# AutoCell

Generated by Doxygen 1.8.13

# **Contents**

1	Mair	n Page			1
2	Pres	sentatio	n		3
3	Clas	ss Index			5
	3.1	Class I	List		 . 5
4	File	Index			7
	4.1	File Lis	st		 . 7
5	Clas	ss Docu	mentation	n	9
	5.1	Cell Cl	ass Refer	rence	 . 9
		5.1.1	Detailed	Description	 . 9
		5.1.2	Construc	ctor & 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
			E 1 4 9	m atata	10

ii CONTENTS

5.2	CellHa	ndler Class	s Reference	12
	5.2.1	Detailed I	Description	14
	5.2.2	Member I	Enumeration Documentation	14
		5.2.2.1	generationTypes	14
	5.2.3	Construc	tor & 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 I	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 I	Data Documentation	21
		5.2.5.1	m_cells	21
		5.2.5.2	m_dimensions	21
5.3	CellHa	ndler::itera	ttor Class Reference	21
	5.3.1	Detailed I	Description	22
	5.3.2	Construc	tor & Destructor Documentation	22
		5.3.2.1	iterator()	22
	5.3.3	Member I	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 A	And Related Function Documentation	24
		5.3.4.1	CellHandler	24
	5.3.5	Member I	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

CONTENTS

6	File I	Documentation Company of the Company	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
Inc	lex		37

# **Chapter 1**

# Main Page

To generate the Documentation, go in Documentation directory and run  ${\tt make}.$ 

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

2 Main Page

# Chapter 2

# **Presentation**

## What is AutoCell

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

4 Presentation

# **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	

6 Class Index

# **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.c	pp				 				 														28
cellhandler.h					 				 														33
main.cpp .					 				 														35

8 File Index

# **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 (

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

unsigned int state = 0)

#### **Parameters**

```
state 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()

Add a new neighbour to the Cell.

#### **Parameters**

neighbour	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 Cell Class Reference

#### 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**

state 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

е	empty	Only empty cells.
rar	ndom	Random cells.
syn	netric	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

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,
2,4,1,1,1,1,2,1,1,
0,0,0,0,0,0,2,2,2,2,
3,4,5,1,2,0,9,0,0,0,
1,2,0,0,0,0,1,2,3,2]
```

#### **Parameters**

filename   Json file which contains the description of all th
---

#### **Exceptions**

QString	Unreadable file
QString	Empty file
QString	Not valid file

Definition at line 26 of file cellhandler.cpp.

References foundNeighbours(), and load().

#### **5.2.3.2 CellHandler()** [2/2]

Construct a CellHandler of the given dimension.

If generationTypes is given, the CellHandler won't be empty.

#### **Parameters**

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

Definition at line 66 of file cellhandler.cpp.

References empty, generate(), m\_cells, m\_dimensions, and positionIncrement().

#### 5.2.3.3 $\sim$ 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*3^{\wedge}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 (
```

Replace Cell values by random values (symetric or not)

unsigned int stateMax = 1, unsigned short density = 50)

CellHandler::generationTypes type,

#### **Parameters**

type	Type of random generation	
stateMax	Generate states between 0 and stateMax	
density	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()

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**

position	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().

#### 5.2.4.7 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  $O(3^{\land}$  dimension) Piece of the tree:

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

#### **Parameters**

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

#### Returns

List of position

Definition at line 479 of file cellhandler.cpp.

References m\_dimensions.

Referenced by getListNeighboursPositions().

#### 5.2.4.8 load()

Load the config file in the CellHandler.

Exemple of a way to print cell states :

#### **Parameters**

json

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()

Increment the QVector given by the value choosen.

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

#### **Parameters**

pos	Position to increment	
value	Value to add, 1 by default	

Definition at line 407 of file cellhandler.cpp.

References m\_dimensions.

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

#### 5.2.4.11 print()

Print in the given stream the CellHandler.

#### **Parameters**

stream Stream to print into	1
-----------------------------	---

Definition at line 243 of file cellhandler.cpp.

References m\_cells, and m\_dimensions.

Referenced by main().

#### 5.2.4.12 save()

Save the  ${\ensuremath{\mathsf{CellHandler}}}\xspace$  current configuration in the file given.

#### **Parameters**

filename Path to the file
---------------------------

#### Returns

False if there was a problem

#### **Exceptions**

QString	Impossible to open the file
---------	-----------------------------

Definition at line 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(), Cell—Handler::iterator::iterator::iterator::iterator::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() << " ";
}</pre>
```

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()

Construct an initial iterator to browse the CellHandler.

