

2D Predator-Prey Model

Generated by Doxygen 1.8.14

Contents

1	Hierarchical Index	1
1.1	Class Hierarchy	1
2	Class Index	3
2.1	Class List	3
3	File Index	5
3.1	File List	5
4	Class Documentation	7
4.1	Cell Class Reference	7
4.1.1	Detailed Description	8
4.1.2	Constructor & Destructor Documentation	8
4.1.2.1	Cell()	8
4.1.3	Member Function Documentation	8
4.1.3.1	getPredDensity()	8
4.1.3.2	getPreyDensity()	9
4.1.3.3	getState()	9
4.1.3.4	setPredDensity()	9
4.1.3.5	setPreyDensity()	9
4.1.3.6	setState()	10
4.1.4	Friends And Related Function Documentation	10
4.1.4.1	TestCell	10
4.2	Grid Class Reference	10
4.2.1	Detailed Description	12

4.2.2	Constructor & Destructor Documentation	12
4.2.2.1	Grid() [1/4]	12
4.2.2.2	Grid() [2/4]	12
4.2.2.3	Grid() [3/4]	12
4.2.2.4	Grid() [4/4]	14
4.2.2.5	~Grid()	14
4.2.3	Member Function Documentation	14
4.2.3.1	dryNeighbours()	14
4.2.3.2	getCellState()	15
4.2.3.3	getColumns()	15
4.2.3.4	getRows()	15
4.2.3.5	operator>() [1/2]	16
4.2.3.6	operator>() [2/2]	16
4.2.3.7	operator=() [1/2]	17
4.2.3.8	operator=() [2/2]	17
4.2.3.9	predDensity()	17
4.2.3.10	preyDensity()	19
4.2.3.11	printDensities()	19
4.2.3.12	printPPM()	20
4.2.3.13	setUniformDistribution()	20
4.2.3.14	setUniformPredDistribution()	21
4.2.3.15	setUniformPreyDistribution()	21
4.2.4	Friends And Related Function Documentation	22
4.2.4.1	operator<<	22
4.2.4.2	TestGrid	22
4.3	TestCell Class Reference	22
4.3.1	Detailed Description	23
4.3.2	Member Function Documentation	23
4.3.2.1	testConstructorDefault()	24
4.3.2.2	testConstructorDry()	24

4.3.2.3	testConstructorDryNegPredDensity()	24
4.3.2.4	testConstructorDryNegPreyDensity()	24
4.3.2.5	testConstructorWet()	24
4.3.2.6	testGetPredDensity()	24
4.3.2.7	testGetPreyDensity()	25
4.3.2.8	testGetState()	25
4.3.2.9	testSetNegPredDensity()	25
4.3.2.10	testSetNegPreyDensity()	25
4.3.2.11	testSetPosPredDensity()	25
4.3.2.12	testSetPosPreyDensity()	25
4.3.2.13	testSetStateDry()	26
4.3.2.14	testSetStateWet()	26
4.4	TestGrid Class Reference	26
4.4.1	Detailed Description	26
4.5	TestUpdateGrid Class Reference	27
4.5.1	Detailed Description	27
4.5.2	Member Function Documentation	27
4.5.2.1	testRealisticGridUpdated()	27
4.5.2.2	testWetGridUpdated()	28
4.5.2.3	testZeroDensityGridUpdated()	28
5	File Documentation	29
5.1	src/Cell.hpp File Reference	29
5.2	src/Grid.hpp File Reference	29
5.3	src/updateGrid.hpp File Reference	29
5.3.1	Detailed Description	29
5.4	test/TestCell.hpp File Reference	30
5.5	test/TestGrid.hpp File Reference	30
5.6	test/TestUpdateGrid.hpp File Reference	30
Index		31

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Cell	7
Grid	10
TestFixture	
TestCell	22
TestGrid	26
TestUpdateGrid	27

Chapter 2

Class Index

2.1 Class List

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

Cell	Models a single cell	7
Grid	Models a 2D landscape of cells	10
TestCell	Tests the Cell class	22
TestGrid	Tests the Grid class	26
TestUpdateGrid	Tests the updateGrid function	27

Chapter 3

File Index

3.1 File List

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

src/ Cell.hpp	29
src/ Grid.hpp	29
src/ updateGrid.hpp	
Updates the grid by one time step	29
test/ TestCell.hpp	30
test/ TestGrid.hpp	30
test/ TestUpdateGrid.hpp	30

Chapter 4

Class Documentation

4.1 Cell Class Reference

Models a single cell.

```
#include <Cell.hpp>
```

Public Types

- enum `State` { `Wet`, `Dry` }
Enum that represents a value that can be either wet or dry.

