

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

Parameters

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

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

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

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

5.2.5.14 save()

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

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

Parameters

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

Returns

False if there was a problem

Exceptions

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

Definition at line 136 of file [cellhandler.cpp](#).

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

5.2.6 Member Data Documentation

5.2.6.1 m_cells

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

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

Definition at line 103 of file [cellhandler.h](#).

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

5.2.6.2 m_dimensions

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

Vector of x dimensions.

Definition at line 102 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](#)

5.3 CellHandler::iteratorT< T, R > 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](#) (T *handler)
Construct an initial iterator to browse the [CellHandler](#).
- [iteratorT](#) & [operator++](#) ()
Increment the current position and handle dimension changes.
- R * [operator->](#) () const
Get the current cell.
- R * [operator*](#) () const
Get the current cell.
- bool [operator!=](#) (bool finished) const
- unsigned int [changedDimension](#) () const
Return the number of dimensions we change.

Private Attributes

- 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](#)

5.3.1 Detailed Description

```
template<typename T, typename R>
class CellHandler::iteratorT< T, R >
```

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

Everywhere, T will be nor [CellHandler](#) nor const [CellHandler](#) and R will be nor [Cell](#) nor const [Cell](#).

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 is 3 dimensions, there will be a empty line between 2D groups.

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

5.3.2 Constructor & Destructor Documentation

5.3.2.1 iteratorT()

```
template<typename T , typename R >
CellHandler::iteratorT< T, R >::iteratorT (
    T * handler )
```

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

Parameters

<i>handler</i>	CellHandler to browse
----------------	---------------------------------------

Definition at line 504 of file [cellhandler.cpp](#).

References [CellHandler::iteratorT< T, R >::m_position](#), and [CellHandler::iteratorT< T, R >::m_zero](#).

5.3.3 Member Function Documentation

5.3.3.1 `changedDimension()`

```
template<typename T , typename R >
unsigned int CellHandler::iteratorT< T, R >::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 `changedDimension` return 2 (because of the 2 zeros).

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

5.3.3.2 `operator!=()`

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

Definition at line 54 of file [cellhandler.h](#).

References [CellHandler::iteratorT< T, R >::m_finished](#).

5.3.3.3 `operator*()`

```
template<typename T , typename R >
R * CellHandler::iteratorT< T, R >::operator\* ( ) const
```

Get the current cell.

Definition at line 555 of file [cellhandler.cpp](#).

5.3.3.4 operator++()

```
template<typename T , typename R >  
CellHandler::iteratorT< T, R > & CellHandler::iteratorT< T, R >::operator++ ( )
```

Increment the current position and handle dimension changes.

Definition at line 518 of file [cellhandler.cpp](#).

5.3.3.5 operator->()

```
template<typename T , typename R >  
R * CellHandler::iteratorT< T, R >::operator-> ( ) const
```

Get the current cell.

Definition at line 546 of file [cellhandler.cpp](#).

5.3.4 Friends And Related Function Documentation

5.3.4.1 CellHandler

```
template<typename T , typename R >  
friend class CellHandler [friend]
```

Definition at line 46 of file [cellhandler.h](#).

5.3.5 Member Data Documentation

5.3.5.1 m_changedDimension

```
template<typename T , typename R >  
unsigned int CellHandler::iteratorT< T, R >::m_changedDimension [private]
```

Save the number of dimension change.

Definition at line 64 of file [cellhandler.h](#).

5.3.5.2 m_finished

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

If we reach the last position.

Definition at line 62 of file [cellhandler.h](#).

Referenced by [CellHandler::iteratorT< T, R >::operator!=\(\)](#).

5.3.5.3 m_handler

```
template<typename T , typename R >
T* CellHandler::iteratorT< T, R >::m_handler [private]
```

[CellHandler](#) to go through.

Definition at line 60 of file [cellhandler.h](#).

5.3.5.4 m_position

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

Current position of the iterator.

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

Referenced by [CellHandler::iteratorT< T, R >::iteratorT\(\)](#).

5.3.5.5 m_zero

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

Nul vector of the good dimension (depend of m_handler)

Definition at line 63 of file [cellhandler.h](#).

Referenced by [CellHandler::iteratorT< T, R >::iteratorT\(\)](#).

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

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

Chapter 6

File Documentation

6.1 cell.cpp File Reference

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

6.2 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 }

```

6.3 cell.h File Reference

```

#include <QVector>
#include <QDebug>

```

Classes

- class [Cell](#)
Contains the state, the next state and the neighbours.