#### **Parameters**

handler   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 changed  $\leftarrow$  Dimension 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.
```

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

- cellhandler.h
- · cellhandler.cpp

Referenced by iterator(), and operator++().

# **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):
       m_state(state), m_nextState(state)
00010 {
00011
00012 }
00013
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 }
00052 bool Cell::addNeighbour(const Cell* neighbour)
return false;
m_neighbours.push_back(neighbour);
00056
        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>
```

28 File Documentation

#### 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
00024
         unsigned int m_state;
          unsigned int m_nextState;
00025
          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);
           if (!loadFile.open(QIODevice::ReadOnly | QIODevice::Text)) {
    qWarning("Couldn't open given file.");
00029
00030
                throw QString(QObject::tr("Couldn't open given file"));
00031
00032
00033
00034
           QJsonParseError parseErr;
00035
           QJsonDocument loadDoc(QJsonDocument::fromJson(loadFile.readAll(), &parseErr));
00036
00037
00038
           if (loadDoc.isNull() || loadDoc.isEmpty()) {
    qWarning() << "Could not read data : ";</pre>
00039
00040
00041
                qWarning() << parseErr.errorString();</pre>
00042
00043
00044
           // Loadding of the json file
           if (!load(loadDoc.object()))
00045
00046
00047
                qWarning("File not valid");
```

6.6 cellhandler.cpp 29

```
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;
          QVector<unsigned int> position;
00069
00070
          unsigned int size = 1;
00071
00072
          // Set position vector to 0
00073
00074
          for (unsigned short i = 0; i < m dimensions.size(); i++)</pre>
00075
00076
              position.push_back(0);
00077
              size *= m_dimensions.at(i);
00078
          }
00079
00080
00081
          // Creation of cells
          for (unsigned int j = 0; j < size; j++)
00082
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(OString 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++)</pre>
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
          OJsonArray cells:
          for (CellHandler::iterator it = begin(); it != end(); ++it)
00153
00154
00155
              cells.append(QJsonValue((int)it->getState()));
00156
00157
          json["cells"] = cells;
00158
00159
```

30 File Documentation

```
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
               QVector<unsigned int> position;
for (unsigned short i = 0; i < m_dimensions.size(); i++)</pre>
00178
00179
00180
00181
                    position.push_back(0);
00182
00183
               QRandomGenerator generator((float)qrand()*(float)time_t()/RAND_MAX);
               for (unsigned int j = 0; j < m_cells.size(); j++)</pre>
00184
00185
               {
00186
                    unsigned int state = 0;
                    // 0 have (1-density)% of chance of being generate
00187
                    if (generator.generateDouble()*100.0 < density)</pre>
00188
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
               QVector<unsigned int> position;
for (unsigned short i = 0; i < m_dimensions.size(); i++)</pre>
00200
00201
00202
                   position.push_back(0);
00204
               }
00205
00206
               QRandomGenerator generator((float)qrand()*(float)time_t()/RAND_MAX);
               QVector<unsigned int> savedStates;
for (unsigned int j = 0; j < m_cells.size(); j++)</pre>
00207
00208
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)</pre>
00213
                    {
00214
                        unsigned int state = 0:
00215
                        // 0 have (1-density)% of chance of being generate
                        if (generator.generateDouble()*100.0 < density)</pre>
00216
00217
                             state = (float)generator.generateDouble()*(stateMax+1);
                        if (state > stateMax)
00218
00219
                            state = stateMax;
00220
                        savedStates.push_back(state);
00221
                        m cells.value(position) ->setState(state);
                        m_cells.value(position)->validState();
00223
                    }
00224
                    else
00225
      unsigned int i = savedStates.size() - (j % m_dimensions.at(0) - (m_dimensions.at(0)-1)/2 + (m_dimensions.at(0) % 2 == 0 ? 0 : 1));
00226
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++)</pre>
00247
00248
               position.push_back(0);
00249
00250
           for (unsigned int j = 0; j < m_cells.size(); j++)</pre>
00251
00252
               stream << m_cells.value(position)->getState() << " ";</pre>
00253
               position.replace(0, position.at(0)+1);
00254
               for (unsigned short i = 0; i < m_dimensions.size(); i++)</pre>
00255
               {
00256
                    if (position.at(i) >= m dimensions.at(i))
```

6.6 cellhandler.cpp 31