Public Member Functions

- `Cell (State state=Wet, double predDensity=0.0, double preyDensity=0.0)`
Creates a `Cell`.
- `~Cell ()`
Default destructor; no dynamic memory allocation is required for this class.
- `double getPredDensity () const`
Getter for the predator density in the cell.
- `double getPreyDensity () const`
Getter for the prey density in the cell.
- `State getState () const`
Getter for the state of the cell.
- `void setPredDensity (double predDensity=0.0)`
Setter for the predator density in the cell.
- `void setPreyDensity (double preyDensity=0.0)`
Setter for the prey density in the cell.
- `void setState (State state=Wet)`
Setter for the state of the cell.

Friends

- class `TestCell`
Test class for the `Cell`.

4.1.1 Detailed Description

Models a single cell.

A cell forms a single "square" in the landscape and is considered to have three properties a predator density, a prey density and whether or not the cell is land (so dry) or water (so wet)

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Cell()

```
Cell::Cell (
    Cell::State state = Wet,
    double predDensity = 0.0,
    double preyDensity = 0.0 )
```

Creates a [Cell](#).

Standard constructor to initialize the member variables of the [Cell](#) object. If no arguments are provided the [Cell](#) defaults to be Wet with predator and prey densities of zero.

Parameters

<i>state</i>	a Cell::State value that is either Wet or Dry setting the state of the cell. Defaults to Wet.
<i>predDensity</i>	a floating point value setting the density of the predators in the cell. Defaults to 0.
<i>preyDensity</i>	a floating point value setting the density of the prey in the cell. Defaults to 0.

4.1.3 Member Function Documentation

4.1.3.1 getPredDensity()

```
double Cell::getPredDensity ( ) const
```

Getter for the predator density in the cell.

Returns

The value of m_predDensity.

4.1.3.2 getPreyDensity()

```
double Cell::getPreyDensity ( ) const
```

Getter for the prey density in the cell.

Returns

The value of `m_preyDensity`.

4.1.3.3 getState()

```
Cell::State Cell::getState ( ) const
```

Getter for the state of the cell.

Returns

The value of `m_state`.

4.1.3.4 setPredDensity()

```
void Cell::setPredDensity (
    double predDensity = 0.0 )
```

Setter for the predator density in the cell.

Parameters

<i>predDensity</i>	a floating point value setting the density of the predators in the cell. Defaults to 0.
--------------------	---

4.1.3.5 setPreyDensity()

```
void Cell::setPreyDensity (
    double preyDensity = 0.0 )
```

Setter for the prey density in the cell.

Parameters

<i>predDensity</i>	a floating point value setting the density of the prey in the cell. Defaults to 0.
--------------------	--

4.1.3.6 setState()

```
void Cell::setState (
    Cell::State state = Wet )
```

Setter for the state of the cell.

Parameters

<i>state</i>	a Cell::State enum value setting the state of the cell to be either Wet or Dry. Defaults to Wet.
--------------	--

4.1.4 Friends And Related Function Documentation

4.1.4.1 TestCell

```
friend class TestCell [friend]
```

Test class for the [Cell](#).

Declare a [TestCell](#) friend member variable to allow access to private member variables in test class.

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

- [src/Cell.hpp](#)
- [src/Cell.cpp](#)

4.2 Grid Class Reference

Models a 2D landscape of cells.

```
#include <Grid.hpp>
```


Public Member Functions

- [Grid](#) ()
Default constructor.
- [Grid](#) (int columns, int rows, int **const data)
Creates [Grid](#) objects from some input array representing the [Grid](#).
- [Grid](#) (const [Grid](#) &sourceGrid)
Creates [Grid](#) objects from a source [Grid](#) with deep copying.
- [Grid](#) ([Grid](#) &&sourceGrid)
Creates [Grid](#) objects from a source [Grid](#) with move semantics.
- [~Grid](#) ()
Releases memory held by m_cellArray.
- [Grid](#) & [operator=](#) (const [Grid](#) &sourceGrid)
Assigns [Grid](#) objects from a source with deep copying.
- [Grid](#) & [operator=](#) ([Grid](#) &&sourceGrid)
Assigns [Grid](#) objects from a source [Grid](#) with move semantics.
- int [getColumns](#) () const
Getter for the number of actual columns in the grid.
- int [getRows](#) () const
Getter for the number of actual rows in the grid.
- void [setUniformPredDistribution](#) (double upperBound, std::default_random_engine &generator)
Sets uniform random predator distribution in each grid cell.
- void [setUniformPreyDistribution](#) (double upperBound, std::default_random_engine &generator)
- void [setUniformDistribution](#) (double predUpperBound, double preyUpperbound, std::default_random_engine &generator)
- double [predDensity](#) (bool includeWetCells=true) const
Calculates average predator density across the grid.
- double [preyDensity](#) (bool includeWetCells=true) const
Calculates average prey density across the grid.
- [Cell](#) & [operator\(\)](#) (int i, int j)
Operator overload to access the [Cell](#) stored at the (i,j)th coordinates of the grid. Since the 2-D landscape is implemented as a 1-D array with the same number of elements for memory reasons, this operator provides a way of accessing the elements of the 1-D array with two indices (i,j) as if it were a 2D-array. This operator should be used whenever direct access to the grid cells is needed and extensive use of it is made in the constructors.
- const [Cell](#) & [operator\(\)](#) (int i, int j) const
Second operator overload to access the [Cell](#) stored at the (i,j)th coordinate if the grid is a constant variable.
- int [dryNeighbours](#) (int i, int j) const
Calculates number of Dry neighbours of a given cell.
- void [printDensities](#) (std::ostream &out) const
Outputs predator and prey densities to output stream.
- int [printPPM](#) (std::ofstream &file, int maxNumberPPM) const
Prints the densities of the predators and prey to a ppm file provided.
- [Cell::State](#) [getCellState](#) (int columnIndex, int rowIndex) const
Retrieves state of cell at a given coordinate.

