

AutoCell

Generated by Doxygen 1.8.13

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	9
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	getNeighbours()	10
5.1.3.3	getState()	11
5.1.3.4	setState()	11
5.1.3.5	validState()	11
5.1.4	Member Data Documentation	11
5.1.4.1	m_neighbours	12
5.1.4.2	m_nextState	12
5.1.4.3	m_state	12

5.2	CellHandler Class Reference	12
5.2.1	Detailed Description	14
5.2.2	Member Enumeration Documentation	14
5.2.2.1	generationTypes	14
5.2.3	Constructor & Destructor Documentation	14
5.2.3.1	CellHandler() [1/2]	14
5.2.3.2	CellHandler() [2/2]	15
5.2.3.3	~CellHandler()	15
5.2.4	Member Function Documentation	16
5.2.4.1	begin()	16
5.2.4.2	end()	16
5.2.4.3	foundNeighbours()	16
5.2.4.4	generate()	16
5.2.4.5	getCell()	17
5.2.4.6	getListNeighboursPositions()	17
5.2.4.7	getListNeighboursPositionsRecursive()	18
5.2.4.8	load()	18
5.2.4.9	nextStates()	19
5.2.4.10	positionIncrement()	19
5.2.4.11	print()	20
5.2.4.12	save()	20
5.2.5	Member Data Documentation	21
5.2.5.1	m_cells	21
5.2.5.2	m_dimensions	21
5.3	CellHandler::iterator Class Reference	21
5.3.1	Detailed Description	22
5.3.2	Constructor & Destructor Documentation	22
5.3.2.1	iterator()	22
5.3.3	Member Function Documentation	23
5.3.3.1	changedDimension()	23
5.3.3.2	operator!=()	23
5.3.3.3	operator*()	23
5.3.3.4	operator++()	24
5.3.3.5	operator->()	24
5.3.4	Friends And Related Function Documentation	24
5.3.4.1	CellHandler	24
5.3.5	Member Data Documentation	24
5.3.5.1	m_changedDimension	24
5.3.5.2	m_finished	25
5.3.5.3	m_handler	25
5.3.5.4	m_position	25
5.3.5.5	m_zero	25

6 File Documentation	27
6.1 cell.cpp File Reference	27
6.2 cell.cpp	27
6.3 cell.h File Reference	27
6.4 cell.h	28
6.5 cellhandler.cpp File Reference	28
6.6 cellhandler.cpp	28
6.7 cellhandler.h File Reference	33
6.8 cellhandler.h	34
6.9 homepage.md File Reference	34
6.10 homepage.md	35
6.11 main.cpp File Reference	35
6.11.1 Function Documentation	35
6.11.1.1 main()	35
6.12 main.cpp	35
6.13 README.md File Reference	36
6.14 README.md	36
Index	37

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	12
CellHandler::iterator	Implementation of iterator design pattern	21

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

cell.cpp	27
cell.h	27
cellhandler.cpp	28
cellhandler.h	33
main.cpp	35

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.
- unsigned int [getState](#) () const
Access current cell state.
- bool [addNeighbour](#) (const [Cell](#) *neighbour)
Add a new neighbour to the [Cell](#).
- QVector< const [Cell](#) * > [getNeighbours](#) () const
Access neighbours list.

Private Attributes

- unsigned int [m_state](#)
Current state.
- unsigned int [m_nextState](#)
Temporary state, before validation.
- QVector< const [Cell](#) * > [m_neighbours](#)
[Cell](#)'s neighbours.

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

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

Definition at line 8 of file [cell.cpp](#).

5.1.3 Member Function Documentation

5.1.3.1 addNeighbour()

```
bool Cell::addNeighbour (
    const Cell * neighbour )
```

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

Parameters

<i>neighbour</i>	New neighbour
------------------	---------------

Returns

False if the neighbour already exists

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

References [m_neighbours](#).

5.1.3.2 getNeighbours()

```
QVector< const Cell * > Cell::getNeighbours ( ) const
```

Access neighbours list.

Definition at line 63 of file [cell.cpp](#).

References [m_neighbours](#).

5.1.3.3 `getState()`

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

Access current cell state.

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

References [m_state](#).