6.4 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

```

6.5 cellhandler.cpp File Reference

```

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

```

6.6 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     // Loading of the json file
00045     if (!loadDoc.isObject())
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,
00066     generationTypes type, unsigned int stateMax, unsigned int density)
00067 {
00068     m_dimensions = dimensions;
00069     QVector<unsigned int> position;
00070     unsigned int size = 1;
00071
00072     // Set position vector to 0
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 !=
00100         m_cells.end(); ++it)
00101     {
00102         delete it.value();
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 (unsigned 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     QJsonDocument saveDoc(json);
00163     saveFile.write(saveDoc.toJson());
00164
00165     saveFile.close();
00166     return true;
00167 }
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 (unsigned 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 (unsigned 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 (unsigned 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[0-9]*"));
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     unsigned int product = 1;
00323     // Dimensions construction
00324     for (unsigned 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 (unsigned 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 void CellHandler::foundNeighbours()
00359 {
00360     QVector<unsigned int> currentPosition;
00361     // Set position vector to 0
00362     for (unsigned short i = 0; i < m_dimensions.size(); i++)
00363     {
00364         currentPosition.push_back(0);
00365     }
00366     // Modification of all the cells
00367     for (unsigned int j = 0; j < m_cells.size(); j++)
00368     {
00369         // Get the list of the neighbours positions
00370         // This function is recursive
00371         QVector<QVector<unsigned int> > listPosition(getListNeighboursPositions(
00372             currentPosition));
00373
00374         // Adding neighbours
00375         for (unsigned int i = 0; i < listPosition.size(); i++)
00376             m_cells.value(currentPosition)->addNeighbour(m_cells.value(listPosition.at(i)),
00377                 Cell::getRelativePosition(currentPosition, listPosition.at(i)));
00378         positionIncrement(currentPosition);
00379     }
00380 }
00381
00382 void CellHandler::positionIncrement(QVector<unsigned int> &pos, unsigned int
00383 value) const
00384 {
00385     pos.replace(0, pos.at(0) + value); // adding the value to the first digit
00386
00387     // Carry management
00388     for (unsigned short i = 0; i < m_dimensions.size(); i++)
00389     {
00390         if (pos.at(i) >= m_dimensions.at(i) && pos.at(i) <
00391             m_dimensions.at(i)*2)
00392         {
00393             pos.replace(i, 0);
00394             if (i + 1 != m_dimensions.size())
00395                 pos.replace(i+1, pos.at(i+1)+1);
00396         }
00397         else if (pos.at(i) >= m_dimensions.at(i))
00398         {
00399             pos.replace(i, pos.at(i) - m_dimensions.at(i));
00400             if (i + 1 != m_dimensions.size())
00401                 pos.replace(i+1, pos.at(i+1)+1);
00402             i--;
00403         }
00404     }
00405 }
00406
00407 QVector<QVector<unsigned int> > CellHandler::getListNeighboursPositions
00408 (const QVector<unsigned int> position) const
00409 {
00410     QVector<QVector<unsigned int> > *list = getListNeighboursPositionsRecursive
00411         (position, position.size(), position);
00412     // We remove the position of the cell
00413     list->removeAll(position);
00414     return *list;
00415 }
00416
00417 QVector<QVector<unsigned int> > *
00418 CellHandler::getListNeighboursPositionsRecursive(const
00419 QVector<unsigned int> position, unsigned int dimension, QVector<unsigned int> lastAdd) const
00420 {
00421     if (dimension == 0) // Stop condition
00422     {
00423         QVector<QVector<unsigned int> > *list = new QVector<QVector<unsigned int> >;
00424         return list;
00425     }
00426     QVector<QVector<unsigned int> > *listPositions = new QVector<QVector<unsigned int> >;
00427
00428     QVector<unsigned int> modifiedPosition(lastAdd);
00429 }