Friends

- class [TestGrid](#)
Test class for the grid.
- std::ostream & [operator<<](#) (std::ostream &out, const [Grid](#) &grid)
Operator overload for outputting the grid.

4.2.1 Detailed Description

Models a 2D landscape of cells.

A [Grid](#) consists of a (2D) array of cells that can either be land or water and each have a predator and prey density. The actual array is implemented with a "halo" of Wet cells, this means that the densities of predators and prey in these cells is zero. This is very useful for implementing any differential equations where we may inadvertently check the `#columns + 1` or `#rows + 1`, so rather than introducing bounds checking, we can introduce this boundary.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 [Grid\(\)](#) [1/4]

```
Grid::Grid ( )
```

Default constructor.

Creates a grid of size 0 with a nullptr as its grid

4.2.2.2 [Grid\(\)](#) [2/4]

```
Grid::Grid (
    int columns,
    int rows,
    int **const data )
```

Creates [Grid](#) objects from some input array representing the [Grid](#).

This constructor will dynamically allocate a 1-D array of Cells with a halo of water cells, which represents the 2-D landscape.

Parameters

<i>columns</i>	Integer value representing the number of columns in the Grid object, not including the halo.
<i>rows</i>	Integer value representing the number of rows in the Grid object, not including the halo.
<i>data</i>	Integer array of values which represent the Wet and Dry cells in the grid, must be 1 and 0 only and must have the same number of rows and columns as the explicit values provided

4.2.2.3 [Grid\(\)](#) [3/4]

```
Grid::Grid (
    const Grid & sourceGrid )
```

Creates [Grid](#) objects from a source [Grid](#) with deep copying.

Due to the dynamic memory allocation that takes place in the constructor, the copy constructor is overloaded to insure that deep copying takes place and that there are no dangling pointers when grid objects go out of scope.

Parameters

<i>sourceGrid</i>	constant Grid reference from which the deep copying will be done.
-------------------	---

4.2.2.4 Grid() [4/4]

```
Grid::Grid (
    Grid && sourceGrid )
```

Creates [Grid](#) objects from a source [Grid](#) with move semantics.

This is implemented for performance reasons. If a grid is ever returned from a function, for example, a function that might update the grid, it is quicker to do this via move semantics rather than copy construction.

Parameters

<i>sourceGrid</i>	constant R-value reference from which the ownership of the member variables will be transferred.
-------------------	--

4.2.2.5 ~Grid()

```
Grid::~~Grid ( )
```

Releases memory held by `m_cellArray`.

Destructor is explicitly implemented since the [Grid](#) class has dynamic memory allocation in its constructors of the `m_cellArray` member variable.

4.2.3 Member Function Documentation**4.2.3.1 dryNeighbours()**

```
int Grid::dryNeighbours (
    int i,
    int j ) const
```

Calculates number of Dry neighbours of a given cell.

Neighbours are only considered to be non-diagonal, so this function counts the number of Dry cells directly above/below and to the left/right of the specified cell.

Parameters

<i>i</i>	column number/x-coordinate of cell.
<i>j</i>	row number/y-coordinate of cell.

Returns

Integer value representing number of Dry neighbours of the cell.

4.2.3.2 getCellState()

```
Cell::State Grid::getCellState (
    int columnIndex,
    int rowIndex ) const
```

Retrieves state of cell at a given coordinate.

Parameters

<i>columnIndex</i>	integer value representing the column in which the state of the cell should be retrieved.
<i>rowIndex</i>	integer value representing the row in which the state of the cell should be retrieved.

Returns

[Cell::State](#) value (either wet or dry) representing the state of the specified cell.

4.2.3.3 getColumns()

```
int Grid::getColumns ( ) const
```

Getter for the number of actual columns in the grid.

Returns

The value of `m_columns`, the actual number of columns in the grid without a halo of water.