5.1.3.4 `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 22 of file [cell.cpp](#).

References [m_nextState](#).

5.1.3.5 `validState()`

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

Validate temporary state.

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

Definition at line 33 of file [cell.cpp](#).

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

5.1.4 Member Data Documentation

5.1.4.1 m_neighbours

```
QVector<const Cell*> Cell::m\_neighbours [private]
```

[Cell](#)'s neighbours.

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

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

5.1.4.2 m_nextState

```
unsigned int Cell::m\_nextState [private]
```

Temporary state, before validation.

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

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

5.1.4.3 m_state

```
unsigned int Cell::m\_state [private]
```

Current state.

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

Referenced by [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 [iterator](#)

Implementation of iterator design pattern.

Public Types

- enum [generationTypes](#) { [empty](#), [random](#), [symetric](#) }
- Type of random generation.*

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=50)
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.
- void [nextStates](#) ()
Valid the state of all cells.
- bool [save](#) (QString filename)
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)
Print in the given stream the [CellHandler](#).
- [iterator](#) [begin](#) ()
Give the iterator which corresponds to the current [CellHandler](#).
- bool [end](#) ()
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 18 of file [cellhandler.h](#).

5.2.2 Member Enumeration Documentation

5.2.2.1 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 63 of file [cellhandler.h](#).

5.2.3 Constructor & Destructor Documentation

5.2.3.1 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 26 of file [cellhandler.cpp](#).

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

5.2.3.2 CellHandler() [2/2]

```
CellHandler::CellHandler (
    const QVector< unsigned int > dimensions,
    generationTypes type = empty,
    unsigned int stateMax = 1,
    unsigned int density = 50 )
```

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 66 of file [cellhandler.cpp](#).

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

5.2.3.3 ~CellHandler()

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

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

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

References [m_cells](#).

5.2.4 Member Function Documentation

5.2.4.1 begin()

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

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

Definition at line [271](#) of file [cellhandler.cpp](#).

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

5.2.4.2 end()

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

End condition of the iterator.

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

Definition at line [281](#) of file [cellhandler.cpp](#).

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

5.2.4.3 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 [376](#) of file [cellhandler.cpp](#).

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

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

5.2.4.4 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 174 of file [cellhandler.cpp](#).

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

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

5.2.4.5 `getCell()`

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

Access the cell to the given position.

Definition at line 108 of file [cellhandler.cpp](#).

References [m_cells](#).

5.2.4.6 `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 437 of file [cellhandler.cpp](#).

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

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


```

 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 317 of file [cellhandler.cpp](#).

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

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

5.2.4.9 nextStates()

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

Valid the state of all cells.

Definition at line 117 of file [cellhandler.cpp](#).

References [m_cells](#).

5.2.4.10 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 407 of file [cellhandler.cpp](#).

References [m_dimensions](#).

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

5.2.4.11 print()

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

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

Parameters

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

Definition at line 243 of file [cellhandler.cpp](#).

References [m_cells](#), and [m_dimensions](#).

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

5.2.4.12 save()

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

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 133 of file [cellhandler.cpp](#).

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

5.2.5 Member Data Documentation

5.2.5.1 m_cells

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

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

Definition at line 91 of file [cellhandler.h](#).

Referenced by [CellHandler\(\)](#), [foundNeighbours\(\)](#), [generate\(\)](#), [getCell\(\)](#), [load\(\)](#), [nextStates\(\)](#), [CellHandler::iterator->operator*\(\)](#), [CellHandler::iterator::operator->\(\)](#), [print\(\)](#), and [~CellHandler\(\)](#).

5.2.5.2 m_dimensions

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

Vector of x dimensions.

Definition at line 90 of file [cellhandler.h](#).

Referenced by [CellHandler\(\)](#), [foundNeighbours\(\)](#), [generate\(\)](#), [getListNeighboursPositionsRecursive\(\)](#), [CellHandler::iterator::iterator\(\)](#), [load\(\)](#), [CellHandler::iterator::operator++\(\)](#), [positionIncrement\(\)](#), [print\(\)](#), and [save\(\)](#).

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

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

5.3 CellHandler::iterator Class Reference

Implementation of iterator design pattern.

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

Public Member Functions

- `iterator` (const `CellHandler` *handler)
Construct an initial iterator to browse the `CellHandler`.
- `iterator & operator++` ()
Increment the current position and handle dimension changes.
- `Cell * operator->` () const
Get the current cell.
- `Cell * operator*` () const
- `bool operator!=` (bool finished) const
- `unsigned int changedDimension` () const
Return the number of dimensions we change.

Private Attributes

- const `CellHandler` * `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

Implementation of iterator design pattern.

Example of use:

