AutoCell

Generated by Doxygen 1.8.13

Contents

Chapter 1

Main Page

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

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

Chapter 2

CellHandler

2.1 Creation

Example of creation

Chapter 3

Presentation

What is AutoCell

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Class Documentation

6.1 Cell Class Reference

Contains the state, the next state and the neighbours.

`#include <cell.h>`

Public Member Functions

- Cell (unsigned int state=0)

  Constructs a cell with the state given. State 0 is dead state.
- void setState (unsigned int state)

  Set temporary state.
- void validState ()

  Validate temporary state.
- void forceState (unsigned int state)

  Force the state change.
- unsigned int getState () const

  Access current cell state.
- bool addNeighbour (const Cell *neighbour)

  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.

6.1.1 Detailed Description

Contains the state, the next state and the neighbours.

Definition at line 10 of file cell.h.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 Cell()

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

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

Parameters

state	Cell state, dead state by default

Definition at line 8 of file cell.cpp.

6.1.3 Member Function Documentation

6.1.3.1 addNeighbour()

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

Add a new neighbour to the Cell.

Parameters

neighbour	New neighbour

Returns

False if the neighbour already exists

Definition at line 64 of file cell.cpp.

References m_neighbours.

6.1.3.2 forceState()

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

Force the state change.

Is equivalent to setState followed by validState

Parameters

state	New state

Definition at line 45 of file cell.cpp.

References m_nextState, and m_state.

6.1.3.3 getNeighbours()

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

Access neighbours list.

Definition at line 75 of file cell.cpp.

References m_neighbours.

6.1.3.4 getState()

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

Access current cell state.

Definition at line 53 of file cell.cpp.

References m_state.

6.1.3.5 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.

6.1.3.6 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.

6.1.4 Member Data Documentation

6.1.4.1 m_neighbours

```
QVector<const Cell*> Cell::m_neighbours  [private]
```

Cell's neighbours.

Definition at line 27 of file cell.h.

Referenced by addNeighbour(), and getNeighbours().

6.1.4.2 m_nextState

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

Temporary state, before validation.

Definition at line 25 of file cell.h.

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

6.1.4.3 m_state

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

Current state.

Definition at line 24 of file cell.h.

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

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

- cell.h
- cell.cpp

6.2 CellHandler Class Reference

Cell container and cell generator.

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

Classes

- class 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 stateMax=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.
- 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.

6.2.1 Detailed Description

Cell container and cell generator.

Generate cells from a json file.

Definition at line 18 of file cellhandler.h.

6.2.2 Member Enumeration Documentation

6.2.2.1 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.

6.2.3 Constructor & Destructor Documentation

6.2.3.1 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

filename	Json file which contains the description of all the cells

Exceptions

QString	Unreadable file
QString	Empty file
QString	Not valid file

Definition at line 26 of file cellhandler.cpp.

References foundNeighbours(), and load().

6.2.3.2 CellHandler() [2/2]

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

Construct a CellHandler of the given dimension.

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

Parameters

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

Definition at line 66 of file cellhandler.cpp.

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

6.2.3.3 ~CellHandler()

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

Destroys all cells in the CellHandler.

Definition at line 97 of file cellhandler.cpp.

References m_cells.

6.2.4 Member Function Documentation

6.2.4.1 begin()

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

Give the iterator which corresponds to the current CellHandler.

Definition at line 256 of file cellhandler.cpp.

Referenced by print(), and save().

6.2.4.2 end()

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

End condition of the iterator.

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

Definition at line 266 of file cellhandler.cpp.

Referenced by print(), and save().

6.2.4.3 foundNeighbours()

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

Set the neighbours of each cells.

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

Definition at line 361 of file cellhandler.cpp.

References getListNeighboursPositions(), m_cells, m_dimensions, and positionIncrement().

Referenced by CellHandler().

6.2.4.4 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

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

Definition at line 175 of file cellhandler.cpp.

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

Referenced by CellHandler().

6.2.4.5 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.

6.2.4.6 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 422 of file cellhandler.cpp.

References getListNeighboursPositionsRecursive().

