Percolate

Generated by Doxygen 1.8.14

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

Class Documentation

2.1 Cell Class Reference

`#include <Cell.hpp>`

Public Types

- enum State { **Full**, **Empty** }

Public Member Functions

- **Cell** (State state=Full, int value=0)
- const State & **getState** () const
- int **getValue** () const
- void **setState** (State state)
- void **setValue** (int value)

2.1.1 Detailed Description

A cell is considered as an element of a grid. Each cell is either full or empty. If it is empty then it has a uniquie number acciocated to it that is updated when we test for percolation.

2.1.2 Member Enumeration Documentation

2.1.2.1 State

`enum Cell::State`

Enumerated data type for whether a cell is full or empty Full=0, and Empty=1.

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

- src/Cell.hpp
- src/Cell.cpp

2.2 Grid2D Class Reference

For the cells in the array.

`#include <Grid2D.hpp>`

Public Member Functions

- Cell & operator() (int i, int j)

  For accessing elements of the grid.
- const Cell & operator() (int i, int j) const

  For accessing elements of a constant grid.
- Grid2D (int rows, int columns, double density, std::default_random_engine &generator)

  Constructor to create a Grid2D instance from dimensions and density.
- Grid2D (const Grid2D &sourceGrid)

  Copy constructor to create Grid2D instance from source Grid2D instance.
- Grid2D (Grid2D &&sourceGrid)

  Move constructor to create a Grid2D instance from a source.
- Grid2D & operator= (const Grid2D &sourceGrid)

  Copy assignment operator for assigning a previously initialised Grid2D instance.
- Grid2D & operator= (Grid2D &&sourceGrid)

  Move assignment operator for assigning previously intialised Grid2D instance.
- ~Grid2D ()

  Destructor for grid must be explicitly implemented since there is dynamic memory assignemnt.
- int getRows () const

  Getter for the number of rows in the grid.
- int getColumns () const

  Getter for the number of columns in the grid.
- void intialise ()

  Intialises the grid ready for the update in the perculation test.
- int update ()

  Updates the m_value in each cell inline with the perculation algorithm.
- bool test ()

  Tests to see whether precolation has occured.
- void printValues ()

  Prints the value at each cell after they have been initialised.

Friends

- std::ostream & operator<< (std::ostream &out, const Grid2D &grid)

  Operator overload for printing the grid to output stream.

2.2.1 Detailed Description

For the cells in the array.

For seeding pseudo random numbers. For generating psuedo random numbers. For bounds checking. For ouput stream operator overloading.

2D grid consisting of filled cells and unfilled cells and the accociated methods to test for perculation through the grid. The grid is indexed like a matric (i,j) is the ith row jth column from the top left. The grid is actually implemented as 1-D dynamically allocated arrary and is surrounded by a ring of filled cells which act as a boundary.

2.2.2 Constructor & Destructor Documentation

2.2.2.1 Grid2D() [1/3]

```
Grid2D::Grid2D (
            int rows,
            int columns,
            double density,
            std::default_random_engine & generator )
```

Constructor to create a Grid2D instance from dimensions and density.

The m_cellArray is allocated dynamically since the dimensions and density will not be known at compile time. The array could also be very large, so it should be allocated from the heap anyway.

Parameters

rows	Integer representing the number of rows in the array, this should not include the buffer rows since this is all implemented internally.
columns	Integer representing the number of columns in the array, this should not include the buffer columns since this is all implemented internally.
density	Floating point value between 0 and 1 representing the density of filled squares within the grid.

To include the halo the number of rows and columns need to be increased by 2 so that there is an extra row on the top and bottom and extra column on either side.

2.2.2.2 Grid2D() [2/3]

```
Grid2D::Grid2D (
            const Grid2D & sourceGrid )
```

Copy constructor to create Grid2D instance from source Grid2D instance.

Deep copying is neccasary since we have dynamic memory allocation int the constructor. Otherwise deferencing of dangling pointers may occur when we release memory at the end of a scope.

Parameters

sourceGrid	constant Grid2D reference instance to be copied from.

2.2.2.3 Grid2D() [3/3]

```
Grid2D::Grid2D (
            Grid2D && sourceGrid )
```

Move constructor to create a Grid2D instance from a source.

This is more performant when we are copying from R-Value references and will be called automatically in those cases.

Parameters

sourceGrid	R-Value reference from which the new Grid2D object will take ownership of the member variables memory.

