

DEMONGEON CRAWL

Generated by Doxygen 1.8.13

Contents

1	Namespace Index	1
1.1	Namespace List	1
2	Class Index	3
2.1	Class List	3
3	Namespace Documentation	5
3.1	maze Namespace Reference	5
3.1.1	Detailed Description	5
4	Class Documentation	7
4.1	maze::Cell< T > Class Template Reference	7
4.1.1	Detailed Description	8
4.1.2	Constructor & Destructor Documentation	9
4.1.2.1	Cell()	9
4.1.3	Member Function Documentation	9
4.1.3.1	at_east()	9
4.1.3.2	at_north()	9
4.1.3.3	at_south()	9
4.1.3.4	at_west()	10
4.1.3.5	get_east()	10
4.1.3.6	get_north()	10
4.1.3.7	get_south()	10
4.1.3.8	get_west()	10
4.1.3.9	set_east()	11

4.1.3.10	set_north()	11
4.1.3.11	set_south()	11
4.1.3.12	set_west()	11
4.1.3.13	to_east()	11
4.1.3.14	to_north()	12
4.1.3.15	to_south()	12
4.1.3.16	to_west()	12
4.2	maze::Maze< T > Class Template Reference	12
4.2.1	Detailed Description	13
4.2.2	Constructor & Destructor Documentation	13
4.2.2.1	Maze() [1/3]	13
4.2.2.2	Maze() [2/3]	14
4.2.2.3	Maze() [3/3]	14
4.2.3	Member Function Documentation	14
4.2.3.1	cell()	14
4.2.4	Member Data Documentation	14
4.2.4.1	cells	15
4.3	maze::RawCell< T > Class Template Reference	15
4.3.1	Detailed Description	15
Index		17

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

maze	Provides functionality required to model a maze	5
----------------------	---	-------------------

Chapter 2

Class Index

2.1 Class List

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

maze::Cell< T >	A safe wrapper around a pointer to a RawCell	7
maze::Maze< T >	A maze	12
maze::RawCell< T >	A single cell in a maze	15

Chapter 3

Namespace Documentation

3.1 maze Namespace Reference

Provides functionality required to model a maze.

Classes

- class [Cell](#)
A safe wrapper around a pointer to a [RawCell](#).
- class [Maze](#)
A maze.
- class [RawCell](#)
A single cell in a maze.

Typedefs

- typedef std::size_t [index](#)
Type alias used for indexes inside the maze.

3.1.1 Detailed Description

Provides functionality required to model a maze.

A maze is a grid of cells, each marks accessibility between them. It can be seen as a potentially disconnected bidirected 2d lattice graph.

Chapter 4

Class Documentation

4.1 maze::Cell< T > Class Template Reference

A safe wrapper around a pointer to a [RawCell](#).

```
#include <maze.hpp>
```

Public Types

- typedef T [type](#)
A type alias for the stored type.

Public Member Functions

- [Cell](#) ([RawCell](#)< [type](#) > *[cell](#), [index](#) [pos_x](#), [index](#) [pos_y](#), [index](#) [bound_x](#), [index](#) [bound_y](#))
Constructs a new wrapper around a cell, from a [RawCell](#), its position, and its bounds.
- [type](#) * [data](#) ()
Returns a pointer to the data stored in the cell.
- bool [is_null](#) ()
Checks if the target [Cell](#) is a null pointer.
- [index](#) [get_pos_x](#) ()
Getter method for [pos_x](#).
- [index](#) [get_pos_y](#) ()
Getter method for [pos_y](#).
- [index](#) [get_bound_x](#) ()
Getter method for [bound_x](#).
- [index](#) [get_bound_y](#) ()
Getter method for [bound_y](#).
- bool [get_north](#) ()
Getter method for north.
- bool [get_east](#) ()
Getter method for east.
- bool [get_south](#) ()
Getter method for south.

- `bool get_west ()`
Getter method for west.
- `bool set_north (bool)`
Setter method for north.
- `bool set_east (bool)`
Setter method for east.
- `bool set_south (bool)`
Setter method for south.
- `bool set_west (bool)`
Setter method for west.
- `Cell at_north ()`
Returns the north adjacent [Cell](#).
- `Cell at_east ()`
Returns the east adjacent [Cell](#).
- `Cell at_south ()`
Returns the south adjacent [Cell](#).
- `Cell at_west ()`
Returns the west adjacent [Cell](#).
- `bool to_north ()`
Mutates cell in place to be its north neighbour.
- `bool to_east ()`
Mutates cell in place to be its east neighbour.
- `bool to_south ()`
Mutates cell in place to be its south neighbour.
- `bool to_west ()`
Mutates cell in place to be its west neighbour.