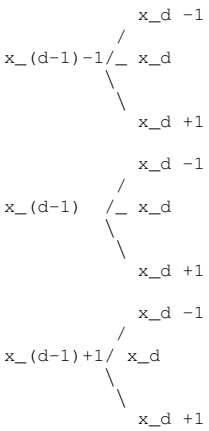
Referenced by foundNeighbours().

6.2.4.7 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^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 464 of file cellhandler.cpp.

References m_dimensions.

Referenced by getListNeighboursPositions().

6.2.4.8 load()

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

Load the config file in the CellHandler.

Exemple of a way to print cell states :

```
QVector<unsigned int> position;
for (unsigned short i = 0; i < m_dimensions.size(); i++)
{
     position.push_back(0);
}
for (unsigned int j = 0; j < m_cells.size(); j++)
{
     std::cout << m_cells.value(position)->getState() << " ";
     position.replace(0, position.at(0)+1);
     for (unsigned short i = 0; i < m_dimensions.size(); i++)
     {
         if (position.at(i) >= m_dimensions.at(i))
         {
             position.replace(i, 0);
             std::cout << std::endl;
             if (i + 1 != m_dimensions.size())
                 position.replace(i+1, position.at(i+1)+1);
         }

     }
}
```

Parameters

json	Json Object which contains the grid configuration

Returns

False if the Json Object is not correct

Definition at line 302 of file cellhandler.cpp.

References m_cells, m_dimensions, and positionIncrement().

Referenced by CellHandler().

6.2.4.9 nextStates()

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

Valid the state of all cells.

Definition at line 117 of file cellhandler.cpp.

References m_cells.

6.2.4.10 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

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

Definition at line 392 of file cellhandler.cpp.

References m_dimensions.

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

6.2.4.11 print()

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

Print in the given stream the CellHandler.

Parameters

stream	Stream to print into

Definition at line 241 of file cellhandler.cpp.

References begin(), and end().

Referenced by main().

6.2.4.12 save()

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

Save the CellHandler current configuration in the file given.

Parameters

filename	Path to the file

Returns

False if there was a problem

Exceptions

QString	Impossible to open the file

Definition at line 133 of file cellhandler.cpp.

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

6.2.5 Member Data Documentation

6.2.5.1 m_cells

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

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

Definition at line 92 of file cellhandler.h.

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

6.2.5.2 m_dimensions

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

Vector of x dimensions.

Definition at line 91 of file cellhandler.h.

Referenced by CellHandler(), foundNeighbours(), generate(), getListNeighboursPositionsRecursive(), CellHandler::iterator::iterator(), load(), CellHandler::iterator::operator++(), positionIncrement(), and save().

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

- cellhandler.h
- cellhandler.cpp

6.3 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

6.3.1 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.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 iterator()

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

Construct an initial iterator to browse the CellHandler.

Parameters

handler	CellHandler to browse

Definition at line 504 of file cellhandler.cpp.

References CellHandler::m_dimensions, m_position, and m_zero.

6.3.3 Member Function Documentation

6.3.3.1 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↩ Dimension return 2 (because of the 2 zeros).

Definition at line 565 of file cellhandler.cpp.

References m_changedDimension.

Referenced by operator!=().

6.3.3.2 operator"!=()

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

Definition at line 47 of file cellhandler.h.

References changedDimension(), and m_finished.

6.3.3.3 operator∗()

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

Definition at line 554 of file cellhandler.cpp.

References CellHandler::m_cells, m_handler, and m_position.

6.3.3.4 operator++()

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

Increment the current position and handle dimension changes.

Definition at line 518 of file cellhandler.cpp.

References m_changedDimension, CellHandler::m_dimensions, m_finished, m_handler, m_position, and m_zero.

6.3.3.5 operator->()

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

Get the current cell.

Definition at line 546 of file cellhandler.cpp.

References CellHandler::m_cells, m_handler, and m_position.

6.3.4 Friends And Related Function Documentation

6.3.4.1 CellHandler

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

Definition at line 39 of file cellhandler.h.

