GameOfLife

1.0

Generated by Doxygen 1.10.0

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 Array1D Class Reference

A class for 1D arrays.

`#include <Array1D.hpp>`

Public Member Functions

- Array1D (int size)
  Constructor.
- ~Array1D ()
  Destructor.
- int & operator() (int i)
  Overload the () operator to access the data.
- void overwrite (Array1D arr, int shift=0)
- void copy_into (Array1D *arr)
- Array1D sub_arr (int i_low, int i_upp)
- void display ()
  Display the data of the array.

Public Attributes

- int size
  Size of the array.
- int * data
  Pointer to the data.

4.1.1 Detailed Description

A class for 1D arrays.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Array1D()

```
Array1D::Array1D (
            int size )  [inline]
```

Constructor.

4.1.2.2 ~Array1D()

```
Array1D::~Array1D ( )  [inline]
```

Destructor.

4.1.3 Member Function Documentation

4.1.3.1 copy_into()

```
void Array1D::copy_into (
            Array1D * arr )  [inline]
```

Copy the data of the array into another array.

Parameters

arr	The array from which the data is to be copied. Accessed by reference.

4.1.3.2 display()

```
void Array1D::display ( )  [inline]
```

Display the data of the array.

4.1.3.3 operator()()

```
int & Array1D::operator() (
            int i )  [inline]
```

Overload the () operator to access the data.

4.1.3.4 overwrite()

```
void Array1D::overwrite (
            Array1D arr,
            int shift = 0 )  [inline]
```

Overwrite the data of the array with the data of another array

Parameters

arr	The array to be copied into the current array
shift	The shift with which the array to be copied is loaded in the current array. If non-zero, arr needs to be smaller than the current array.

4.1.3.5 sub_arr()

```
Array1D Array1D::sub_arr (
            int i_low,
            int i_upp )  [inline]
```

Create a subarray of the current array.

Parameters

i_low	The lower index of the subarray
i_upp	The upper index of the subarray

4.1.4 Member Data Documentation

4.1.4.1 data

```
int* Array1D::data
```

Pointer to the data.

4.1.4.2 size

```
int Array1D::size
```

Size of the array.

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

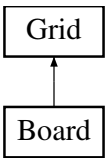
- src/lib/Array1D.hpp

4.2 Board Class Reference

A class inheriting from Grid, that adds the functionality to update the board.

```
#include <Board.hpp>
```

Inheritance diagram for Board:



Public Member Functions

- Board (int N_row, int N_col)

  Constructor.
- void init_from_motherboard (Grid *motherboard, int row_low, int col_left)

  Initialize the board with values from (a subgrid of) the motherboard.
- void set_bottom_ghost_row (Array1D *target)

  Set the bottom ghost row based on an input array.
- void set_upper_ghost_row (Array1D *target)

  Set the upper ghost row based on an input array.
- void set_left_ghost_col (Array1D *target)

  Set the left ghost column based on an input array.
- void set_right_ghost_col (Array1D *target)

  Set the right ghost column based on an input array.
- void store_neighbour_row (Array1D *store, int n_row)

  Store the neighbour counts of a row in an array.
- void store_upper_ghost_neighbour_row (Array1D *store)

  Store the neighbour counts of the upper ghost row in an array.
- void store_bottom_ghost_neighbour_row (Array1D *store)

  Store the neighbour counts of the bottom ghost row in an array.
- void ghost_display ()

  Display the board, including the ghost rows and columns.
- void update_board ()

  Update the board based on the rules of the game of life.