```
CellHandler handler("file.atc");
for (CellHandler::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 37 of file `cellhandler.h`.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 iterator()

```
CellHandler::iterator::iterator (
    const CellHandler * handler )
```

Construct an initial iterator to browse the `CellHandler`.

Parameters

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

Definition at line 519 of file [cellhandler.cpp](#).

References [CellHandler::m_dimensions](#), [m_position](#), and [m_zero](#).

5.3.3 Member Function Documentation

5.3.3.1 `changedDimension()`

```
unsigned int CellHandler::iterator::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 580 of file [cellhandler.cpp](#).

References [m_changedDimension](#).

Referenced by [operator!==\(\)](#).

5.3.3.2 `operator!==()`

```
bool CellHandler::iterator::operator!= (
    bool finished ) const [inline]
```

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

References [changedDimension\(\)](#), and [m_finished](#).

5.3.3.3 `operator*()`

```
Cell * CellHandler::iterator::operator* ( ) const
```

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

References [CellHandler::m_cells](#), [m_handler](#), and [m_position](#).

5.3.3.4 operator++()

```
CellHandler::iterator & CellHandler::iterator::operator++ ( )
```

Increment the current position and handle dimension changes.

Definition at line 533 of file [cellhandler.cpp](#).

References [m_changedDimension](#), [CellHandler::m_dimensions](#), [m_finished](#), [m_handler](#), [m_position](#), and [m_zero](#).

5.3.3.5 operator->()

```
Cell * CellHandler::iterator::operator-> ( ) const
```

Get the current cell.

Definition at line 561 of file [cellhandler.cpp](#).

References [CellHandler::m_cells](#), [m_handler](#), and [m_position](#).

5.3.4 Friends And Related Function Documentation

5.3.4.1 CellHandler

```
friend class CellHandler [friend]
```

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

5.3.5 Member Data Documentation

5.3.5.1 m_changedDimension

```
unsigned int CellHandler::iterator::m_changedDimension [private]
```

Save the number of dimension change.

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

Referenced by [changedDimension\(\)](#), and [operator++\(\)](#).

5.3.5.2 m_finished

```
bool CellHandler::iterator::m_finished = false [private]
```

If we reach the last position.

Definition at line 55 of file [cellhandler.h](#).

Referenced by [operator!=\(\)](#), and [operator++\(\)](#).

5.3.5.3 m_handler

```
const CellHandler* CellHandler::iterator::m_handler [private]
```

[CellHandler](#) to go through.

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

Referenced by [operator*\(\)](#), [operator++\(\)](#), and [operator->\(\)](#).

5.3.5.4 m_position

```
QVector<unsigned int> CellHandler::iterator::m_position [private]
```

Current position of the iterator.

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

Referenced by [iterator\(\)](#), [operator*\(\)](#), [operator++\(\)](#), and [operator->\(\)](#).

5.3.5.5 m_zero

```
QVector<unsigned int> CellHandler::iterator::m_zero [private]
```

Nul vector of the good dimension (depend of m_handler)

Definition at line 56 of file [cellhandler.h](#).

Referenced by [iterator\(\)](#), and [operator++\(\)](#).

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
00008 Cell::Cell(unsigned int state):
00009     m_state(state), m_nextState(state)
00010 {
00011 }
00012 }
00013
00022 void Cell::setState(unsigned int state)
00023 {
00024     m_nextState = state;
00025 }
00026
00033 void Cell::validState()
00034 {
00035     m_state = m_nextState;
00036 }
00037
00041 unsigned int Cell::getState() const
00042 {
00043     return m_state;
00044 }
00045
00052 bool Cell::addNeighbour(const Cell* neighbour)
00053 {
00054     if (m_neighbours.count(neighbour))
00055         return false;
00056     m_neighbours.push_back(neighbour);
00057     return true;
00058 }
00059
00063 QVector<const Cell*> Cell::getNeighbours() const
00064 {
00065     return m_neighbours;
00066 }
```