4.2.3.4 getRows()

```
int Grid::getRows ( ) const
```

Getter for the number of actual rows in the grid.

Returns

The value of `m_rows`, the actual number of rows in the grid without a halo of water.

4.2.3.5 operator() [1/2]

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

Operator overload to access the [Cell](#) stored at the (i,j)th coordinates of the grid. Since the 2-D landscape is implemented as a 1-D array with the same number of elements for memory reasons, this operator provides a way of accessing the elements of the 1-D array with two indices (i,j) as if it were a 2D-array. This operator should be used whenever direct access to the grid cells is needed and extensive use of it is made in the constructors.

The indexing system for this operator and therefore the [Grid](#) itself follows that given in the specification document for this project in the figure in section 2.1. i.e. (i,j) corresponds to x (column number) and y coordinates (row number) where the origin is taken to be in the bottom left corner. This is not the same as matrix notation. For example (3,4) would return the cell in the 3rd column of the 4th row.

Parameters

<i>i</i>	column number/x-coordinate of cell.
<i>j</i>	row/y-coordinate of cell.

Returns

[Cell](#) reference to the cell at the (i,j) coordinate.

4.2.3.6 operator() [2/2]

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

Second operator overload to access the [Cell](#) stored at the (i,j)th coordinate if the grid is a constant variable.

This function behaves in exactly the same way as its non-constant counterpart. However the non-constant operator() overload will not work with any constant grid objects, since it returns a reference, it would be able to edit contents of the constant Cells `m_cellArray` member. For an explanation of the indexing system please see the non-constant overload.

Parameters

<i>i</i>	column number/x-coordinate of cell.
<i>j</i>	row/y-coordinate of cell.

Returns

Constant [Cell](#) reference to the cell at the (i,j) coordinate, so that the main method cannot edit the contents of a constant [Grid](#).

4.2.3.7 operator=() [1/2]

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

Assigns [Grid](#) objects from a source with deep copying.

Due to the dynamic memory allocation that takes place in the constructor, the copy assignment operator is overloaded to insure that deep copying takes place in any assignment and that there are no dangling pointers when grid objects go out of scope. It also checks for self assignment which could otherwise lead to memory problems. This pairs with the copy constructor that also does deep copying.

Parameters

<i>sourceGrid</i>	constant Grid reference from which the deep copying will be done in the assignment of the grid member variables.
-------------------	--

Returns

[Grid](#) reference *this, so that the newly assigned operator may be chained into other assignment operations.

4.2.3.8 operator=() [2/2]

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

Assigns [Grid](#) objects from a source [Grid](#) with move semantics.

This is implemented for performance reasons to go with the move constructor. If an R-value [Grid](#) is assigned to a [Grid](#) variable, this operator will be used instead of the normal copy assignment and will be more performant.

Parameters

<i>sourceGrid</i>	constant R-value Grid reference from which the ownership of the member variables will be transferred.
-------------------	---

Returns

[Grid](#) reference *this, so that the newly assigned operator may be chained into other assignment operations.

4.2.3.9 predDensity()

```
double Grid::predDensity (
    bool includeWetCells = true ) const
```

Calculates average predator density across the grid.

Calculates the average value of the predator density across the entire grid, not included the halo of Wet cells. However, the average can be taken over just the Dry cells in the grid, or over the Dry and Wet cells in the grid if the corresponding argument is provided.

Parameters

<i>includeWetCells</i>	bool value that represents whether the user wants to include the Wet cells in the grid as the well as the Dry cells, as the total number of cells to take the average over. This value defaults to true i.e. the default behaviour is to average over all cells in the grid including the Wet ones where predator/prey densities will be zero.
------------------------	--

Returns

Floating point value representing the average predator density across the grid either including or not including the Wet cells, depending on how the function was called.

4.2.3.10 preyDensity()

```
double Grid::preyDensity (
    bool includeWetCells = true ) const
```

Calculates average prey density across the grid.

Calculates the average value of the prey density across the entire grid, not included the halo of Wet cells. However, the average can be taken over just the Dry cells in the grid, or over the Dry and Wet cells in the grid if the corresponding argument is provided.

Parameters

<i>includeWetCells</i>	bool value that represents whether the user wants to include the Wet cells in the grid as the well as the Dry cells, as the total number of cells to take the average over. This value defaults to true i.e. the default behaviour is to average over all cells in the grid including the Wet ones where predator/prey densities will be zero.
------------------------	--

Returns

Floating point value representing the average prey density across the grid either including or not including the Wet cells, depending on how the function was called.

Total up the number of Wet cells as well.

4.2.3.11 printDensities()

```
void Grid::printDensities (
    std::ostream & out ) const
```

Outputs predator and prey densities to output stream.

Outputs densities of predator and prey for each cell in the format: i j predator density prey density and repeats this for all cells in the grid.