6.3.5 Member Data Documentation

6.3.5.1 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++().

6.3.5.2 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++().

6.3.5.3 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->().

6.3.5.4 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->().

6.3.5.5 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 7

File Documentation

7.1 cell.cpp File Reference

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

7.2 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
00045 void Cell::forceState(unsigned int state)
00046 {
00047     m_state = m_nextState = state;
00048 }
00049
00053 unsigned int Cell::getState() const
00054 {
00055     return m_state;
00056 }
00057
00064 bool Cell::addNeighbour(const Cell* neighbour)
00065 {
00066     if (m_neighbours.count(neighbour))
00067         return false;
00068     m_neighbours.push_back(neighbour);
00069     return true;
00070 }
00071
00075 QVector<const Cell*> Cell::getNeighbours() const
00076 {
00077     return m_neighbours;
00078 }
```

7.3 cell.h File Reference

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

Classes

- class Cell

  Contains the state, the next state and the neighbours.

7.4 cell.h

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

7.5 cellhandler.cpp File Reference

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

7.6 cellhandler.cpp

```
00001 #include <iostream>
00002 #include "cellhandler.h"
00003
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     // Loadding of the json file
00045     if (!load(loadDoc.object()))
00046     {
00047         qWarning("File not valid");
00048         throw QString(QObject::tr("File not valid"));
00049     }
00050
00051     foundNeighbours();
00052
00053
00054 }
00055
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 
00165     return true;
00166 }
00167 
00175 void CellHandler::generate(CellHandler::generationTypes
      type, unsigned int stateMax, unsigned short density)
00176 {
00177     if (type == random)
00178     {
00179         QVector<unsigned int> position;
00180         for (unsigned short i = 0; i < m_dimensions.size(); i++)
00181         {
00182             position.push_back(0);
00183         }
00184         QRandomGenerator generator((float)qrand()*(float)time_t()/RAND_MAX);
00185         for (unsigned int j = 0; j < m_cells.size(); j++)
00186         {
00187             unsigned int state = 0;
00188             // 0 have (1-density)% of chance of being generate
00189             if (generator.generateDouble()*100.0 < density)
00190                  state = (float)(generator.generateDouble()*stateMax) +1;
00191             if (state > stateMax)
00192                 state = stateMax;
00193             m_cells.value(position)->forceState(state);
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)->forceState(state);
00222             }
00223             else
00224             {
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                 m_cells.value(position)->forceState(savedStates.at(i));
00227             }
00228             positionIncrement(position);
00229 
00230 
00231         }
00232 
00233     }
00234 }
00235 
00241 void CellHandler::print(std::ostream &stream)
00242 {
00243     for (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
00256 CellHandler::iterator CellHandler::begin()
00257 {
00258     return iterator(this);
00259 }
00260
00266 bool CellHandler::end()
00267 {
00268     return true;
00269 }
00270
00302 bool CellHandler::load(const QJsonObject &json)
00303 {
00304     if (!json.contains("dimensions") || !json["dimensions"].isString())
00305         return false;
00306
00307     // RegExp to validate dimensions field format : "10x10"
00308     QRegExpValidator dimensionValidator(QRegExp("([0-9]*x?)*"));
00309     QString stringDimensions = json["dimensions"].toString();
00310     int pos= 0;
00311     if (dimensionValidator.validate(stringDimensions, pos) != QRegExpValidator::Acceptable)
00312         return false;
00313
00314     // Split of dimensions field : "10x10" => "10", "10"
00315     QRegExp rx("x");
00316     QStringList list = json["dimensions"].toString().split(rx, QString::SkipEmptyParts);
00317
00318     unsigned int product = 1;
00319     // Dimensions construction
00320     for (unsigned int i = 0; i < list.size(); i++)
00321     {
00322         product = product * list.at(i).toInt();
00323         m_dimensions.push_back(list.at(i).toInt());
00324     }
00325     if (!json.contains("cells") || !json["cells"].isArray())
00326         return false;
00327
00328     QJsonArray cells = json["cells"].toArray();
00329     if (cells.size() != product)
00330         return false;
00331
00332     QVector<unsigned int> position;
00333     // Set position vector to 0
00334     for (unsigned short i = 0; i < m_dimensions.size(); i++)
00335     {
00336         position.push_back(0);
00337     }
00338
00339     // Creation of cells
00340     for (unsigned int j = 0; j < cells.size(); j++)
00341     {
00342         if (!cells.at(j).isDouble())
00343             return false;
00344         if (cells.at(j).toDouble() < 0)
00345             return false;
00346         m_cells.insert(position, new Cell(cells.at(j).toDouble()));
00347
00348         positionIncrement(position);
00349     }
00350
00351     return true;
00352
00353 }
00354
00361 void CellHandler::foundNeighbours()
00362 {
00363     QVector<unsigned int> currentPosition;
00364     // Set position vector to 0
00365     for (unsigned short i = 0; i < m_dimensions.size(); i++)
00366     {
00367         currentPosition.push_back(0);
00368     }
00369     // Modification of all the cells
00370     for (unsigned int j = 0; j < m_cells.size(); j++)
00371     {
00372         // Get the list of the neighbours positions
00373         // This function is recursive
00374         QVector<QVector<unsigned int> > listPosition(getListNeighboursPositions(
      currentPosition));
```