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     unsigned int getState() const;
00018
00019     bool addNeighbour(const Cell* neighbour);
00020     QVector<const Cell*> getNeighbours() const;
00021
00022 private:
00023     unsigned int m_state;
00024     unsigned int m_nextState;
00025
00026     QVector<const Cell*> m_neighbours;
00027 };
00028
00029 #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
00026 CellHandler::CellHandler(const QString filename)
00027 {
00028     QFile loadFile(filename);
00029     if (!loadFile.open(QIODevice::ReadOnly | QIODevice::Text)) {
00030         qWarning("Couldn't open given file.");
00031         throw QString(QObject::tr("Couldn't open given file"));
00032     }
00033
00034     QJsonParseError parseErr;
00035     QJsonDocument loadDoc(QJsonDocument::fromJson(loadFile.readAll(), &parseErr));
00036
00037
00038
00039     if (loadDoc.isNull() || loadDoc.isEmpty()) {
00040         qWarning() << "Could not read data : ";
00041         qWarning() << parseErr.errorString();
00042     }
00043
00044     // Loading 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
00066 CellHandler::CellHandler(const QVector<unsigned int> dimensions,
    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
00074     for (unsigned short i = 0; i < m_dimensions.size(); i++)
00075     {
00076         position.push_back(0);
00077         size *= m_dimensions.at(i);
00078     }
00079
00080
00081     // Creation of cells
00082     for (unsigned int j = 0; j < size; j++)
00083     {
00084         m_cells.insert(position, new Cell(0));
00085
00086         positionIncrement(position);
00087     }
00088
00089     if (type != empty)
00090         generate(type, stateMax, density);
00091
00092 }
00093
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
00108 Cell *CellHandler::getCell(const QVector<unsigned int> position) const
00109 {
00110     return m_cells.value(position);
00111 }
00112
00117 void CellHandler::nextStates()
00118 {
00119     for (QMap<QVector<unsigned int>, Cell* >::iterator it = m_cells.begin(); it !=
    m_cells.end(); ++it)
00120     {
00121         it.value()->validState();
00122     }
00123 }
00124
00133 bool CellHandler::save(QString filename)
00134 {
00135     QFile saveFile(filename);
00136     if (!saveFile.open(QIODevice::WriteOnly)) {
00137         qWarning("Couldn't create or open given file.");
00138         throw QString(QObject::tr("Couldn't create or open given file"));
00139     }
00140
00141     QJsonObject json;
00142     QString stringDimension;
00143     // Creation of the dimension string
00144     for (unsigned int i = 0; i < m_dimensions.size(); i++)
00145     {
00146         if (i != 0)
00147             stringDimension.push_back("x");
00148         stringDimension.push_back(QString::number(m_dimensions.at(i)));
00149     }
00150     json["dimensions"] = QJsonValue(stringDimension);
00151
00152     QJsonArray cells;
00153     for (CellHandler::iterator it = begin(); it != end(); ++it)
00154     {
00155         cells.append(QJsonValue((int)it->getState()));
00156     }
00157     json["cells"] = cells;
00158
00159

```