Parameters

<i>out</i>	std::ostream reference which is the output stream. This will also work for fstream since it inherits from ostream.
------------	--

4.2.3.12 printPPM()

```
int Grid::printPPM (
    std::ofstream & file,
    int maxNumberPPM ) const
```

Prints the densities of the predators and prey to a ppm file provided.

This function will print the to the file in the format:

P3 (required for a PPM and means this is a RGB colour image in ASCII) m_columns m_rows (width and height of image) MAX (maximum value for each colour) 0 predDensity preyDensity (at coordinate (m_rows, 1)) 0 predDensity preyDensity (at coordinate (m_rows, 2)) 0 predDensity preyDensity (at coordinate (m_rows, 3)) 0 predDensity preyDensity (at coordinate (1, m_columns))

To overcome the fact that each line can contain at most 70 characters each coordinate pixel is printed to a new line, this works fine formatting wise since the number of rows and columns are printed explicitly at the top of the file, so for any display purposes the file knows how to "count" the values into the correct coordinates.

Parameters

<i>file</i>	ppm file to which the densities should be printed.
-------------	--

4.2.3.13 setUniformDistribution()

```
void Grid::setUniformDistribution (
    double predUpperBound,
    double preyUpperbound,
    std::default_random_engine & generator )
```

Sets uniform random predator and prey distribution in each grid cell.

This function works by just calling the [setUniformPredDistribution\(\)](#) and [setUniformPreyDistribution\(\)](#) functions with their respective upper bounds. For a more in depth discussion of the functionality please see those functions where the implementation is explained in full.

Parameters

<i>predUpperBound</i>	floating point value that provides the upper bound for the random number distribution of the predators.
<i>preyUpperBound</i>	floating point value that provides the upper bound for the random number distribution of the prey.
<i>generator</i>	a default_random_engine reference from the std library <random> class.

4.2.3.14 setUniformPredDistribution()

```
void Grid::setUniformPredDistribution (
    double upperBound,
    std::default_random_engine & generator )
```

Sets uniform random predator distribution in each grid cell.

Initially predator density will be zero in each cell since the constructors do no initialization of densities. This function sets the density of predators in every Dry grid cell to a random number between 0 and upperBound according to a uniform distribution. i.e. If upperBound = 5.0, then each dry cell will be assigned a density between 0.0 and 5.0 according to a uniform distribution.

Parameters

<i>upperBound</i>	floating point value that provides the upper bound for the random number distribution.
<i>generator</i>	a default_random_engine reference from the std library <random> class. This is used to generate the random number from the distribution which is a local variable within the function. The generator should be provided by the main method so that chains of random predator densities can be reproduced if required for debugging. The generator is passed as a reference so that if any other functions which makes use of the same distribution are called from the main method, they do not produce the same chain of random numbers, rather they act on the next random number given by the generator.

4.2.3.15 setUniformPreyDistribution()

```
void Grid::setUniformPreyDistribution (
    double upperBound,
    std::default_random_engine & generator )
```

Sets uniform random prey distribution in each grid cell.

Initially prey density will be zero in each cell since the constructors do no initialization of densities. This function sets the density of predators in every Dry grid cell to a random number between 0 and upperBound according to a uniform distribution. i.e. If upperBound = 5.0, then each dry cell will be assigned a density between 0.0 and 5.0 according to a uniform distribution.

Parameters

<i>upperBound</i>	floating point value that provides the upper bound for the random number distribution.
<i>generator</i>	a default_random_engine reference from the std library <random> class. This is used to generate the random number from the distribution which is a local variable within the function. The generator should be provided by the main method so that chains of random prey densities can be reproduced if required for debugging. The generator is passed as a reference so that if any other functions which makes use of the same distribution are called from the main method, they do not produce the same chain of random numbers, rather they act on the next random number given by the generator.

4.2.4 Friends And Related Function Documentation

4.2.4.1 operator<<

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

Operator overload for outputting the grid.

Outputs the [Grid](#) into an output stream in the format: #columns #rows x x x x x x
x x x where x takes on the values of 0 or 1 depending on whether the corresponding (i,j)th cell from the bottom
left corner is Wet or Dry respectively.

This is a friend function since the first operand (argument on left of operator) is not the [Cell](#)

Parameters

<i>out</i>	std::ostream reference which is the output.
<i>grid</i>	const Grid reference which is the grid to be sent into the output stream.

Returns

std::ostream reference that is the same as the out parameter, this is so one can put this operator in a chain of output operators.

4.2.4.2 TestGrid

```
friend class TestGrid [friend]
```

Test class for the grid.

Declare a [TestGrid](#) friend member variable to allow access to private member variables in test class.