```
00375
00376         // Adding neighbours
00377         for (unsigned int i = 0; i < listPosition.size(); i++)
00378             m_cells.value(currentPosition)->addNeighbour(m_cells.value(listPosition.at(i)));
00379
00380         positionIncrement(currentPosition);
00381     }
00382 }
00383
00392 void CellHandler::positionIncrement(QVector<unsigned int> &pos, unsigned int
      value) const
00393 {
00394     pos.replace(0, pos.at(0) + value); // adding the value to the first digit
00395
00396     // Carry management
00397     for (unsigned short i = 0; i < m_dimensions.size(); i++)
00398     {
00399         if (pos.at(i) >= m_dimensions.at(i) && pos.at(i) <
      m_dimensions.at(i)*2)
00400         {
00401             pos.replace(i, 0);
00402             if (i + 1 != m_dimensions.size())
00403                 pos.replace(i+1, pos.at(i+1)+1);
00404         }
00405         else if (pos.at(i) >= m_dimensions.at(i))
00406         {
00407             pos.replace(i, pos.at(i) - m_dimensions.at(i));
00408             if (i + 1 != m_dimensions.size())
00409                 pos.replace(i+1, pos.at(i+1)+1);
00410             i--;
00411         }
00412
00413     }
00414 }
00415
00422 QVector<QVector<unsigned int> >& CellHandler::getListNeighboursPositions
      (const QVector<unsigned int> position) const
00423 {
00424     QVector<QVector<unsigned int> > *list = getListNeighboursPositionsRecursive
      (position, position.size(), position);
00425     // We remove the position of the cell
00426     list->removeAll(position);
00427     return *list;
00428 }
00429
00464 QVector<QVector<unsigned int> >*
      CellHandler::getListNeighboursPositionsRecursive(const
      QVector<unsigned int> position, unsigned int dimension, QVector<unsigned int> lastAdd) const
00465 {
00466     if (dimension == 0) // Stop condition
00467     {
00468         QVector<QVector<unsigned int> > *list = new QVector<QVector<unsigned int> >;
00469         return list;
00470     }
00471     QVector<QVector<unsigned int> > *listPositions = new QVector<QVector<unsigned int> >;
00472
00473     QVector<unsigned int> modifiedPosition(lastAdd);
00474
00475     // "x_d - 1" tree
00476     if (modifiedPosition.at(dimension-1) != 0) // Avoid "negative" position
00477         modifiedPosition.replace(dimension-1, position.at(dimension-1) - 1);
00478     listPositions->append(*getListNeighboursPositionsRecursive(position,
       dimension -1, modifiedPosition));
00479     if (!listPositions->count(modifiedPosition))
00480         listPositions->push_back(modifiedPosition);
00481
00482     // "x_d" tree
00483     modifiedPosition.replace(dimension-1, position.at(dimension-1));
00484     listPositions->append(*getListNeighboursPositionsRecursive(position,
       dimension -1, modifiedPosition));
00485     if (!listPositions->count(modifiedPosition))
00486         listPositions->push_back(modifiedPosition);
00487
00488     // "x_d + 1" tree
00489     if (modifiedPosition.at(dimension -1) + 1 < m_dimensions.at(dimension-1)) // Avoid position
       out of the cell space
00490         modifiedPosition.replace(dimension-1, position.at(dimension-1) +1);
00491     listPositions->append(*getListNeighboursPositionsRecursive(position,
       dimension -1, modifiedPosition));
00492     if (!listPositions->count(modifiedPosition))
00493         listPositions->push_back(modifiedPosition);
00494
00495     return listPositions;
00496
00497 }
00498
00504 CellHandler::iterator::iterator(const CellHandler *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
00518 CellHandler::iterator &CellHandler::iterator::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
00546 Cell *CellHandler::iterator::operator->() const
00547 {
00548     return m_handler->m_cells.value(m_position);
00549 }
00550
00554 Cell *CellHandler::iterator::operator*() const
00555 {
00556     return m_handler->m_cells.value(m_position);
00557 }
00558
00565 unsigned int CellHandler::iterator::changedDimension() const
00566 {
00567     return m_changedDimension;
00568 }
00569
```

7.7 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.

7.8 cellhandler.h