```

00160     QJsonDocument saveDoc(json);
00161     saveFile.write(saveDoc.toJson());
00162
00163     saveFile.close();
00164     return true;
00165 }
00166
00174 void CellHandler::generate(CellHandler::generationTypes
type, unsigned int stateMax, unsigned short density)
00175 {
00176     if (type == random)
00177     {
00178         QVector<unsigned int> position;
00179         for (unsigned short i = 0; i < m_dimensions.size(); i++)
00180         {
00181             position.push_back(0);
00182         }
00183         QRandomGenerator generator((float)qrand()*(float)time_t()/RAND_MAX);
00184         for (unsigned int j = 0; j < m_cells.size(); j++)
00185         {
00186             unsigned int state = 0;
00187             // 0 have (1-density)% of chance of being generate
00188             if (generator.generateDouble()*100.0 < density)
00189                 state = (float)generator.generateDouble()*(stateMax+1);
00190             if (state > stateMax)
00191                 state = stateMax;
00192             m_cells.value(position)->setState(state);
00193             m_cells.value(position)->validState();
00194
00195             positionIncrement(position);
00196         }
00197     }
00198     else if (type == symetric)
00199     {
00200         QVector<unsigned int> position;
00201         for (unsigned short i = 0; i < m_dimensions.size(); i++)
00202         {
00203             position.push_back(0);
00204         }
00205
00206         QRandomGenerator generator((float)qrand()*(float)time_t()/RAND_MAX);
00207         QVector<unsigned int> savedStates;
00208         for (unsigned int j = 0; j < m_cells.size(); j++)
00209         {
00210             if (j % m_dimensions.at(0) == 0)
00211                 savedStates.clear();
00212             if (j % m_dimensions.at(0) < (m_dimensions.at(0)+1) / 2)
00213             {
00214                 unsigned int state = 0;
00215                 // 0 have (1-density)% of chance of being generate
00216                 if (generator.generateDouble()*100.0 < density)
00217                     state = (float)generator.generateDouble()*(stateMax+1);
00218                 if (state > stateMax)
00219                     state = stateMax;
00220                 savedStates.push_back(state);
00221                 m_cells.value(position)->setState(state);
00222                 m_cells.value(position)->validState();
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)->setState(savedStates.at(i));
00228                 m_cells.value(position)->validState();
00229             }
00230             positionIncrement(position);
00231         }
00232     }
00233 }
00234
00235 }
00236 }
00237
00243 void CellHandler::print(std::ostream &stream)
00244 {
00245     QVector<unsigned int> position;
00246     for (unsigned short i = 0; i < m_dimensions.size(); i++)
00247     {
00248         position.push_back(0);
00249     }
00250     for (unsigned int j = 0; j < m_cells.size(); j++)
00251     {
00252         stream << m_cells.value(position)->getState() << " ";
00253         position.replace(0, position.at(0)+1);
00254         for (unsigned short i = 0; i < m_dimensions.size(); i++)
00255         {
00256             if (position.at(i) >= m_dimensions.at(i))

```

```

00257         {
00258             position.replace(i, 0);
00259             stream << std::endl;
00260             if (i + 1 != m_dimensions.size())
00261                 position.replace(i+1, position.at(i+1)+1);
00262         }
00263     }
00264 }
00265
00266 }
00267
00271 CellHandler::iterator CellHandler::begin()
00272 {
00273     return iterator(this);
00274 }
00275
00281 bool CellHandler::end()
00282 {
00283     return true;
00284 }
00285
00317 bool CellHandler::load(const QJsonObject &json)
00318 {
00319     if (!json.contains("dimensions") || !json["dimensions"].isString())
00320         return false;
00321
00322     // RegExp to validate dimensions field format : "10x10"
00323     QRegExpValidator dimensionValidator(QRegExp("[0-9]*x[0-9]*"));
00324     QString stringDimensions = json["dimensions"].toString();
00325     int pos = 0;
00326     if (dimensionValidator.validate(stringDimensions, pos) != QRegExpValidator::Acceptable)
00327         return false;
00328
00329     // Split of dimensions field : "10x10" => "10", "10"
00330     QRegExp rx("x");
00331     QStringList list = json["dimensions"].toString().split(rx, QString::SkipEmptyParts);
00332
00333     unsigned int product = 1;
00334     // Dimensions construction
00335     for (unsigned int i = 0; i < list.size(); i++)
00336     {
00337         product = product * list.at(i).toInt();
00338         m_dimensions.push_back(list.at(i).toInt());
00339     }
00340     if (!json.contains("cells") || !json["cells"].isArray())
00341         return false;
00342
00343     QJsonArray cells = json["cells"].toArray();
00344     if (cells.size() != product)
00345         return false;
00346
00347     QVector<unsigned int> position;
00348     // Set position vector to 0
00349     for (unsigned short i = 0; i < m_dimensions.size(); i++)
00350     {
00351         position.push_back(0);
00352     }
00353
00354     // Creation of cells
00355     for (unsigned int j = 0; j < cells.size(); j++)
00356     {
00357         if (!cells.at(j).isDouble())
00358             return false;
00359         if (cells.at(j).toDouble() < 0)
00360             return false;
00361         m_cells.insert(position, new Cell(cells.at(j).toDouble()));
00362
00363         positionIncrement(position);
00364     }
00365
00366     return true;
00367 }
00368
00376 void CellHandler::foundNeighbours()
00377 {
00378     QVector<unsigned int> currentPosition;
00379     // Set position vector to 0
00380     for (unsigned short i = 0; i < m_dimensions.size(); i++)
00381     {
00382         currentPosition.push_back(0);
00383     }
00384     // Modification of all the cells
00385     for (unsigned int j = 0; j < m_cells.size(); j++)
00386     {
00387         // Get the list of the neighbours positions
00388         // This function is recursive

```

