AutoCell

Generated by Doxygen 1.8.13

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 Cell Class Reference

```
#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.

3.1.1 Detailed Description

Definition at line 7 of file cell.h.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 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.

3.1.3 Member Function Documentation

3.1.3.1 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 52 of file cell.cpp.

References m_neighbours.

3.1.3.2 getNeighbours()

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

Access neighbours list.

Definition at line 63 of file cell.cpp.

References m_neighbours.

3.1.3.3 getState()

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

Access current cell state.

Definition at line 41 of file cell.cpp.

References m_state.

3.1.3.4 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.

3.1.3.5 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.

3.1.4 Member Data Documentation

3.1.4.1 m_neighbours

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

Cell's neighbours.

Definition at line 23 of file cell.h.

Referenced by addNeighbour(), and getNeighbours().

3.1.4.2 m_nextState

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

Temporary state, before validation.

Definition at line 21 of file cell.h.

Referenced by setState(), and validState().

3.1.4.3 m_state

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

Current state.

Definition at line 20 of file cell.h.

Referenced by getState(), and validState().

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

- cell.h
- cell.cpp

3.2 CellHandler Class Reference

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

Public Member Functions

- CellHandler (QString filename)

  Construct all the cells from the json file given.
- virtual ∼CellHandler ()

  Destroys all cells in the CellHandler.
- Cell ∗ getCell (const QVector< unsigned int > position) const

  Access the cell to the given position.

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.

3.2.1 Detailed Description

Definition at line 13 of file cellhandler.h.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 CellHandler()

```
CellHandler::CellHandler (
               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

Definition at line 23 of file cellhandler.cpp.

References foundNeighbours(), and load().

3.2.2.2 ∼CellHandler()

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

Destroys all cells in the CellHandler.

Definition at line 55 of file cellhandler.cpp.

References m_cells.

3.2.3 Member Function Documentation

3.2.3.1 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 161 of file cellhandler.cpp.

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

Referenced by CellHandler().

3.2.3.2 getCell()

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

Access the cell to the given position.

Definition at line 66 of file cellhandler.cpp.

References m_cells.

3.2.3.3 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 222 of file cellhandler.cpp.

References getListNeighboursPositionsRecursive().

Referenced by foundNeighbours().

3.2.3.4 getListNeighboursPositionsRecursive()

```
QVector< QVector< unsigned int > > * CellHandler::getListNeighboursPositionsRecursive (
               const QVector< unsigned int > position,
               unsigned int dimension,
               QVector< unsigned int > lastAdd ) const  [private]
```

Recursive function which browse the position possibilities tree.

Careful, the complexity is in O(3^dimension)
Piece of the tree:

```
            x_d -1
          /
x_(d-1)-1/_ x_d
         \
          \
            x_d +1

            x_d -1
          /
x_(d-1)  /_ x_d
         \
          \
            x_d +1

            x_d -1
          /
x_(d-1)+1/ x_d
         \
          \
            x_d +1
```

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

Parameters

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

Returns

List of position

Definition at line 264 of file cellhandler.cpp.

References m_dimensions.

Referenced by getListNeighboursPositions().

3.2.3.5 load()

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

Load the config file in the CellHandler.

Exemple of a way to print cell states :

```
position.clear();
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 102 of file cellhandler.cpp.

References m_cells, m_dimensions, and positionIncrement().

Referenced by CellHandler().

3.2.3.6 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 192 of file cellhandler.cpp.

References m_dimensions.

Referenced by foundNeighbours(), and load().

3.2.4 Member Data Documentation

3.2.4.1 m_cells

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

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

Definition at line 29 of file cellhandler.h.

Referenced by foundNeighbours(), getCell(), load(), and ∼CellHandler().

3.2.4.2 m_dimensions

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

Vector of x dimensions.

Definition at line 28 of file cellhandler.h.

Referenced by foundNeighbours(), getListNeighboursPositionsRecursive(), load(), and positionIncrement().

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

- cellhandler.h
- cellhandler.cpp

Chapter 4

File Documentation

4.1 cell.cpp File Reference

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

4.2 cell.cpp

```
#include "cell.h"

Cell::Cell(unsigned int state):
    m_state(state), m_nextState(state)
{

}

void Cell::setState(unsigned int state)
{
    m_nextState = state;
}

void Cell::validState()
{
    m_state = m_nextState;
}

unsigned int Cell::getState() const
{
    return m_state;
}

bool Cell::addNeighbour(const Cell* neighbour)
{
    if (m_neighbours.count(neighbour))
        return false;
    m_neighbours.push_back(neighbour);
    return true;
}