```



```

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 T, typename R>
00504 CellHandler::iteratorT<T,R>::iteratorT(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 T, typename R>
00518 CellHandler::iteratorT<T,R> &
CellHandler::iteratorT<T, R>::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 T, typename R>
00546 R* CellHandler::iteratorT<T,R>::operator->() const
00547 {
00548     return m_handler->m_cells.value(m_position);
00549 }
00550
00551
00554 template<typename T, typename R>
00555 R *CellHandler::iteratorT<T, R>::operator*() const
00556 {
00557     return m_handler->m_cells.value(m_position);
00558 }
00559
00565 template<typename T, typename R>
00566 unsigned int CellHandler::iteratorT<T,R>::changedDimension()
const
00567 {
00568     return m_changedDimension;
00569 }
00570

```

6.7 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< T, R >](#)
Implementation of iterator design pattern with a template to generate iterator and const_iterator at the same time.

6.8 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
00043     template <typename T, typename R>
00044     class iteratorT
00045     {
00046     public:
00047         friend class CellHandler;
00048         iteratorT(T* handler);
00049
00050         iteratorT& operator++();
00051         R* operator->() const;
00052         R* operator*() const;
00053
00054         bool operator!=(bool finished) const { return (m_finished != finished); }
00055         unsigned int changedDimension() const;
00056
00057
00058
00059     private:
00060         T *m_handler;
00061         QVector<unsigned int> m_position;
00062         bool m_finished = false;
00063         QVector<unsigned int> m_zero;
00064         unsigned int m_changedDimension;
00065     };
00066 public:
00067     typedef iteratorT<const CellHandler, const Cell>
00068     const_iterator;
00069     typedef iteratorT<CellHandler, Cell> iterator;
00070
00072     enum generationTypes {
00073         empty,
```

```

00074         random,
00075         symetric
00076     };
00077
00078     CellHandler(const QString filename);
00079     CellHandler(const QVector<unsigned int> dimensions,
generationTypes type = empty, unsigned int stateMax = 1, unsigned int density = 20);
00080     virtual ~CellHandler();
00081
00082     Cell* getCell(const QVector<unsigned int> position) const;
00083     QVector<unsigned int> getDimensions() const;
00084     void nextStates() const;
00085
00086     bool save(QString filename) const;
00087
00088     void generate(generationTypes type, unsigned int stateMax = 1, unsigned short
density = 50);
00089     void print(std::ostream &stream) const;
00090
00091     const_iterator begin() const;
00092     iterator begin();
00093     bool end() const;
00094
00095 private:
00096     bool load(const QJsonObject &json);
00097     void foundNeighbours();
00098     void positionIncrement(QVector<unsigned int> &pos, unsigned int value = 1) const;
00099     QVector<QVector<unsigned int> > *getListNeighboursPositionsRecursive
(const QVector<unsigned int> position, unsigned int dimension, QVector<unsigned int> lastAdd) const;
00100     QVector<QVector<unsigned int> > &getListNeighboursPositions(const
QVector<unsigned int> position) const;
00101
00102     QVector<unsigned int> m_dimensions;
00103     QMap<QVector<unsigned int>, Cell* > m_cells;
00104 };
00105
00106 template class CellHandler::iteratorT<CellHandler, Cell>;
00107 template class CellHandler::iteratorT<const CellHandler, const Cell>
;
00108
00109 #endif // CELLHANDLER_H

```

6.9 main.cpp File Reference

```

#include <QApplication>
#include <QDebug>
#include <iostream>
#include "cellhandler.h"

```

Functions

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

6.9.1 Function Documentation

6.9.1.1 main()

```

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

```

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

References [CellHandler::print\(\)](#).

6.10 main.cpp