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

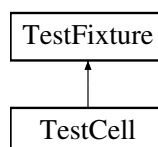
- [src/Grid.hpp](#)
- [src/Grid.cpp](#)

4.3 TestCell Class Reference

Tests the [Cell](#) class.

```
#include <TestCell.hpp>
```

Inheritance diagram for TestCell:



Public Member Functions

- void [setUp](#) ()
Override [setUp\(\)](#), initialise test cases etc.
- void [tearDown](#) ()
Override [tearDown\(\)](#), free allocated memory in test cases, etc.

Protected Member Functions

- void [testGetPredDensity](#) ()
Test method for [Cell::getPredDensity\(\)](#) method.
- void [testGetPreyDensity](#) ()
Test method for [Cell::getPreyDensity\(\)](#) method.
- void [testGetState](#) ()
Test method for [Cell::getState\(\)](#) method.
- void [testSetPosPredDensity](#) ()
Test method for [Cell::setPredDensity\(\)](#) method.
- void [testSetPosPreyDensity](#) ()
Test method for [Cell::setPreyDensity\(\)](#) method.
- void [testSetNegPredDensity](#) ()
Test method for [Cell::setPredDensity\(\)](#) method.
- void [testSetNegPreyDensity](#) ()
Test method for [Cell::setPreyDensity\(\)](#) method.
- void [testSetStateDry](#) ()
Test method for [Cell::setState\(\)](#) method.
- void [testSetStateWet](#) ()
Test method for [Cell::setState\(\)](#) method.
- void [testConstructorWet](#) ()
Test method for [Cell::Cell\(\)](#) method.
- void [testConstructorDry](#) ()
Test method for [Cell::Cell\(\)](#) method.
- void [testConstructorDryNegPredDensity](#) ()
Test method for [Cell::Cell\(\)](#) method.
- void [testConstructorDryNegPreyDensity](#) ()
Test method for [Cell::Cell\(\)](#) method.
- void [testConstructorDefault](#) ()
Test method for [Cell::Cell\(\)](#) method.

4.3.1 Detailed Description

Tests the [Cell](#) class.

This test class is written using the CppUnit test framework and tests the various methods of the [Cell](#) class.

4.3.2 Member Function Documentation

4.3.2.1 testConstructorDefault()

```
void TestCell::testConstructorDefault ( ) [protected]
```

Test method for [Cell::Cell\(\)](#) method.

Tests the default behaviour of the constructor i.e. if no arguments provided should create a wet cell with zero densities.

4.3.2.2 testConstructorDry()

```
void TestCell::testConstructorDry ( ) [protected]
```

Test method for [Cell::Cell\(\)](#) method.

Tests that the constructor creates a cell with correct densities if it is dry.

4.3.2.3 testConstructorDryNegPredDensity()

```
void TestCell::testConstructorDryNegPredDensity ( ) [protected]
```

Test method for [Cell::Cell\(\)](#) method.

Tests that the constructor creates a cell with zero predator density if the predator density provided to the constructor is less than 0.

4.3.2.4 testConstructorDryNegPreyDensity()

```
void TestCell::testConstructorDryNegPreyDensity ( ) [protected]
```

Test method for [Cell::Cell\(\)](#) method.

Tests that the constructor creates a cell with zero prey density if the predator density provided to the constructor is less than 0.

4.3.2.5 testConstructorWet()

```
void TestCell::testConstructorWet ( ) [protected]
```

Test method for [Cell::Cell\(\)](#) method.

Tests that the constructor creates a cell with zero densities if it is wet.

4.3.2.6 testGetPredDensity()

```
void TestCell::testGetPredDensity ( ) [protected]
```

Test method for [Cell::getPredDensity\(\)](#) method.

Tests that the getter returns the correct predator density for the cell.

4.3.2.7 testGetPreyDensity()

```
void TestCell::testGetPreyDensity ( ) [protected]
```

Test method for [Cell::getPreyDensity\(\)](#) method.

Tests that the getter returns the correct prey density for the cell.

4.3.2.8 testGetState()

```
void TestCell::testGetState ( ) [protected]
```

Test method for [Cell::getState\(\)](#) method.

Tests that the getter returns the state for the cell.

4.3.2.9 testSetNegPredDensity()

```
void TestCell::testSetNegPredDensity ( ) [protected]
```

Test method for [Cell::setPredDensity\(\)](#) method.

Tests that the setter sets the correct density for negative arguments.

4.3.2.10 testSetNegPreyDensity()

```
void TestCell::testSetNegPreyDensity ( ) [protected]
```

Test method for [Cell::setPreyDensity\(\)](#) method.

Tests that the setter sets the correct density for negative arguments.

4.3.2.11 testSetPosPredDensity()

```
void TestCell::testSetPosPredDensity ( ) [protected]
```

Test method for [Cell::setPredDensity\(\)](#) method.

Tests that the setter sets the correct density for positive arguments.

4.3.2.12 testSetPosPreyDensity()

```
void TestCell::testSetPosPreyDensity ( ) [protected]
```

Test method for [Cell::setPreyDensity\(\)](#) method.