QVector<const Cell*> Cell::getNeighbours() const
{
    return m_neighbours;
}
```

4.3 cell.h File Reference

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

Classes

- class Cell

4.4 cell.h

```
#ifndef CELL_H
#define CELL_H

#include <QVector>
#include <QDebug>

class Cell
{
public:
    Cell(unsigned int state = 0);

    void setState(unsigned int state);
    void validState();
    unsigned int getState() const;

    bool addNeighbour(const Cell* neighbour);
    QVector<const Cell*> getNeighbours() const;

private:
    unsigned int m_state;
    unsigned int m_nextState;

    QVector<const Cell*> m_neighbours;
};

#endif // CELL_H
```

4.5 cellhandler.cpp File Reference

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

4.6 cellhandler.cpp

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

CellHandler::CellHandler(QString filename)
{
    QFile loadFile(filename);
    if (!loadFile.open(QIODevice::ReadOnly | QIODevice::Text)) {
        qWarning("Couldn't open given file.");
        throw QString(QObject::tr("Couldn't open given file"));
    }

    QJsonParseError parseErr;
    QJsonDocument loadDoc(QJsonDocument::fromJson(loadFile.readAll(), &parseErr));



    if (loadDoc.isNull() || loadDoc.isEmpty()) {
        qWarning() << "Could not read data : ";
        qWarning() << parseErr.errorString();
    }

    // Loadding of the json file
    if (!load(loadDoc.object()))
    {
        qWarning("File not valid");
        throw QString(QObject::tr("File not valid"));
    }

    foundNeighbours();

}

CellHandler::~CellHandler()
{
    for (QMap<QVector<unsigned int>, Cell* >::iterator it = m_cells.begin(); it != m_cells.end(); ++it)
    {
        delete it.value();
    }
}

Cell *CellHandler::getCell(const QVector<unsigned int> position) const
{
    return m_cells.value(position);
}

bool CellHandler::load(const QJsonObject &json)
{
    if (!json.contains("dimensions") || !json["dimensions"].isString())
        return false;

    // RegExp to validate dimensions field format : "10x10"
    QRegExpValidator dimensionValidator(QRegExp("([0-9]*x?)*"));
    QString stringDimensions = json["dimensions"].toString();
    int pos= 0;
    if (dimensionValidator.validate(stringDimensions, pos) != QRegExpValidator::Acceptable)
        return false;

    // Split of dimensions field : "10x10" => "10", "10"
    QRegExp rx("x");
    QStringList list = json["dimensions"].toString().split(rx, QString::SkipEmptyParts);

    unsigned int product = 1;
    // Dimensions construction
    for (unsigned int i = 0; i < list.size(); i++)
    {
        product = product * list.at(i).toInt();
        m_dimensions.push_back(list.at(i).toInt());
    }
    if (!json.contains("cells") || !json["cells"].isArray())
        return false;

    QJsonArray cells = json["cells"].toArray();
    if (cells.size() != product)
        return false;

    QVector<unsigned int> position;
    // Set position vector to 0
    for (unsigned short i = 0; i < m_dimensions.size(); i++)
    {
        position.push_back(0);
    }

    // Creation of cells
    for (unsigned int j = 0; j < cells.size(); j++)
    {
        if (!cells.at(j).isDouble())
            return false;
        if (cells.at(j).toDouble() < 0)
            return false;
        m_cells.insert(position, new Cell(cells.at(j).toDouble()));

        positionIncrement(position);
    }

    return true;

}

void CellHandler::foundNeighbours()
{
    QVector<unsigned int> currentPosition;
    // Set position vector to 0
    for (unsigned short i = 0; i < m_dimensions.size(); i++)
    {
        currentPosition.push_back(0);
    }
    // Modification of all the cells
    for (unsigned int j = 0; j < m_cells.size(); j++)
    {
        // Get the list of the neighbours positions
        // This function is recursive
        QVector<QVector<unsigned int> > listPosition(getListNeighboursPositions(currentPosition));

        // Adding neighbours
        for (unsigned int i = 0; i < listPosition.size(); i++)
            m_cells.value(currentPosition)->addNeighbour(m_cells.value(listPosition.at(i)));

        positionIncrement(currentPosition);
    }
}

void CellHandler::positionIncrement(QVector<unsigned int> &pos, unsigned int value) const
{
    pos.replace(0, pos.at(0) + value); // adding the value to the first digit

    // Carry management
    for (unsigned short i = 0; i < m_dimensions.size(); i++)
    {
        if (pos.at(i) >= m_dimensions.at(i) && pos.at(i) < m_dimensions.at(i)*2)
        {
            pos.replace(i, 0);
            if (i + 1 != m_dimensions.size())
                pos.replace(i+1, pos.at(i+1)+1);
        }
        else if (pos.at(i) >= m_dimensions.at(i))
        {
            pos.replace(i, pos.at(i) - m_dimensions.at(i));
            if (i + 1 != m_dimensions.size())
                pos.replace(i+1, pos.at(i+1)+1);
            i--;
        }

    }
}

QVector<QVector<unsigned int> >& CellHandler::getListNeighboursPositions(const QVector<unsigned int> position) const
{
    QVector<QVector<unsigned int> > *list = getListNeighboursPositionsRecursive(position, position.size(), position);
    // We remove the position of the cell
    list->removeAll(position);
    return *list;
}

QVector<QVector<unsigned int> >* CellHandler::getListNeighboursPositionsRecursive(const QVector<unsigned int> position, unsigned int dimension, QVector<unsigned int> lastAdd) const
{
    if (dimension == 0)
    {
        QVector<QVector<unsigned int> > *list = new QVector<QVector<unsigned int> >;
        return list;
    }
    QVector<QVector<unsigned int> > *listPositions = new QVector<QVector<unsigned int> >;

    QVector<unsigned int> modifiedPosition(lastAdd);

    // "x_d - 1" tree
    if (modifiedPosition.at(dimension-1) != 0) // Avoid "negative" position
        modifiedPosition.replace(dimension-1, position.at(dimension-1) - 1);
    listPositions->append(*getListNeighboursPositionsRecursive(position, dimension -1, modifiedPosition));
    if (!listPositions->count(modifiedPosition))
        listPositions->push_back(modifiedPosition);

    // "x_d" tree
    modifiedPosition.replace(dimension-1, position.at(dimension-1));
    listPositions->append(*getListNeighboursPositionsRecursive(position, dimension -1, modifiedPosition));
    if (!listPositions->count(modifiedPosition))
        listPositions->push_back(modifiedPosition);

    // "x_d + 1" tree
    if (modifiedPosition.at(dimension -1) + 1 < m_dimensions.at(dimension-1)) // Avoid position out of the cell space
        modifiedPosition.replace(dimension-1, position.at(dimension-1) +1);
    listPositions->append(*getListNeighboursPositionsRecursive(position, dimension -1, modifiedPosition));
    if (!listPositions->count(modifiedPosition))
        listPositions->push_back(modifiedPosition);

    return listPositions;

}
```