```
00001 #ifndef CELLHANDLER_H
00002 #define CELLHANDLER_H
00003
00004 #include <QString>
00005 #include <QFile>
00006 #include <QJsonDocument>
00007 #include <QtWidgets>
00008 #include <QMap>
00009 #include <QRegExpValidator>
00010
00011 #include "cell.h"
00012
00018 class CellHandler
00019 {
00020 public:
00037     class iterator
00038     {
00039         friend class CellHandler;
00040     public:
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
00050
00051
00052     private:
00053         const CellHandler *m_handler;
00054         QVector<unsigned int> m_position;
00055         bool m_finished = false;
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
00070     CellHandler(const QString filename);
00071     CellHandler(const QVector<unsigned int> dimensions,
      generationTypes type = empty, unsigned int stateMax = 1, unsigned int density = 20);
00072     virtual ~CellHandler();
00073
00074     Cell* getCell(const QVector<unsigned int> position) const;
00075     void nextStates();
00076
00077     bool save(QString filename);
00078     void generate(generationTypes type, unsigned int stateMax = 1, unsigned short
      density = 50);
00079     void print(std::ostream &stream);
00080
00081     iterator begin();
00082     bool end();
00083
00084 private:
00085     bool load(const QJsonObject &json);
00086     void foundNeighbours();
00087     void positionIncrement(QVector<unsigned int> &pos, unsigned int value = 1) const;
00088     QVector<QVector<unsigned int> > *getListNeighboursPositionsRecursive
      (const QVector<unsigned int> position, unsigned int dimension, QVector<unsigned int> lastAdd) const;
00089     QVector<QVector<unsigned int> > &getListNeighboursPositions(const
      QVector<unsigned int> position) const;
00090
00091     QVector<unsigned int> m_dimensions;
00092     QMap<QVector<unsigned int>, Cell* > m_cells;
00093 };
00094
00095 #endif // CELLHANDLER_H
```

7.9 CellHandler.md File Reference

7.10 CellHandler.md

```
00001 \section Creation
00002
00003 Example of creation
```

7.11 main.cpp File Reference

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

Functions

- int main (int argc, char ∗argv[ ])

7.11.1 Function Documentation

7.11.1.1 main()

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

Definition at line 7 of file main.cpp.

References CellHandler::print(), and CellHandler::random.

7.12 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("testSave.atc");
00011     CellHandler handler(QVector<unsigned int>{10,10},
      CellHandler::random, 1, 20);
00012     handler.print(std::cout);
00013     //handler.save("testSave.atc");
00014     return 0;
00015 }
```

7.13 presentation.md File Reference

7.14 presentation.md

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

7.15 README.md File Reference

7.16 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