```
00001 #include <QApplication>
00002 #include <QDebug>
00003 #include <iostream>
00004 #include "cellhandler.h"
00005
00006 int main(int argc, char * argv[])
00007 {
00008     //QApplication app(argc, argv);
00009     CellHandler handler(QVector<unsigned int>{10,10});
00010     handler.print(std::cout);
00011
00012     for (CellHandler::iterator it = handler.begin(); it != handler.end(); ++it)
00013         it->forceState(2);
00014     handler.print(std::cout);
00015     return 0;
00016 }
```

6.11 presentation.md File Reference

6.12 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
```

6.13 README.md File Reference

6.14 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 directly in
    Documentation directory ('docPdf.pdf')).
```

Index

- ~CellHandler
 - CellHandler, 17
- addNeighbour
 - Cell, 10
- begin
 - CellHandler, 18
- Cell, 9
 - addNeighbour, 10
 - Cell, 10
 - forceState, 11
 - getNeighbour, 11
 - getNeighbours, 11
 - getRelativePosition, 12
 - getState, 12
 - m_neighbours, 13
 - m_nextState, 13
 - m_state, 13
 - setState, 12
 - validState, 13
- cell.cpp, 29
- cell.h, 30
- CellHandler, 14
 - ~CellHandler, 17
 - begin, 18
 - CellHandler, 16, 17
 - CellHandler::iteratorT, 27
 - const_iterator, 15
 - end, 18
 - foundNeighbours, 18
 - generate, 19
 - generationTypes, 16
 - getCell, 19
 - getDimensions, 19
 - getListNeighboursPositions, 20
 - getListNeighboursPositionsRecursive, 20
 - iterator, 16
 - load, 21
 - m_cells, 23
 - m_dimensions, 24
 - nextStates, 22
 - positionIncrement, 22
 - print, 22
 - save, 23
- CellHandler::iteratorT< T, R >, 24
- CellHandler::iteratorT
 - CellHandler, 27
 - changedDimension, 26
 - iteratorT, 25
 - m_changedDimension, 27
 - m_finished, 27
 - m_handler, 28
 - m_position, 28
 - m_zero, 28
 - operator!=, 26
 - operator*, 26
 - operator++, 26
 - operator->, 27
- cellhandler.cpp, 30, 31
- cellhandler.h, 36
- changedDimension
 - CellHandler::iteratorT, 26
- const_iterator
 - CellHandler, 15
- end
 - CellHandler, 18
- forceState
 - Cell, 11
- foundNeighbours
 - CellHandler, 18
- generate
 - CellHandler, 19
- generationTypes
 - CellHandler, 16
- getCell
 - CellHandler, 19
- getDimensions
 - CellHandler, 19
- getListNeighboursPositions
 - CellHandler, 20
- getListNeighboursPositionsRecursive
 - CellHandler, 20
- getNeighbour
 - Cell, 11
- getNeighbours
 - Cell, 11
- getRelativePosition
 - Cell, 12
- getState
 - Cell, 12
- iterator
 - CellHandler, 16
- iteratorT
 - CellHandler::iteratorT, 25

- load
 - CellHandler, [21](#)
- m_cells
 - CellHandler, [23](#)
- m_changedDimension
 - CellHandler::iteratorT, [27](#)
- m_dimensions
 - CellHandler, [24](#)
- m_finished
 - CellHandler::iteratorT, [27](#)
- m_handler
 - CellHandler::iteratorT, [28](#)
- m_neighbours
 - Cell, [13](#)
- m_nextState
 - Cell, [13](#)
- m_position
 - CellHandler::iteratorT, [28](#)
- m_state
 - Cell, [13](#)
- m_zero
 - CellHandler::iteratorT, [28](#)
- main
 - main.cpp, [37](#)
- main.cpp, [37](#), [38](#)
 - main, [37](#)
- nextStates
 - CellHandler, [22](#)
- operator!=
 - CellHandler::iteratorT, [26](#)
- operator*
 - CellHandler::iteratorT, [26](#)
- operator++
 - CellHandler::iteratorT, [26](#)
- operator->
 - CellHandler::iteratorT, [27](#)
- positionIncrement
 - CellHandler, [22](#)
- presentation.md, [38](#)
- print
 - CellHandler, [22](#)
- README.md, [38](#)
- save
 - CellHandler, [23](#)
- setState
 - Cell, [12](#)
- validState
 - Cell, [13](#)