Public Member Functions inherited from Grid

- Grid (int N_row, int N_col, int N_nb_crit=3)

  Constructor.
- ~Grid ()

  Destructor.
- int & operator() (int i, int j)

  Overload the () operator to access the data.
- void store_row (Array1D *store, int n_row, int shift=0)
- void store_col (Array1D *store, int n_col)
- Array1D sub_row (int n_row, int i_low, int i_upp)
- Array1D sub_col (int n_col, int i_low, int i_upp)
- void display ()

  Display the data of the grid.
- Array1D periodic_row (int n_row)
- void save (std::string file)
- void store_data (int *arr)
- void read_data (int *arr)
- void overwrite_sub_board (int *arr, int row_low, int row_upp, int col_low, int col_upp)

Public Attributes

- Array1D bottom_ghost_row

  The ghost row at the bottom of the grid, including the corners.
- Array1D upper_ghost_row

  The ghost row at the top of the grid, including the corners.
- Array1D left_ghost_col

  The ghost column on the left side of the grid.
- Array1D right_ghost_col

  The ghost column on the right side of the grid.
- Array1D temp1

  Storage arrays to hold the horizontal neighbours counts in a row.
- Array1D temp2
- Array1D temp3

Public Attributes inherited from Grid

- int N_row

  Number of rows in the grid.
- int N_col

  Number of columns in the grid.
- int * data

  Pointer to the data.
- int N_nb_crit

  Number of critical neighbours used in the game rules.
- int size

  Number of rows times the number of columns.

4.2.1 Detailed Description

A class inheriting from Grid, that adds the functionality to update the board.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Board()

```
Board::Board (
            int N_row,
            int N_col )  [inline]
```

Constructor.

Parameters

N_row	The number of rows in the grid
N_col	The number of columns in the grid

4.2.3 Member Function Documentation

4.2.3.1 ghost_display()

```
void Board::ghost_display ( )  [inline]
```

Display the board, including the ghost rows and columns.

4.2.3.2 init_from_motherboard()

```
void Board::init_from_motherboard (
            Grid * motherboard,
            int row_low,
            int col_left )  [inline]
```

Initialize the board with values from (a subgrid of) the motherboard.

Parameters

motherboard	The motherboard grid to copy values from
row_low	The lowest row index to copy from the motherboard
col_left	The leftmost column index to copy from the motherboard

4.2.3.3 set_bottom_ghost_row()

```
void Board::set_bottom_ghost_row (
            Array1D * target )  [inline]
```

Set the bottom ghost row based on an input array.

Parameters

target	The array to copy values from

4.2.3.4 set_left_ghost_col()

```
void Board::set_left_ghost_col (
            Array1D * target )  [inline]
```

Set the left ghost column based on an input array.

Parameters

target	The array to copy values from

4.2.3.5 set_right_ghost_col()

```
void Board::set_right_ghost_col (
            Array1D * target )  [inline]
```

Set the right ghost column based on an input array.

Parameters

target	The array to copy values from

4.2.3.6 set_upper_ghost_row()

```
void Board::set_upper_ghost_row (
            Array1D * target )  [inline]
```

Set the upper ghost row based on an input array.

Parameters

target	The array to copy values from

4.2.3.7 store_bottom_ghost_neighbour_row()

```
void Board::store_bottom_ghost_neighbour_row (
            Array1D * store )  [inline]
```

Store the neighbour counts of the bottom ghost row in an array.

Parameters

store	The array to store the neighbour counts in

4.2.3.8 store_neighbour_row()

```
void Board::store_neighbour_row (
            Array1D * store,
            int n_row )  [inline]
```

Store the neighbour counts of a row in an array.

Parameters

store	The array to store the neighbour counts in
n_row	The row index to store the neighbour counts of

4.2.3.9 store_upper_ghost_neighbour_row()

```
void Board::store_upper_ghost_neighbour_row (
            Array1D * store )  [inline]
```

Store the neighbour counts of the upper ghost row in an array.

Parameters

store	The array to store the neighbour counts in

4.2.3.10 update_board()

```
void Board::update_board ( )  [inline]
```

Update the board based on the rules of the game of life.

4.2.4 Member Data Documentation

4.2.4.1 bottom_ghost_row

```
Array1D Board::bottom_ghost_row
```

The ghost row at the bottom of the grid, including the corners.

4.2.4.2 left_ghost_col

```
Array1D Board::left_ghost_col
```

The ghost column on the left side of the grid.

