

AutoCell

Generated by Doxygen 1.8.13

Contents

1	Class Index	1
1.1	Class List	1
2	File Index	3
2.1	File List	3
3	Class Documentation	5
3.1	Cell Class Reference	5
3.1.1	Detailed Description	5
3.1.2	Constructor & Destructor Documentation	6
3.1.2.1	Cell()	6
3.1.3	Member Function Documentation	6
3.1.3.1	addNeighbour()	6
3.1.3.2	getNeighbours()	6
3.1.3.3	getState()	7
3.1.3.4	setState()	7
3.1.3.5	validState()	7
3.1.4	Member Data Documentation	7
3.1.4.1	m_neighbours	8
3.1.4.2	m_nextState	8
3.1.4.3	m_state	8
3.2	CellHandler Class Reference	8
3.2.1	Detailed Description	9
3.2.2	Constructor & Destructor Documentation	9

3.2.2.1	CellHandler()	9
3.2.2.2	~CellHandler()	10
3.2.3	Member Function Documentation	10
3.2.3.1	foundNeighbours()	10
3.2.3.2	getCell()	10
3.2.3.3	getListNeighboursPositions()	10
3.2.3.4	getListNeighboursPositionsRecursive()	11
3.2.3.5	load()	12
3.2.3.6	positionIncrement()	13
3.2.4	Member Data Documentation	13
3.2.4.1	m_cells	13
3.2.4.2	m_dimensions	13
4	File Documentation	15
4.1	cell.cpp File Reference	15
4.2	cell.cpp	15
4.3	cell.h File Reference	15
4.4	cell.h	16
4.5	cellhandler.cpp File Reference	16
4.6	cellhandler.cpp	16
4.7	cellhandler.h File Reference	19
4.8	cellhandler.h	19
4.9	main.cpp File Reference	19
4.9.1	Function Documentation	20
4.9.1.1	main()	20
4.10	main.cpp	20
Index		21

Chapter 1

Class Index

1.1 Class List

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

Cell	5
CellHandler	8

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

cell.cpp	15
cell.h	15
cellhandler.cpp	16
cellhandler.h	19
main.cpp	19

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

<i>state</i>	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

<i>neighbour</i>	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

<i>state</i>	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, 0, 1, 2, 3, 2]
}
```

Parameters

<i>filename</i>	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 $O(n \cdot 3^d)$, 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

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

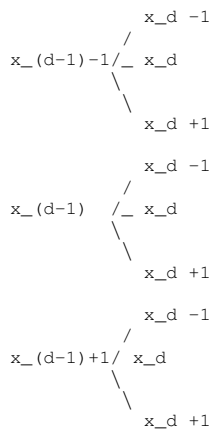
3.2.3.4 getListNeighboursPositionsRecursive()

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

Piece of the tree:



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

Parameters

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

Returns

List of position

Definition at line 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

<i>json</i>	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

<i>pos</i>	Position to increment
<i>value</i>	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

```
00001 #include "cell.h"
00002
00008 Cell::Cell(unsigned int state):
00009     m_state(state), m_nextState(state)
00010 {
00011
00012 }
00013
00022 void Cell::setState(unsigned int state)
00023 {
00024     m_nextState = state;
00025 }
00026
00033 void Cell::validState()
00034 {
00035     m_state = m_nextState;
00036 }
00037
00041 unsigned int Cell::getState() const
00042 {
00043     return m_state;
00044 }
00045
00052 bool Cell::addNeighbour(const Cell* neighbour)
00053 {
00054     if (m_neighbours.count(neighbour))
00055         return false;
00056     m_neighbours.push_back(neighbour);
00057     return true;
00058 }
00059
00063 QVector<const Cell*> Cell::getNeighbours() const
00064 {
00065     return m_neighbours;
00066 }
```

4.3 cell.h File Reference

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

Classes

- class [Cell](#)

4.4 cell.h

```

00001 #ifndef CELL_H
00002 #define CELL_H
00003
00004 #include <QVector>
00005 #include <QDebug>
00006
00007 class Cell
00008 {
00009 public:
00010     Cell(unsigned int state = 0);
00011
00012     void setState(unsigned int state);
00013     void validState();
00014     unsigned int getState() const;
00015
00016     bool addNeighbour(const Cell* neighbour);
00017     QVector<const Cell*> getNeighbours() const;
00018
00019 private:
00020     unsigned int m_state;
00021     unsigned int m_nextState;
00022
00023     QVector<const Cell*> m_neighbours;
00024 };
00025
00026 #endif // CELL_H

```

4.5 cellhandler.cpp File Reference

