2D Predator-Prey Model

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

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.

2.1.1 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)

2.1.2 Constructor & Destructor Documentation

2.1.2.1 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

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

2.1.3 Member Function Documentation

2.1.3.1 getPredDensity()

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

Getter for the predator density in the cell.

Returns

The value of m_predDensity.

2.1.3.2 getPreyDensity()

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

Getter for the prey density in the cell.

Returns

The value of m_preyDensity.

2.1.3.3 getState()

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

Getter for the state of the cell.

Returns

The value of m_state.

2.1.3.4 setPredDensity()

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

Setter for the predator density in the cell.

Parameters

predDensity	a floating point value setting the density of the predators in the cell. Defaults to 0.

2.1.3.5 setPreyDensity()

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

Setter for the prey density in the cell.

Parameters

predDensity	a floating point value setting the density of the prey in the cell. Defaults to 0.

2.1.3.6 setState()

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

Setter for the state of the cell.

Parameters

state	a Cell::State enum value setting the state of the cell to be either Wet or Dry. Defaults to Wet.

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

- source/Cell.hpp
- source/Cell.cpp

2.2 Grid Class Reference

Models a 2D landscape of cells.

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

Public Member Functions

- Grid ()
  Default constructor.
- Grid (std::ifstream &inputFile)
  Creates Grid objects from some input file representing the Grid.
- Grid (int columns, int rows, const int *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 uppperBound, 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

  Calulates average predator density across the grid.
- double preyDensity (bool includeWetCells=true) const

  Calulates average prey density across the grid.
- Cell & operator() (int i, int j)
- const Cell & operator() (int i, int j) const

  Second operator overload to acess the Cell stored at the (i,j)th coordinate if the grid is a constant variable.
- int dryNeighbours (int i, int j) const

  Calulates number of Dry neighbours of a given cell.
- void printDensities (std::ostream &out) const

  Outputs predator and prey densities to output stream.

Friends

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

  Operator overload for outputting the grid.

2.2.1 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.

2.2.2 Constructor & Destructor Documentation

2.2.2.1 Grid() [1/5]

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

Default constructor.

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

2.2.2.2 Grid() [2/5]

```
Grid::Grid (
            std::ifstream & inputFile )
```

Creates Grid objects from some input file 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.

2.2.2.3 Grid() [3/5]

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

columns	Integer value represnting the number of columns in the Grid object, not inluding the halo.
rows	Integer value representing the number of rows in the Grid objecet, not including the halo.
data	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

2.2.2.4 Grid() [4/5]

```
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

sourceGrid	constant Grid reference from which the deep copying will be done.

2.2.2.5 Grid() [5/5]

```
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

sourceGrid	constant R-value reference from which the ownership of the member variables will be transfered.

2.2.2.6 ~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.

2.2.3 Member Function Documentation

2.2.3.1 dryNeighbours()

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

Calulates 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	column number/x-coordinate of cell.
j	row number/y-corrdinate of cell.

Returns

Integer value representing number of Dry neighbours of the cell.

2.2.3.2 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.

2.2.3.3 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.

2.2.3.4 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 implemeted as a 1-D array with the same number of elements for memory reasons, this opertor provides a way of acessing the elements of the 1-D array with two indicies (i,j) as if it were a 2D-array. This operator should be used whenever direct acess 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 orgin 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	column number/x-coordinate of cell.
j	row/y-coordinate of cell.

Returns

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

2.2.3.5 operator()() [2/2]

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

Second operator overload to acess 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	column number/x-coordinate of cell.
j	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.

2.2.3.6 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 opeartor 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

sourceGrid	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.

2.2.3.7 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

sourceGrid	constant R-value Grid reference from which the ownership of the member variables will be transfered.

Returns

Grid reference ∗this, so that the newly assigned operator may be chained into other assignment operations.

2.2.3.8 predDensity()

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

Calulates average predator density across the grid.

Calulates 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

includeWetCells	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.

Add the pred densities of each cell up.

Total up the number of Wet cells as well.

2.2.3.9 preyDensity()

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

Calulates average prey density across the grid.

Calulates 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

includeWetCells	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.

2.2.3.10 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

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

2.2.3.11 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 setUniormPreyDistribution() functions with their respective upper bounds. For a more in depth discussion of the functionality plesae see those functions where the implementation is explained in full.

Parameters

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

2.2.3.12 setUniformPredDistribution()

```
void Grid::setUniformPredDistribution (
```

```
            double uppperBound,
            std::default_random_engine & generator )
```

Sets uniform random predator distribution in each grid cell.

Intially predator density will be zero in each cell since the constructors do no intialization 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

upperBound	floating point value that provides the upper bound for the random number distriubtion.
generator	a default_random_engine refernece from the std library <random> class. This is used to generate the random number from the distriubtion 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.

2.2.3.13 setUniformPreyDistribution()

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

Sets uniform random prey distribution in each grid cell.

Intially prey density will be zero in each cell since the constructors do no intialization 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

upperBound	floating point value that provides the upper bound for the random number distriubtion.
generator	a default_random_engine refernece from the std library <random> class. This is used to generate the random number from the distriubtion 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.

2.2.4 Friends And Related Function Documentation

2.2.4.1 operator<<

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

Operator overload for outputting the grid.

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

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

Parameters

out	std::ostream reference which is the output.
grid	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 opearator in a chain of output operators.

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

- source/Grid.hpp
- source/Grid.cpp

Index