Protected Attributes

- `RawCell< type > * cell`
The underlying [RawCell](#).
- `index pos_x`
The x position in the maze.
- `index pos_y`
The y position in the maze.
- `index bound_x`
The x boundry of the maze.
- `index bound_y`
The y boundry of the maze.

4.1.1 Detailed Description

```
template<typename T>
class maze::Cell< T >
```

A safe wrapper around a pointer to a [RawCell](#).

Provides getter and setter methods. [Cell](#) is invalidated if [Maze](#) goes out of scope.

Definition at line 77 of file [maze.hpp](#).

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Cell()

```
template<typename T>
maze::Cell< T >::Cell (
    RawCell< type > * cell,
    index pos_x,
    index pos_y,
    index bound_x,
    index bound_y ) [inline]
```

Constructs a new wrapper around a cell, from a [RawCell](#), its position, and its bounds.

The [RawCell](#) should be the [RawCell](#) being Wrapped, not the [RawCell](#) at the start of the [Maze](#)'s memory.

Definition at line 89 of file [maze.hpp](#).

4.1.3 Member Function Documentation

4.1.3.1 at_east()

```
template<typename T>
Cell maze::Cell< T >::at_east ( )
```

Returns the east adjacent [Cell](#).

Target [RawCell](#) will be null if done at a boundary.

4.1.3.2 at_north()

```
template<typename T>
Cell maze::Cell< T >::at_north ( )
```

Returns the north adjacent [Cell](#).

Target [RawCell](#) will be null if done at a boundary.

4.1.3.3 at_south()

```
template<typename T>
Cell maze::Cell< T >::at_south ( )
```

Returns the south adjacent [Cell](#).

Target [RawCell](#) will be null if done at a boundary.

4.1.3.4 at_west()

```
template<typename T>
Cell maze::Cell< T >::at_west ( )
```

Returns the west adjacent [Cell](#).

Target [RawCell](#) will be null if done at a boundry.

4.1.3.5 get_east()

```
template<typename T>
bool maze::Cell< T >::get_east ( )
```

Getter method for east.

Returns false if out of bounds, regardless of underlying value.

4.1.3.6 get_north()

```
template<typename T>
bool maze::Cell< T >::get_north ( )
```

Getter method for north.

Returns false if out of bounds, regardless of underlying value.

4.1.3.7 get_south()

```
template<typename T>
bool maze::Cell< T >::get_south ( )
```

Getter method for south.

Returns false if out of bounds, regardless of underlying value.

4.1.3.8 get_west()

```
template<typename T>
bool maze::Cell< T >::get_west ( )
```

Getter method for west.

Returns false if out of bounds, regardless of underlying value.

4.1.3.9 set_east()

```
template<typename T>
bool maze::Cell< T >::set_east (
    bool )
```

Setter method for east.

Returns false if out of bounds.

4.1.3.10 set_north()

```
template<typename T>
bool maze::Cell< T >::set_north (
    bool )
```

Setter method for north.

Returns false if out of bounds.

4.1.3.11 set_south()

```
template<typename T>
bool maze::Cell< T >::set_south (
    bool )
```

Setter method for south.

Returns false if out of bounds.

4.1.3.12 set_west()

```
template<typename T>
bool maze::Cell< T >::set_west (
    bool )
```

Setter method for west.

Returns false if out of bounds.

4.1.3.13 to_east()

```
template<typename T>
bool maze::Cell< T >::to_east ( )
```

Mutates cell in place to be its east neighbour.

Returns false and does not change if at a boundry.

4.1.3.14 to_north()

```
template<typename T>
bool maze::Cell< T >::to_north ( )
```

Mutates cell in place to be its north neighbour.

Returns false and does not change if at a boundry.

4.1.3.15 to_south()

```
template<typename T>
bool maze::Cell< T >::to_south ( )
```

Mutates cell in place to be its south neighbour.

Returns false and does not change if at a boundry.

4.1.3.16 to_west()

```
template<typename T>
bool maze::Cell< T >::to_west ( )
```

Mutates cell in place to be its west neighbour.

Returns false and does not change if at a boundry.

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

- maze.hpp

4.2 maze::Maze< T > Class Template Reference

A maze.

```
#include <maze.hpp>
```

Public Types

- typedef T [type](#)
A type alias for the stored type.

Public Member Functions

- [Maze](#) ([index size_x](#), [index size_y](#))
Creates a new [Maze](#) from size.
- [Maze](#) ([index size_x](#), [index size_y](#), [RawCell](#)< [type](#) > init)
Creates a new [Maze](#) from size and a template.
- [Maze](#) ([RawCell](#)< [type](#) > *cell, [index size_x](#), [index size_y](#))
Creates a [Maze](#) from its raw parts.
- [~Maze](#) ()
Deallocates the memory reserved for the cells of the maze.
- [index get_size_x](#) ()
Getter for size_x.
- [index get_size_y](#) ()
Getter for size_y.
- [Cell](#)< [type](#) > [cell](#) ([index](#) pos_x, [index](#) pos_y)
Creates and returns a [Cell](#) given the indexes of that cell.