```

00389         QVector<QVector<unsigned int> > listPosition(getListNeighboursPositions(
currentPosition));
00390
00391         // Adding neighbours
00392         for (unsigned int i = 0; i < listPosition.size(); i++)
00393             m_cells.value(currentPosition)->addNeighbour(m_cells.value(listPosition.at(i)));
00394
00395         positionIncrement(currentPosition);
00396     }
00397 }
00398
00407 void CellHandler::positionIncrement(QVector<unsigned int> &pos, unsigned int
value) const
00408 {
00409     pos.replace(0, pos.at(0) + value); // adding the value to the first digit
00410
00411     // Carry management
00412     for (unsigned short i = 0; i < m_dimensions.size(); i++)
00413     {
00414         if (pos.at(i) >= m_dimensions.at(i) && pos.at(i) <
m_dimensions.at(i)*2)
00415         {
00416             pos.replace(i, 0);
00417             if (i + 1 != m_dimensions.size())
00418                 pos.replace(i+1, pos.at(i+1)+1);
00419         }
00420         else if (pos.at(i) >= m_dimensions.at(i))
00421         {
00422             pos.replace(i, pos.at(i) - m_dimensions.at(i));
00423             if (i + 1 != m_dimensions.size())
00424                 pos.replace(i+1, pos.at(i+1)+1);
00425             i--;
00426         }
00427     }
00428 }
00429 }
00430
00437 QVector<QVector<unsigned int> >& CellHandler::getListNeighboursPositions
(const QVector<unsigned int> position) const
00438 {
00439     QVector<QVector<unsigned int> > *list = getListNeighboursPositionsRecursive
(position, position.size(), position);
00440     // We remove the position of the cell
00441     list->removeAll(position);
00442     return *list;
00443 }
00444
00479 QVector<QVector<unsigned int> >*
CellHandler::getListNeighboursPositionsRecursive(const
QVector<unsigned int> position, unsigned int dimension, QVector<unsigned int> lastAdd) const
00480 {
00481     if (dimension == 0) // Stop condition
00482     {
00483         QVector<QVector<unsigned int> > *list = new QVector<QVector<unsigned int> >;
00484         return list;
00485     }
00486     QVector<QVector<unsigned int> > *listPositions = new QVector<QVector<unsigned int> >;
00487
00488     QVector<unsigned int> modifiedPosition(lastAdd);
00489
00490     // "x_d - 1" tree
00491     if (modifiedPosition.at(dimension-1) != 0) // Avoid "negative" position
00492         modifiedPosition.replace(dimension-1, position.at(dimension-1) - 1);
00493     listPositions->append(*getListNeighboursPositionsRecursive(position,
dimension -1, modifiedPosition));
00494     if (!listPositions->count(modifiedPosition))
00495         listPositions->push_back(modifiedPosition);
00496
00497     // "x_d" tree
00498     modifiedPosition.replace(dimension-1, position.at(dimension-1));
00499     listPositions->append(*getListNeighboursPositionsRecursive(position,
dimension -1, modifiedPosition));
00500     if (!listPositions->count(modifiedPosition))
00501         listPositions->push_back(modifiedPosition);
00502
00503     // "x_d + 1" tree
00504     if (modifiedPosition.at(dimension -1) + 1 < m_dimensions.at(dimension-1)) // Avoid position
out of the cell space
00505         modifiedPosition.replace(dimension-1, position.at(dimension-1) +1);
00506     listPositions->append(*getListNeighboursPositionsRecursive(position,
dimension -1, modifiedPosition));
00507     if (!listPositions->count(modifiedPosition))
00508         listPositions->push_back(modifiedPosition);
00509
00510     return listPositions;
00511 }
00512 }

```