4.2.4.3 right_ghost_col

```
Array1D Board::right_ghost_col
```

The ghost column on the right side of the grid.

4.2.4.4 temp1

```
Array1D Board::temp1
```

Storage arrays to hold the horizontal neighbours counts in a row.

4.2.4.5 temp2

```
Array1D Board::temp2
```

4.2.4.6 temp3

```
Array1D Board::temp3
```

4.2.4.7 upper_ghost_row

```
Array1D Board::upper_ghost_row
```

The ghost row at the top of the grid, including the corners.

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

- src/lib/Board.hpp

4.3 GameParams Class Reference

A class that stores the parameters for the Game of Life.

```
#include <GameParams.hpp>
```

Public Member Functions

- GameParams ()
  Default constructor.
- void readParams (const std::string &filename)
- void display () const
  Function that displays the parameters.

Public Attributes

- int board_size {10}
  The size of the board.
- int N_critical {3}
  The number of critical neighbours for a cell to survive.
- int save_interval {1}
  The interval at which the board is saved.
- int evolve_steps {20}
  The number of steps over which the board is evolved.
- int random_data {1}
  Whether to initialize the board with random data or from a file. 1: random, 0: file (board_file)
- int num_threads {1}
  The number of OMP threads to use.
- double prob_live {0.5}
  The probability that a cell is alive at the start, parameter in a Binomial distribution.
- std::string board_file {"examples/"}
  The path to the initialization file, in case random_data is 0.
- std::string output_path {"examples/"}
  The path where to store the output files.

4.3.1 Detailed Description

A class that stores the parameters for the Game of Life.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 GameParams()

```
GameParams::GameParams ( )  [inline]
```

Default constructor.

4.3.3 Member Function Documentation

4.3.3.1 display()

```
void GameParams::display ( ) const  [inline]
```

Function that displays the parameters.

4.3.3.2 readParams()

```
void GameParams::readParams (
            const std::string & filename )  [inline]
```

Function that reads the parameters from a text file

Parameters

filename	path to params file, parsed through command line

4.3.4 Member Data Documentation

4.3.4.1 board_file

```
std::string GameParams::board_file {"examples/"}
```

The path to the initialization file, in case random_data is 0.

4.3.4.2 board_size

```
int GameParams::board_size {10}
```

The size of the board.

4.3.4.3 evolve_steps

```
int GameParams::evolve_steps {20}
```

The number of steps over which the board is evolved.

4.3.4.4 N_critical

```
int GameParams::N_critical {3}
```

The number of critical neighbours for a cell to survive.

4.3.4.5 num_threads

```
int GameParams::num_threads {1}
```

The number of OMP threads to use.

4.3.4.6 output_path

```
std::string GameParams::output_path {"examples/"}
```

The path where to store the output files.

4.3.4.7 prob_live

```
double GameParams::prob_live {0.5}
```

The probability that a cell is alive at the start, parameter in a Binomial distribution.

4.3.4.8 random_data

```
int GameParams::random_data {1}
```

Whether to initialize the board with random data or from a file. 1: random, 0: file (board_file)

4.3.4.9 save_interval

```
int GameParams::save_interval {1}
```

The interval at which the board is saved.

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

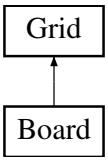
- src/lib/GameParams.hpp

4.4 Grid Class Reference

A class for a 2D grid that contains the entire board for the Game of Life.

`#include <Grid.hpp>`

Inheritance diagram for Grid:



Public Member Functions

- Grid (int N_row, int N_col, int N_nb_crit=3)
  Constructor.
- ∼Grid ()
  Destructor.
- int & operator() (int i, int j)
  Overload the () operator to access the data.
- void store_row (Array1D ∗store, int n_row, int shift=0)
- void store_col (Array1D ∗store, int n_col)
- Array1D sub_row (int n_row, int i_low, int i_upp)
- Array1D sub_col (int n_col, int i_low, int i_upp)
- void display ()
  Display the data of the grid.
- Array1D periodic_row (int n_row)
- void save (std::string file)
- void store_data (int ∗arr)
- void read_data (int ∗arr)
- void overwrite_sub_board (int ∗arr, int row_low, int row_upp, int col_low, int col_upp)

Public Attributes

- int N_row
  Number of rows in the grid.
- int N_col
  Number of columns in the grid.
- int ∗ data
  Pointer to the data.
- int N_nb_crit
  Number of critical neighbours used in the game rules.
- int size
  Number of rows times the number of columns.