```

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

```

4.6 cellhandler.cpp

```

00001 #include <iostream>
00002 #include "cellhandler.h"
00003
00023 CellHandler::CellHandler(QString filename)
00024 {
00025     QFile loadFile(filename);
00026     if (!loadFile.open(QIODevice::ReadOnly | QIODevice::Text)) {
00027         qWarning("Couldn't open given file.");
00028         throw QString(QObject::tr("Couldn't open given file"));
00029     }
00030
00031     QJsonParseError parseErr;
00032     QJsonDocument loadDoc(QJsonDocument::fromJson(loadFile.readAll(), &parseErr));
00033
00034
00035
00036     if (loadDoc.isNull() || loadDoc.isEmpty()) {
00037         qWarning() << "Could not read data : ";
00038         qWarning() << parseErr.errorString();
00039     }
00040
00041     // Loading of the json file
00042     if (!load(loadDoc.object()))
00043     {
00044         qWarning("File not valid");
00045         throw QString(QObject::tr("File not valid"));
00046     }

```

```

00047
00048     foundNeighbours();
00049
00050 }
00051
00055 CellHandler::~CellHandler()
00056 {
00057     for (QMap<QVector<unsigned int>, Cell* >::iterator it = m_cells.begin(); it !=
00058         m_cells.end(); ++it)
00059     {
00060         delete it.value();
00061     }
00062 }
00066 Cell *CellHandler::getCell(const QVector<unsigned int> position) const
00067 {
00068     return m_cells.value(position);
00069 }
00070
00102 bool CellHandler::load(const QJsonObject &json)
00103 {
00104     if (!json.contains("dimensions") || !json["dimensions"].isString())
00105         return false;
00106
00107     // RegExp to validate dimensions field format : "10x10"
00108     QRegExpValidator dimensionValidator(QRegExp("[0-9]*x[0-9]*"));
00109     QString stringDimensions = json["dimensions"].toString();
00110     int pos = 0;
00111     if (dimensionValidator.validate(stringDimensions, pos) != QRegExpValidator::Acceptable)
00112         return false;
00113
00114     // Split of dimensions field : "10x10" => "10", "10"
00115     QRegExp rx("x");
00116     QStringList list = json["dimensions"].toString().split(rx, QString::SkipEmptyParts);
00117
00118     unsigned int product = 1;
00119     // Dimensions construction
00120     for (unsigned int i = 0; i < list.size(); i++)
00121     {
00122         product = product * list.at(i).toInt();
00123         m_dimensions.push_back(list.at(i).toInt());
00124     }
00125     if (!json.contains("cells") || !json["cells"].isArray())
00126         return false;
00127
00128     QJsonArray cells = json["cells"].toArray();
00129     if (cells.size() != product)
00130         return false;
00131
00132     QVector<unsigned int> position;
00133     // Set position vector to 0
00134     for (unsigned short i = 0; i < m_dimensions.size(); i++)
00135     {
00136         position.push_back(0);
00137     }
00138
00139     // Creation of cells
00140     for (unsigned int j = 0; j < cells.size(); j++)
00141     {
00142         if (!cells.at(j).isDouble())
00143             return false;
00144         if (cells.at(j).toDouble() < 0)
00145             return false;
00146         m_cells.insert(position, new Cell(cells.at(j).toDouble()));
00147
00148         positionIncrement(position);
00149     }
00150
00151     return true;
00152 }
00153
00161 void CellHandler::foundNeighbours()
00162 {
00163     QVector<unsigned int> currentPosition;
00164     // Set position vector to 0
00165     for (unsigned short i = 0; i < m_dimensions.size(); i++)
00166     {
00167         currentPosition.push_back(0);
00168     }
00169     // Modification of all the cells
00170     for (unsigned int j = 0; j < m_cells.size(); j++)
00171     {
00172         // Get the list of the neighbours positions
00173         // This function is recursive
00174         QVector<QVector<unsigned int> > listPosition(getListNeighboursPositions(
00175             currentPosition));

```