```

00513
00519 CellHandler::iterator::iterator(const CellHandler *handler):
00520     m_handler(handler), m_changedDimension(0)
00521 {
00522     // Initialisation of m_position
00523     for (unsigned short i = 0; i < handler->m_dimensions.size(); i++)
00524     {
00525         m_position.push_back(0);
00526     }
00527     m_zero = m_position;
00528 }
00529
00533 CellHandler::iterator &CellHandler::iterator::operator++
00534 ()
00535 {
00536     m_position.replace(0, m_position.at(0) + 1); // adding the value to the first digit
00537     m_changedDimension = 0;
00538     // Carry management
00539     for (unsigned short i = 0; i < m_handler->m_dimensions.size(); i++)
00540     {
00541         if (m_position.at(i) >= m_handler->m_dimensions.at(i))
00542         {
00543             m_position.replace(i, 0);
00544             m_changedDimension++;
00545             if (i + 1 != m_handler->m_dimensions.size())
00546                 m_position.replace(i+1, m_position.at(i+1)+1);
00547         }
00548     }
00549     // If we return to zero, we have finished
00550     if (m_position == m_zero)
00551         m_finished = true;
00552     return *this;
00553 }
00554
00561 Cell *CellHandler::iterator::operator->() const
00562 {
00563     return m_handler->m_cells.value(m_position);
00564 }
00565
00569 Cell *CellHandler::iterator::operator*() const
00570 {
00571     return m_handler->m_cells.value(m_position);
00572 }
00573
00580 unsigned int CellHandler::iterator::changedDimension() const
00581 {
00582     return m_changedDimension;
00583 }
00584

```

6.7 cellhandler.h File Reference

```

#include <QString>
#include <QFile>
#include <QJsonDocument>
#include <QtWidgets>
#include <QMap>
#include <QRegExpValidator>
#include "cell.h"

```

Classes

- class [CellHandler](#)
Cell container and cell generator.
- class [CellHandler::iterator](#)
Implementation of iterator design pattern.

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
00011 #include "cell.h"
00012
00013 class CellHandler
00014 {
00015 public:
00016     class iterator
00017     {
00018     friend class CellHandler;
00019     public:
00020         iterator(const CellHandler* handler);
00021
00022         iterator& operator++();
00023         Cell* operator->() const;
00024         Cell* operator*() const;
00025
00026         bool operator!=(const bool finished) const { return (m_finished != finished); }
00027         unsigned int changedDimension() const;
00028
00029     private:
00030         const CellHandler *m_handler;
00031         QVector<unsigned int> m_position;
00032         bool m_finished = false;
00033         QVector<unsigned int> m_zero;
00034         unsigned int m_changedDimension;
00035     };
00036
00037     enum generationTypes {
00038         empty,
00039         random,
00040         symetric
00041     };
00042
00043     CellHandler(const QString filename);
00044     CellHandler(const QVector<unsigned int> dimensions,
00045         generationTypes type = empty, unsigned int stateMax = 1, unsigned int density = 50);
00046     virtual ~CellHandler();
00047
00048     Cell* getCell(const QVector<unsigned int> position) const;
00049     void nextStates();
00050
00051     bool save(QString filename);
00052     void generate(generationTypes type, unsigned int stateMax = 1, unsigned short
00053         density = 50);
00054     void print(std::ostream &stream);
00055
00056     iterator begin();
00057     bool end();
00058
00059 private:
00060     bool load(const QJsonObject &json);
00061     void foundNeighbours();
00062     void positionIncrement(QVector<unsigned int> &pos, unsigned int value = 1) const;
00063     QVector<QVector<unsigned int> > *getListNeighboursPositionsRecursive
00064         (const QVector<unsigned int> position, unsigned int dimension, QVector<unsigned int> lastAdd) const;
00065     QVector<QVector<unsigned int> > &getListNeighboursPositions(const
00066         QVector<unsigned int> position) const;
00067
00068     QVector<unsigned int> m_dimensions;
00069     QMap<QVector<unsigned int>, Cell* > m_cells;
00070 };
00071
00072 #endif // CELLHANDLER_H

```

6.9 homepage.md File Reference

6.10 homepage.md

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

6.11 main.cpp File Reference

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

Functions

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

6.11.1 Function Documentation

6.11.1.1 main()

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

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

References [CellHandler::print\(\)](#), and [CellHandler::symetric](#).