Tests that the setter sets the correct density for positive arguments.

4.3.2.13 testSetStateDry()

```
void TestCell::testSetStateDry ( ) [protected]
```

Test method for [Cell::setState\(\)](#) method.

Tests that the setter sets the correct state for Cell::Dry arguments.

4.3.2.14 testSetStateWet()

```
void TestCell::testSetStateWet ( ) [protected]
```

Test method for [Cell::setState\(\)](#) method.

Tests that the setter sets the state for Cell::Wet arguments.

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

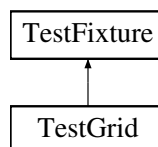
- [test/TestCell.hpp](#)
- [test/TestCell.cpp](#)

4.4 TestGrid Class Reference

Tests the [Grid](#) class.

```
#include <TestGrid.hpp>
```

Inheritance diagram for TestGrid:



Public Member Functions

- void **setUp** ()
- void **tearDown** ()
- void **testConstructor1** ()
- void **testConstructor2** ()

4.4.1 Detailed Description

Tests the [Grid](#) class.

This test class is written using the CppUnit test framework and tests the various methods of the [Grid](#) class.

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

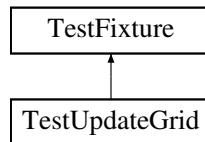
- [test/TestGrid.hpp](#)
- [test/TestGrid.cpp](#)

4.5 TestUpdateGrid Class Reference

Tests the updateGrid function.

```
#include <TestUpdateGrid.hpp>
```

Inheritance diagram for TestUpdateGrid:



Public Member Functions

- void `setUp()` ()
Override `setUp()`, initialise test cases etc.
- void `tearDown()` ()
Override `tearDown()`, free allocated memory in test cases, etc.

Protected Member Functions

- void `testWetGridUpdated()` ()
Test method for `updateGrid()` function.
- void `testZeroDensityGridUpdated()` ()
Test method for `updateGrid()` function.
- void `testRealisticGridUpdated()` ()
Test method for `updateGrid()` function.

4.5.1 Detailed Description

Tests the updateGrid function.

This test class is written using the CppUnit test framework and tests the function that updates the grid.

4.5.2 Member Function Documentation

4.5.2.1 testRealisticGridUpdated()

```
void TestUpdateGrid::testRealisticGridUpdated ( ) [protected]
```

Test method for `updateGrid()` function.

Tests that the function correctly updates a grid that initially has non-zero densities.

4.5.2.2 testWetGridUpdated()

```
void TestUpdateGrid::testWetGridUpdated ( ) [protected]
```

Test method for updateGrid() function.

Tests that the function correctly updates a grid that is wet everywhere.

4.5.2.3 testZeroDensityGridUpdated()

```
void TestUpdateGrid::testZeroDensityGridUpdated ( ) [protected]
```

Test method for updateGrid() function.

Tests that the function correctly updates a grid that initially has zero densities everywhere.

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

- test/[TestUpdateGrid.hpp](#)
- test/TestUpdateGrid.cpp

Chapter 5

File Documentation

5.1 src/Cell.hpp File Reference

Classes

- class [Cell](#)
Models a single cell.

5.2 src/Grid.hpp File Reference

```
#include "Cell.hpp"
#include <fstream>
#include <cassert>
#include <iostream>
#include <random>
#include <stdexcept>
```

Classes

- class [Grid](#)
Models a 2D landscape of cells.

5.3 src/updateGrid.hpp File Reference

Updates the grid by one time step.

```
#include "Grid.hpp"
```

Functions

- [Grid](#) **updateGrid** ([Grid](#) &grid, double r, double a, double b, double m, double k, double l, double deltaT)

5.3.1 Detailed Description

Updates the grid by one time step.

Function to update the densities of predator and prey in a grid instance according to the differential equation specified in the coursework pdf.

Parameters

<i>grid</i>	Grid reference instance to be updated.
<i>r</i>	double instance representing the birth rate of hares (prey).
<i>a</i>	double instance representing the predation rate at which pumas (predators).
<i>b</i>	double instance representing the birth rate of pumas (predators) per one hare (prey) eaten.
<i>m</i>	double instance representing the puma (predator) mortality rate.
<i>k</i>	double instance representing the diffusion rate for hares (prey).
<i>l</i>	double instance representing the diffusion rate for pumas (predators).
<i>deltaT</i>	double instance representing the time step value.

Returns

[Grid](#) instance which is the original grid updated by one time step.

5.4 test/TestCell.hpp File Reference

```
#include <cppunit/extensions/HelperMacros.h>
#include <cppunit/TestFixture.h>
#include "Cell.hpp"
```

Classes

- class [TestCell](#)
Tests the [Cell](#) class.