Ownership of member variables in sourceGrid is transfered to the new grid.

Ownership of the member variables is then removed from the sourceGrid. Technically since m_columns and m_↩ cellArray are just Integer variables that are non-dynamic they do not need to have their ownership by the sourceGrid removed, it is only done here for completeness.

2.2.3 Member Function Documentation

2.2.3.1 getColumns()

```
int Grid2D::getColumns ( ) const
```

Getter for the number of columns in the grid.

Returns

Integer value representing the number of columns in the grid.

2.2.3.2 getRows()

```
int Grid2D::getRows ( ) const
```

Getter for the number of rows in the grid.

Returns

Integer value representing the number of rows in the grid.

2.2.3.3 intialise()

```
void Grid2D::intialise ( )
```

Intialises the grid ready for the update in the perculation test.

Sets the m_value of each cell to a unique integer.

2.2.3.4 operator()() [1/2]

```
Cell & Grid2D::operator() (
            int i,
            int j )
```

For accessing elements of the grid.

By overloading the () operator we are able to access elements of the m_cellArray as if they were members of a 2D array rather than a 1-D one. This conceptually simplifies the code were specific cells are accessed. the indexing system (i,j) follows that of a matrix starting from 0.

Parameters

i	Integer value representing the the row to be accessed.
j	Integer value representing the column to be accessed.

Returns

Cell refernce to the cell at (i,j) in the matrix.

2.2.3.5 operator()() [2/2]

```
const Cell & Grid2D::operator() (
            int i,
            int j ) const
```

For accessing elements of a constant grid.

This works in exactly the same way as the opertor() for non-constant grids. It has to be overloaded twice however so that it can deal with constant grids as well.

Parameters

i	Integer value representing the the row to be accessed.
j	Integer value representing the column to be accessed.

Returns

Constant Cell refernce to the cell at (i,j) in the matrix.

2.2.3.6 operator=() [1/2]

```
Grid2D & Grid2D::operator= (
            const Grid2D & sourceGrid )
```

Copy assignment operator for assigning a previously initialised Grid2D instance.

Deep copying is neccasary since we have dynamic memory allocation int the constructor. Otherwise deferencing of dangling pointers may occur when we release memory at the end of a scope.

Parameters

srouceGrid	constant Grid 2D reference instance from which the current Grid2D instance should be intialized.

Returns

Grid2D reference corresponding to ∗this so assignment operations can be chained together.

The cells are copied and placed in the newely allocated array, exluding the halo which has already been allocated.

2.2.3.7 operator=() [2/2]

```
Grid2D & Grid2D::operator= (
            Grid2D && sourceGrid )
```

Move assignment operator for assigning previously intialised Grid2D instance.

This is more performant when we are assigning from R-Value references and will be called automatically in those cases.

Parameters

sourceGrid	R-Value reference from which the (∗this) Grid2D object will take ownership of the member variables memory.

Returns

Grid2D reference corresponding to ∗this so assignment operations can be chained together.

2.2.3.8 test()

```
bool Grid2D::test ( )
```

Tests to see whether precolation has occured.

If perculation has occured there will be a cluster with that reaches from the top to the bottom of the Grid. Since we know that each cluster has the same number accociated to each of its cells after the updates have stopped we can just check whether any of the cells in the last row have the same accociated number as any in the first row.

Returns

Boolean representing whether preculation has occured. True if percolated, false otherwise.

2.2.3.9 update()

```
int Grid2D::update ( )
```

Updates the m_value in each cell inline with the perculation algorithm.

Looks for clusters (empty cells that are connected by either adjacent cells or cells that are ontop of one another) until they are all found, by looking at each empty cell, replacing its m_value with the maximum of its four nearest neighbours.

Returns

Integer value representing the number of cells that have changed.

2.2.4 Friends And Related Function Documentation

2.2.4.1 operator<<

```
std::ostream& operator<< (
            std::ostream & out,
            const Grid2D & grid )  [friend]
```

Operator overload for printing the grid to output stream.

Needs to be overloaded as a friend function since its left operand is not (∗this)

Parameters

out	std::ostream reference which the grid will be streamed into.
grid	const Grid2D reference that is the grid being outputed.

Returns

std:ostream reference to the input ostream so that output can be chained.

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

- src/Grid2D.hpp
- src/Grid2D.cpp

Index