4.4.1 Detailed Description

A class for a 2D grid that contains the entire board for the Game of Life.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Grid()

```
Grid::Grid (
            int N_row,
            int N_col,
            int N_nb_crit = 3 )  [inline]
```

Constructor.

4.4.2.2 ~Grid()

```
Grid::~Grid ( )  [inline]
```

Destructor.

4.4.3 Member Function Documentation

4.4.3.1 display()

```
void Grid::display ( )  [inline]
```

Display the data of the grid.

4.4.3.2 operator()()

```
int & Grid::operator() (
            int i,
            int j )  [inline]
```

Overload the () operator to access the data.

4.4.3.3 overwrite_sub_board()

```
void Grid::overwrite_sub_board (
            int * arr,
            int row_low,
            int row_upp,
            int col_low,
            int col_upp )  [inline]
```

Overwrite a subgrid of the grid with the data in an array

Parameters

arr	The array from which the data is to be copied
row_low	The index of the lower row of the subgrid
row_upp	The index of the upper row of the subgrid
col_low	The index of the lower column of the subgrid
col_upp	The index of the upper column of the subgrid

4.4.3.4 periodic_row()

```
Array1D Grid::periodic_row (
            int n_row )  [inline]
```

Return a row, with one cell added to the left and right, for periodic boundary conditions

Parameters

n_row	The index of the row to be returned with the additional cells

4.4.3.5 read_data()

```
void Grid::read_data (
            int * arr )  [inline]
```

Read the data of the grid from an array

Parameters

arr	The array from which the data is to be read

4.4.3.6 save()

```
void Grid::save (
            std::string file )  [inline]
```

Save the data of the grid to a file

Parameters

file	The name of the file to which the data is to be saved

4.4.3.7 store_col()

```
void Grid::store_col (
            Array1D * store,
            int n_col )  [inline]
```

Store a column of the grid in an Array1D object

Parameters

store	The Array1D object in which the column is to be stored
n_col	The index of the column to be stored

4.4.3.8 store_data()

```
void Grid::store_data (
            int * arr )  [inline]
```

Store the data of the grid in an array

Parameters

arr	The array in which the data is to be stored

4.4.3.9 store_row()

```
void Grid::store_row (
            Array1D * store,
            int n_row,
            int shift = 0 )  [inline]
```

Store a row of the grid in an Array1D object

Parameters

store	The Array1D object in which the row is to be stored
n_row	The index of the row to be stored
shift	The shift with which the row is loaded in the Array1D object

4.4.3.10 sub_col()

```
Array1D Grid::sub_col (
            int n_col,
            int i_low,
            int i_upp )  [inline]
```

Return a subarray of a given column

Parameters

n_col	The index of the column from which the subarray is to be taken
i_low	The lower index of the subarray
i_upp	The upper index of the subarray

4.4.3.11 sub_row()

```
Array1D Grid::sub_row (
            int n_row,
            int i_low,
            int i_upp )  [inline]
```

Return a subarray of a given row

Parameters

n_row	The index of the row from which the subarray is to be taken
i_low	The lower index of the subarray
i_upp	The upper index of the subarray

4.4.4 Member Data Documentation

4.4.4.1 data

```
int* Grid::data
```

Pointer to the data.

4.4.4.2 N_col

```
int Grid::N_col
```

Number of columns in the grid.

4.4.4.3 N_nb_crit

```
int Grid::N_nb_crit
```

Number of critical neighbours used in the game rules.

4.4.4.4 N_row

```
int Grid::N_row
```

Number of rows in the grid.