```
00257
                   {
00258
                        position.replace(i, 0);
                        stream << std::endl;
if (i + 1 != m_dimensions.size())</pre>
00259
00260
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
           \label{eq:QRegExpValidator} $$ QRegExp("([0-9]*x?)*")); $$
00324
           QString stringDimensions = json["dimensions"].toString();
00325
           int pos= 0;
00326
          if (dimensionValidator.validate(stringDimensions, pos) != ORegExpValidator::Acceptable)
00327
               return false;
00328
00329
           // Split of dimensions field : "10x10" => "10", "10"
00330
           QRegExp rx("x");
           QStringList list = json["dimensions"].toString().split(rx, QString::SkipEmptyParts);
00331
00332
00333
           unsigned int product = 1;
00334
           // Dimensions construction
00335
           for (unsigned int i = 0; i < list.size(); i++)</pre>
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
00342
          QJsonArray cells = json["cells"].toArray();
if (cells.size() != product)
00343
00344
00345
              return false;
00346
00347
           QVector<unsigned int> position;
00348
           // Set position vector to \ensuremath{\text{0}}
00349
           for (unsigned short i = 0; i < m_dimensions.size(); i++)</pre>
00350
00351
               position.push back(0);
00352
00353
00354
           // Creation of cells
00355
           for (unsigned int j = 0; j < cells.size(); j++)</pre>
00356
00357
               if (!cells.at(j).isDouble())
00358
                    return false;
00359
               if (cells.at(j).toDouble() < 0)</pre>
                    return false;
00360
00361
               m_cells.insert(position, new Cell(cells.at(j).toDouble()));
00362
00363
              positionIncrement(position);
00364
          }
00365
00366
          return true;
00367
00368 }
00369
00376 void CellHandler::foundNeighbours()
00377 {
00378
           QVector<unsigned int> currentPosition;
           // Set position vector to 0
for (unsigned short i = 0; i < m_dimensions.size(); i++)</pre>
00379
00380
00381
          {
00382
               currentPosition.push back(0);
00383
           ^{\prime} // Modification of all the cells
00384
00385
           for (unsigned int j = 0; j < m_cells.size(); j++)</pre>
00386
               // Get the list of the neighbours positions
00387
00388
               // This function is recursive
```

32 File Documentation

```
00389
                        QVector<QVector<unsigned int> > listPosition(getListNeighboursPositions(
          currentPosition));
00390
00391
                        // Adding neighbours
                        for (unsigned int i = 0; i < listPosition.size(); i++)</pre>
00392
                               m_cells.value(currentPosition)->addNeighbour(m_cells.value(listPosition.at(i)));
00393
00394
00395
                        positionIncrement(currentPosition);
00396
                 }
00397 }
00398
00407 void CellHandler::positionIncrement(OVector<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
                  for (unsigned short i = 0; i < m_dimensions.size(); i++)</pre>
00412
00413
00414
                         if (pos.at(i) >= m_dimensions.at(i) && pos.at(i) <</pre>
          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
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
                         OVector<OVector<unsigned int> > *list = new OVector<OVector<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
                 // "x d - 1" tree
00490
                 if (modifiedPosition.at(dimension-1) != 0) // Avoid "negative" position
00491
00492
                        modifiedPosition.replace(dimension-1, position.at(dimension-1) - 1);
                 listPositions->append(*getListNeighboursPositionsRecursive(position,
00493
            dimension -1, modifiedPosition));
00494
                 if (!listPositions->count(modifiedPosition))
00495
                        listPositions->push_back(modifiedPosition);
00496
00497
00498
                 modifiedPosition.replace(dimension-1, position.at(dimension-1));
00499
                  {\tt listPositions-> append} \ (*{\tt getListNeighboursPositionsRecursive}\ ({\tt position}, {\tt constant}) \ (*{\tt getListNeighboursPositionsRecursive}\ ({\tt position}, {\tt constant})) \ ({\tt position}, {\tt constant}) \ ({\tt po
            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))
                        listPositions->push_back(modifiedPosition);
00508
00509
00510
                 return listPositions;
00511
00512 }
```

```
00519 CellHandler::iterator::iterator(const CellHandler *handler):
00520
              m_handler(handler), m_changedDimension(0)
00521 {
          // Initialisation of m_position
for (unsigned short i = 0; i < handler->m_dimensions.size(); i++)
00522
00523
00524
00525
              m_position.push_back(0);
00526
00527
          m_zero = m_position;
00528 }
00529
00533 CellHandler::iterator &CellHandler::iterator::operator++
00534 {
00535
          m_position.replace(0, m_position.at(0) + 1); // adding the value to the first digit
00536
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
00550
          // If we return to zero, we have finished
00551
          if (m_position == m_zero)
00552
              m_finished = true;
00553
00554
          return *this;
00555
00556 }
00557
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 {
          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.