```

00175
00176     // Adding neighbours
00177     for (unsigned int i = 0; i < listPosition.size(); i++)
00178         m_cells.value(currentPosition)->addNeighbour(m_cells.value(listPosition.at(i)));
00179
00180     positionIncrement(currentPosition);
00181 }
00182 }
00183
00192 void CellHandler::positionIncrement(QVector<unsigned int> &pos, unsigned int
    value) const
00193 {
00194     pos.replace(0, pos.at(0) + value); // adding the value to the first digit
00195
00196     // Carry management
00197     for (unsigned short i = 0; i < m_dimensions.size(); i++)
00198     {
00199         if (pos.at(i) >= m_dimensions.at(i) && pos.at(i) <
m_dimensions.at(i)*2)
00200         {
00201             pos.replace(i, 0);
00202             if (i + 1 != m_dimensions.size())
00203                 pos.replace(i+1, pos.at(i+1)+1);
00204         }
00205         else if (pos.at(i) >= m_dimensions.at(i))
00206         {
00207             pos.replace(i, pos.at(i) - m_dimensions.at(i));
00208             if (i + 1 != m_dimensions.size())
00209                 pos.replace(i+1, pos.at(i+1)+1);
00210             i--;
00211         }
00212     }
00213 }
00214 }
00215
00222 QVector<QVector<unsigned int> >& CellHandler::getListNeighboursPositions
    (const QVector<unsigned int> position) const
00223 {
00224     QVector<QVector<unsigned int> > *list = getListNeighboursPositionsRecursive
    (position, position.size(), position);
00225     // We remove the position of the cell
00226     list->removeAll(position);
00227     return *list;
00228 }
00229
00264 QVector<QVector<unsigned int> >*
    CellHandler::getListNeighboursPositionsRecursive(const
    QVector<unsigned int> position, unsigned int dimension, QVector<unsigned int> lastAdd) const
00265 {
00266     if (dimension == 0)
00267     {
00268         QVector<QVector<unsigned int> > *list = new QVector<QVector<unsigned int> >;
00269         return list;
00270     }
00271     QVector<QVector<unsigned int> > *listPositions = new QVector<QVector<unsigned int> >;
00272
00273     QVector<unsigned int> modifiedPosition(lastAdd);
00274
00275     // "x_d - 1" tree
00276     if (modifiedPosition.at(dimension-1) != 0) // Avoid "negative" position
00277         modifiedPosition.replace(dimension-1, position.at(dimension-1) - 1);
00278     listPositions->append(*getListNeighboursPositionsRecursive(position,
    dimension - 1, modifiedPosition));
00279     if (!listPositions->count(modifiedPosition))
00280         listPositions->push_back(modifiedPosition);
00281
00282     // "x_d" tree
00283     modifiedPosition.replace(dimension-1, position.at(dimension-1));
00284     listPositions->append(*getListNeighboursPositionsRecursive(position,
    dimension - 1, modifiedPosition));
00285     if (!listPositions->count(modifiedPosition))
00286         listPositions->push_back(modifiedPosition);
00287
00288     // "x_d + 1" tree
00289     if (modifiedPosition.at(dimension - 1) + 1 < m_dimensions.at(dimension-1)) // Avoid position
    out of the cell space
00290         modifiedPosition.replace(dimension-1, position.at(dimension-1) + 1);
00291     listPositions->append(*getListNeighboursPositionsRecursive(position,
    dimension - 1, modifiedPosition));
00292     if (!listPositions->count(modifiedPosition))
00293         listPositions->push_back(modifiedPosition);
00294
00295     return listPositions;
00296 }
00297 }

```

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

```
00001 #ifndef CELLHANDLER_H
00002 #define CELLHANDLER_H
00003
00004 #include <QString>
00005 #include <QFile>
00006 #include <QJsonDocument>
00007 #include <QtWidgets>
00008 #include <QMap>
00009 #include <QRegExpValidator>
00010
00011 #include "cell.h"
00012
00013 class CellHandler
00014 {
00015 public:
00016     CellHandler(QString filename);
00017     virtual ~CellHandler();
00018
00019     Cell* getCell(const QVector<unsigned int> position) const;
00020
00021 private:
00022     bool load(const QJsonObject &json);
00023     void foundNeighbours();
00024     void positionIncrement(QVector<unsigned int> &pos, unsigned int value = 1) const;
00025     QVector<QVector<unsigned int> > *getListNeighboursPositionsRecursive
00026     (const QVector<unsigned int> position, unsigned int dimension, QVector<unsigned int> lastAdd) const;
00027     QVector<QVector<unsigned int> > &getListNeighboursPositions(const
00028     QVector<unsigned int> position) const;
00029     QVector<unsigned int> m_dimensions;
00030     QMap<QVector<unsigned int>, Cell* > m_cells;
00031 };
00032 #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

- ~CellHandler
 - CellHandler, [10](#)
- addNeighbour
 - Cell, [6](#)
- Cell, [5](#)
 - addNeighbour, [6](#)
 - Cell, [6](#)
 - getNeighbours, [6](#)
 - getState, [6](#)
 - m_neighbours, [7](#)
 - m_nextState, [8](#)
 - m_state, [8](#)
 - setState, [7](#)
 - validState, [7](#)
- cell.cpp, [15](#)
- cell.h, [15](#), [16](#)
- CellHandler, [8](#)
 - ~CellHandler, [10](#)
 - CellHandler, [9](#)
 - foundNeighbours, [10](#)
 - getCell, [10](#)
 - getListNeighboursPositions, [10](#)
 - getListNeighboursPositionsRecursive, [11](#)
 - load, [12](#)
 - m_cells, [13](#)
 - m_dimensions, [13](#)
 - positionIncrement, [12](#)
- cellhandler.cpp, [16](#)
- cellhandler.h, [19](#)
- foundNeighbours
 - CellHandler, [10](#)
- getCell
 - CellHandler, [10](#)
- getListNeighboursPositions
 - CellHandler, [10](#)
- getListNeighboursPositionsRecursive
 - CellHandler, [11](#)
- getNeighbours
 - Cell, [6](#)
- getState
 - Cell, [6](#)
- load
 - CellHandler, [12](#)
- m_cells
 - CellHandler, [13](#)
- m_dimensions
 - CellHandler, [13](#)
- m_neighbours
 - Cell, [7](#)
- m_nextState
 - Cell, [8](#)
- m_state
 - Cell, [8](#)
- main
 - main.cpp, [20](#)
- main.cpp, [19](#), [20](#)
 - main, [20](#)
- positionIncrement
 - CellHandler, [12](#)
- setState
 - Cell, [7](#)
- validState
 - Cell, [7](#)