6.12 main.cpp

```
00001 #include <QApplication>
00002 #include <QDebug>
00003 #include <iostream>
00004 #include "cell.h"
00005 #include "cellhandler.h"
00006
00007 int main(int argc, char * argv[])
00008 {
00009     QApplication app(argc, argv);
00010     CellHandler handler(QVector<unsigned int>{30,30},
00011         CellHandler::symetric, 5);
00011     handler.print(std::cout);
00012     handler.save("testSave.atc");
00013     return 0;
00014 }
```

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, 15
- addNeighbour
 - Cell, 10
- begin
 - CellHandler, 16
- Cell, 9
 - addNeighbour, 10
 - Cell, 10
 - getNeighbours, 10
 - getState, 10
 - m_neighbours, 11
 - m_nextState, 12
 - m_state, 12
 - setState, 11
 - validState, 11
- cell.cpp, 27
- cell.h, 27, 28
- CellHandler, 12
 - ~CellHandler, 15
 - begin, 16
 - CellHandler, 14, 15
 - CellHandler::iterator, 24
 - end, 16
 - foundNeighbours, 16
 - generate, 16
 - generationTypes, 14
 - getCell, 17
 - getListNeighboursPositions, 17
 - getListNeighboursPositionsRecursive, 17
 - load, 18
 - m_cells, 21
 - m_dimensions, 21
 - nextStates, 19
 - positionIncrement, 19
 - print, 20
 - save, 20
- CellHandler::iterator, 21
 - CellHandler, 24
 - changedDimension, 23
 - iterator, 22
 - m_changedDimension, 24
 - m_finished, 24
 - m_handler, 25
 - m_position, 25
 - m_zero, 25
 - operator!=, 23
 - operator*, 23
 - operator++, 23
 - operator->, 24
- cellhandler.cpp, 28
- cellhandler.h, 33, 34
- changedDimension
 - CellHandler::iterator, 23
- end
 - CellHandler, 16
- foundNeighbours
 - CellHandler, 16
- generate
 - CellHandler, 16
- generationTypes
 - CellHandler, 14
- getCell
 - CellHandler, 17
- getListNeighboursPositions
 - CellHandler, 17
- getListNeighboursPositionsRecursive
 - CellHandler, 17
- getNeighbours
 - Cell, 10
- getState
 - Cell, 10
- homepage.md, 34
- iterator
 - CellHandler::iterator, 22
- load
 - CellHandler, 18
- m_cells
 - CellHandler, 21
- m_changedDimension
 - CellHandler::iterator, 24
- m_dimensions
 - CellHandler, 21
- m_finished
 - CellHandler::iterator, 24
- m_handler
 - CellHandler::iterator, 25
- m_neighbours
 - Cell, 11
- m_nextState
 - Cell, 12

- m_position
 - CellHandler::iterator, [25](#)
- m_state
 - Cell, [12](#)
- m_zero
 - CellHandler::iterator, [25](#)
- main
 - main.cpp, [35](#)
- main.cpp, [35](#)
 - main, [35](#)
- nextStates
 - CellHandler, [19](#)
- operator!=
 - CellHandler::iterator, [23](#)
- operator*
 - CellHandler::iterator, [23](#)
- operator++
 - CellHandler::iterator, [23](#)
- operator->
 - CellHandler::iterator, [24](#)
- positionIncrement
 - CellHandler, [19](#)
- print
 - CellHandler, [20](#)
- README.md, [36](#)
- save
 - CellHandler, [20](#)
- setState
 - Cell, [11](#)
- validState
 - Cell, [11](#)