4.7 cellhandler.h File Reference

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

Classes

- class CellHandler

4.8 cellhandler.h

```
#ifndef CELLHANDLER_H
#define CELLHANDLER_H

#include <QString>
#include <QFile>
#include <QJsonDocument>
#include <QtWidgets>
#include <QMap>
#include <QRegExpValidator>

#include "cell.h"

class CellHandler
{
public:
    CellHandler(QString filename);
    virtual ~CellHandler();

    Cell* getCell(const QVector<unsigned int> position) const;

private:
    bool load(const QJsonObject &json);
    void foundNeighbours();
    void positionIncrement(QVector<unsigned int> &pos, unsigned int value = 1) const;
    QVector<QVector<unsigned int> > *getListNeighboursPositionsRecursive(const QVector<unsigned int> position, unsigned int dimension, QVector<unsigned int> lastAdd) const;
    QVector<QVector<unsigned int> > &getListNeighboursPositions(const QVector<unsigned int> position) const;

    QVector<unsigned int> m_dimensions;
    QMap<QVector<unsigned int>, Cell* > m_cells;
};

#endif // CELLHANDLER_H
```

4.9 main.cpp File Reference

```
#include <QApplication>
#include <QDebug>
#include "cellhandler.h"
#include <QFileDialog>
```

Functions

- int main (int argc, char ∗argv[ ])

4.9.1 Function Documentation

4.9.1.1 main()

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

Definition at line 7 of file main.cpp.

4.10 main.cpp

```
00001 #include <QApplication>
00002 #include <QDebug>
00003 #include "cellhandler.h"
00004
00005 #include <QFileDialog>
00006
00007 int main(int argc, char * argv[])
00008 {
00009     QApplication app(argc, argv);
00010
00011     CellHandler handler("test.atc");
00012     //return app.exec();
00013     return 0;
00014 }
```

Index