5.5 test/TestGrid.hpp File Reference

```
#include <cppunit/extensions/HelperMacros.h>
#include <cppunit/TestFixture.h>
#include "Grid.hpp"
```

Classes

- class [TestGrid](#)
Tests the [Grid](#) class.

5.6 test/TestUpdateGrid.hpp File Reference

```
#include <cppunit/extensions/HelperMacros.h>
#include <cppunit/TestFixture.h>
#include "Grid.hpp"
#include "updateGrid.hpp"
```

Classes

- class [TestUpdateGrid](#)
Tests the [updateGrid](#) function.

Index

- ~Grid
 - Grid, [14](#)
- Cell, [7](#)
 - Cell, [8](#)
 - getPredDensity, [8](#)
 - getPreyDensity, [8](#)
 - getState, [9](#)
 - setPredDensity, [9](#)
 - setPreyDensity, [9](#)
 - setState, [10](#)
 - TestCell, [10](#)
- dryNeighbours
 - Grid, [14](#)
- getCellState
 - Grid, [15](#)
- getColumns
 - Grid, [15](#)
- getPredDensity
 - Cell, [8](#)
- getPreyDensity
 - Cell, [8](#)
- getRows
 - Grid, [15](#)
- getState
 - Cell, [9](#)
- Grid, [10](#)
 - ~Grid, [14](#)
 - dryNeighbours, [14](#)
 - getCellState, [15](#)
 - getColumns, [15](#)
 - getRows, [15](#)
 - Grid, [12](#), [14](#)
 - operator<<, [22](#)
 - operator(), [15](#), [16](#)
 - operator=, [16](#), [17](#)
 - predDensity, [17](#)
 - preyDensity, [19](#)
 - printDensities, [19](#)
 - printPPM, [20](#)
 - setUniformDistribution, [20](#)
 - setUniformPredDistribution, [21](#)
 - setUniformPreyDistribution, [21](#)
 - TestGrid, [22](#)
- operator<<
 - Grid, [22](#)
- operator()
 - Grid, [15](#), [16](#)
- operator=
 - Grid, [16](#), [17](#)
- predDensity
 - Grid, [17](#)
- preyDensity
 - Grid, [19](#)
- printDensities
 - Grid, [19](#)
- printPPM
 - Grid, [20](#)
- setPredDensity
 - Cell, [9](#)
- setPreyDensity
 - Cell, [9](#)
- setState
 - Cell, [10](#)
- setUniformDistribution
 - Grid, [20](#)
- setUniformPredDistribution
 - Grid, [21](#)
- setUniformPreyDistribution
 - Grid, [21](#)
- src/Cell.hpp, [29](#)
- src/Grid.hpp, [29](#)
- src/updateGrid.hpp, [29](#)
- test/TestCell.hpp, [30](#)
- test/TestGrid.hpp, [30](#)
- test/TestUpdateGrid.hpp, [30](#)
- TestCell, [22](#)
 - Cell, [10](#)
 - testConstructorDefault, [23](#)
 - testConstructorDry, [24](#)
 - testConstructorDryNegPredDensity, [24](#)
 - testConstructorDryNegPreyDensity, [24](#)
 - testConstructorWet, [24](#)
 - testGetPredDensity, [24](#)
 - testGetPreyDensity, [24](#)
 - testGetState, [25](#)
 - testSetNegPredDensity, [25](#)
 - testSetNegPreyDensity, [25](#)
 - testSetPosPredDensity, [25](#)
 - testSetPosPreyDensity, [25](#)
 - testSetStateDry, [25](#)
 - testSetStateWet, [26](#)
- testConstructorDefault
 - TestCell, [23](#)

- testConstructorDry
 - TestCell, [24](#)
- testConstructorDryNegPredDensity
 - TestCell, [24](#)
- testConstructorDryNegPreyDensity
 - TestCell, [24](#)
- testConstructorWet
 - TestCell, [24](#)
- testGetPredDensity
 - TestCell, [24](#)
- testGetPreyDensity
 - TestCell, [24](#)
- testGetState
 - TestCell, [25](#)
- TestGrid, [26](#)
 - Grid, [22](#)
- testRealisticGridUpdated
 - TestUpdateGrid, [27](#)
- testSetNegPredDensity
 - TestCell, [25](#)
- testSetNegPreyDensity
 - TestCell, [25](#)
- testSetPosPredDensity
 - TestCell, [25](#)
- testSetPosPreyDensity
 - TestCell, [25](#)
- testSetStateDry
 - TestCell, [25](#)
- testSetStateWet
 - TestCell, [26](#)
- TestUpdateGrid, [27](#)
 - testRealisticGridUpdated, [27](#)
 - testWetGridUpdated, [27](#)
 - testZeroDensityGridUpdated, [28](#)
- testWetGridUpdated
 - TestUpdateGrid, [27](#)
- testZeroDensityGridUpdated
 - TestUpdateGrid, [28](#)