34 File Documentation

#### 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>
00011 #include "cell.h"
00012
00018 class CellHandler
00019 {
00020 public:
00037
          class iterator
          {
00039
               friend class CellHandler;
          public:
00040
00041
              iterator(const CellHandler* handler);
00042
00043
               iterator& operator++();
00044
               Cell* operator->() const;
00045
               Cell* operator*() const;
00046
00047
               bool operator!=(bool finished) const { return (m_finished != finished); }
00048
               unsigned int changedDimension() const;
00049
00051
00052
          private:
00053
               const CellHandler *m_handler;
               QVector<unsigned int> m_position;
bool m_finished = false;
00054
00055
00056
               QVector<unsigned int> m_zero;
00057
               unsigned int m_changedDimension;
00058
00059
00063
          enum generationTypes {
00064
              empty,
00065
               random.
00066
               symetric
00067
00068
00069
          CellHandler(const QString filename);
generationTypes type = empty, unsigned int stateMax = 1, unsigned int density = 50);
00071    virtual ~CellHandler();
00070
00072
00073
           Cell* getCell(const QVector<unsigned int> position) const;
00074
          void nextStates();
00075
00076
          bool save(QString filename);
00077
           \label{eq:condition} \mbox{void } \mbox{\tt generate} \mbox{\tt (generationTypes type, unsigned int stateMax = 1, unsigned short)}
      density = 50);
00078
          void print(std::ostream &stream);
00079
08000
           iterator begin();
00081
          bool end();
00082
00083 private:
00084
          bool load(const QJsonObject &json);
00085
          void foundNeighbours();
          void positionIncrement(QVector<unsigned int> &pos, unsigned int value = 1) const;
QVector<QVector<unsigned int> > *getListNeighboursPositionsRecursive
00086
00087
      (const QVector<unsigned int> position, unsigned int dimension, QVector<unsigned int> lastAdd) const;
00088
          QVector<QVector<unsigned int> > &getListNeighboursPositions(const
      QVector<unsigned int> position) const;
00089
00090
           QVector<unsigned int> m_dimensions;
00091
           QMap<QVector<unsigned int>, Cell* > m_cells;
00092 };
00094 #endif // CELLHANDLER_H
```

#### 6.9 homepage.md File Reference

6.10 homepage.md 35

## 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

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);
CellHandler::symetric, 5);
00011 handler print (2)
00010
             CellHandler handler (QVector<unsigned int>{30,30},
            handler.print(std::cout);
handler.save("testSave.atc");
00012
00013
             return 0;
00014 }
```

36 File Documentation

## 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 directely in Documentation directory ('docPdf.pdf').
```

# Index

$\sim$ CellHandler	operator*, 23
CellHandler, 15	operator++, 23
	operator->, 24
addNeighbour	cellhandler.cpp, 28
Cell, 10	cellhandler.h, 33, 34
	changedDimension
begin	CellHandler::iterator, 23
CellHandler, 16	,
	end
Cell, 9	CellHandler, 16
addNeighbour, 10	
Cell, 10	foundNeighbours
getNeighbours, 10	CellHandler, 16
getState, 10	,
m_neighbours, 11	generate
m_nextState, 12	CellHandler, 16
	generationTypes
m_state, 12	CellHandler, 14
setState, 11	getCell
validState, 11	CellHandler, 17
cell.cpp, 27	getListNeighboursPositions
cell.h, 27, 28	CellHandler, 17
CellHandler, 12	getListNeighboursPositionsRecursive
$\sim$ CellHandler, 15	-
begin, 16	CellHandler, 17
CellHandler, 14, 15	getNeighbours
CellHandler::iterator, 24	Cell, 10
end, 16	getState
foundNeighbours, 16	Cell, 10
generate, 16	1.04
generationTypes, 14	homepage.md, 34
getCell, 17	the make in
getListNeighboursPositions, 17	iterator
getListNeighboursPositionsRecursive, 17	CellHandler::iterator, 22
load, 18	land
m_cells, 21	load
m_dimensions, 21	CellHandler, 18
nextStates, 19	m collo
	m_cells
positionIncrement, 19	CellHandler, 21
print, 20	m_changedDimension
save, 20	CellHandler::iterator, 24
CellHandler::iterator, 21	m_dimensions
CellHandler, 24	CellHandler, 21
changedDimension, 23	m_finished
iterator, 22	CellHandler::iterator, 24
m_changedDimension, 24	m_handler
m_finished, 24	CellHandler::iterator, 25
m_handler, 25	m_neighbours
m_position, 25	Cell, 11
m_zero, 25	m_nextState
operator!=, 23	Cell, 12

38 INDEX

```
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 *\\
     Cell Handler :: iterator, {\color{red} 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
```