Protected Attributes

- [RawCell](#)< [type](#) > * [cells](#)
The raw cells that make up the maze.
- [index size_x](#)
The size of the mazes x dimension.
- [index size_y](#)
The size of the mazes y dimension.

4.2.1 Detailed Description

```
template<typename T>
class maze::Maze< T >
```

A maze.

Stores size, and controls memory of elements. A maze can be seen as a potentially disconnected bidirected 2d lattice graph.

Definition at line 259 of file [maze.hpp](#).

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Maze() [1/3]

```
template<typename T >
maze::Maze< T >::Maze (
    index size_x,
    index size_y )
```

Creates a new [Maze](#) from size.

Allocates memory and initializes fully blocked cells.

4.2.2.2 Maze() [2/3]

```
template<typename T >
maze::Maze< T >::Maze (
    index size_x,
    index size_y,
    RawCell< type > init )
```

Creates a new [Maze](#) from size and a template.

Allocates memory and initializes fully with copies of a supplied cell.

4.2.2.3 Maze() [3/3]

```
template<typename T >
maze::Maze< T >::Maze (
    RawCell< type > * cell,
    index size_x,
    index size_y )
```

Creates a [Maze](#) from its raw parts.

Takes "ownership" of `cells` and will deallocate it when its destructor is called.

4.2.3 Member Function Documentation

4.2.3.1 cell()

```
template<typename T >
Cell<type> maze::Maze< T >::cell (
    index pos_x,
    index pos_y )
```

Creates and returns a [Cell](#) given the indexes of that cell.

[Cell](#) is invalidated when maze is destructured.

4.2.4 Member Data Documentation

4.2.4.1 cells

```
template<typename T >
RawCell<type>* maze::Maze< T >::cells [protected]
```

The raw cells that make up the maze.

Stored continuously in memory. This is deallocated at the point of maze destruction.

Definition at line 310 of file [maze.hpp](#).

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

- [maze.hpp](#)

4.3 maze::RawCell< T > Class Template Reference

A single cell in a maze.

```
#include <maze.hpp>
```

Public Types

- typedef T [type](#)
A type alias for the stored type.

Public Member Functions

- [RawCell](#) ([type data](#))
Creates a new [RawCell<type>](#) with no exits.
- [RawCell](#) ([type data](#), bool [north](#), bool [east](#), bool [south](#), bool [west](#))
Creates a new [RawCell<type>](#) with specified exits.

Public Attributes

- [type data](#)
The data stored inside the cell.
- bool [north](#)
Legal to exit to the north.
- bool [east](#)
Legal to exit to the east.
- bool [south](#)
Legal to exit to the south.
- bool [west](#)
Legal to exit to the west.

4.3.1 Detailed Description

```
template<typename T>
class maze::RawCell< T >
```

A single cell in a maze.

It models both the data stored in the cell and the legal exits of the cell.

Definition at line 21 of file [maze.hpp](#).

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

- [maze.hpp](#)

Index

- at_east
 - maze::Cell, 9
- at_north
 - maze::Cell, 9
- at_south
 - maze::Cell, 9
- at_west
 - maze::Cell, 9
- Cell
 - maze::Cell, 9
- cell
 - maze::Maze, 14
- cells
 - maze::Maze, 14
- get_east
 - maze::Cell, 10
- get_north
 - maze::Cell, 10
- get_south
 - maze::Cell, 10
- get_west
 - maze::Cell, 10
- Maze
 - maze::Maze, 13, 14
- maze, 5
- maze::Cell
 - at_east, 9
 - at_north, 9
 - at_south, 9
 - at_west, 9
 - Cell, 9
 - get_east, 10
 - get_north, 10
 - get_south, 10
 - get_west, 10
 - set_east, 10
 - set_north, 11
 - set_south, 11
 - set_west, 11
 - to_east, 11
 - to_north, 11
 - to_south, 12
 - to_west, 12
- maze::Cell< T >, 7
- maze::Maze
 - cell, 14
 - cells, 14
 - Maze, 13, 14
 - maze::Maze< T >, 12
 - maze::RawCell< T >, 15
 - set_east
 - maze::Cell, 10
 - set_north
 - maze::Cell, 11
 - set_south
 - maze::Cell, 11
 - set_west
 - maze::Cell, 11
 - to_east
 - maze::Cell, 11
 - to_north
 - maze::Cell, 11
 - to_south
 - maze::Cell, 12
 - to_west
 - maze::Cell, 12