4.4.4.5 size

```
int Grid::size
```

Number of rows times the number of columns.

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

- src/lib/Grid.hpp

Chapter 5

File Documentation

5.1 src/lib/Array1D.hpp File Reference

```
#include <iostream>
```

Classes

- class Array1D

  A class for 1D arrays.

5.2 Array1D.hpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002
00003 #ifndef ARRAY1D_HPP
00004 #define ARRAY1D_HPP
00005
00007 class Array1D{
00008     public:
00010         int size;
00012         int* data;
00013
00015         Array1D(int size) {
00016             this -> size = size;
00017             this -> data = new int[size];
00018         }
00020         ~Array1D(){
00021             delete[] this -> data;
00022         }
00024         int& operator()(int i){
00025             return this -> data[i];
00026         }
00027
00031         void overwrite(Array1D arr, int shift = 0){
00032             for (int i = 0; i < arr.size; ++i){
00033                 data[i + shift] = arr(i);
00034             }
00035         }
00036
00039         void copy_into(Array1D* arr){
00040             for (int i = 0; i < size; ++i){
00041                 data[i] = (*arr)(i);
00042             }
00043         }
00044
00048         Array1D sub_arr(int i_low, int i_upp){
```

```
00049             int len;
00050             if (i_low > i_upp){
00051                 len = size + i_upp - i_low;
00052             } else {
00053                 len = i_upp - i_low;
00054             }
00055             Array1D sub(len);
00056             for (int i = 0; i < len; ++i) {
00057                 sub(i) = data[(i_low + i) % size];
00058             }
00059             return sub;
00060         }
00061 
00063         void display() {
00064             for (int i = 0; i < size; ++i) {
00065                     std::cout « data[i] « " ";
00066             }
00067             std::cout « std::endl;
00068         }
00069 };
00070 
00071 #endif
```

5.3 src/lib/Board.hpp File Reference

```
#include <iostream>
#include <fstream>
#include <omp.h>
#include "Array1D.hpp"
#include "Grid.hpp"
#include <cassert>
```

Classes

- class Board

  A class inheriting from Grid, that adds the functionality to update the board.

Macros

- #define BOARD_HPP

5.3.1 Macro Definition Documentation

5.3.1.1 BOARD_HPP

```
#define BOARD_HPP
```

5.4 Board.hpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include <fstream>
00003 #include <omp.h>
00004 #include "Array1D.hpp"
00005 #include "Grid.hpp"
00006 #include <cassert>
00007
00008 #ifndef BOARD_HPP
00009 #define BOARD_HPP
00010
00012 class Board : public Grid{
00013     public:
00015         Array1D bottom_ghost_row;
00017         Array1D upper_ghost_row;
00019         Array1D left_ghost_col;
00021         Array1D right_ghost_col;
00022
00024         Array1D temp1, temp2, temp3;
00025
00027
00031         Board(int N_row, int N_col) : Grid(N_row, N_col),
00032         bottom_ghost_row(N_col+2), upper_ghost_row(N_col+2), left_ghost_col(N_row),
      right_ghost_col(N_row),
00033          temp1(N_col), temp2(N_col), temp3(N_col) {
00034             // Check if the grid is large enough to be sensible in the update procedure.
00035             assert (N_row > 2 && N_col > 2);
00036         }
00037
00039
00044         void init_from_motherboard(Grid* motherboard, int row_low, int col_left){
00045             N_nb_crit = (*motherboard).N_nb_crit;
00046             #pragma omp parallel for collapse(2)
00047                 for (int i = 0; i < N_row; ++i) {
00048                     for (int j = 0; j < N_col; ++j) {
00049                         data[i*N_col+j] = (*motherboard)(row_low + i, col_left + j);
00050                     }
00051                 }
00052         }
00053
00055
00058         void set_bottom_ghost_row(Array1D* target) {
00059             assert (target->size == N_col+2);
00060             #pragma omp parallel for
00061             for (int i = 0; i < N_col+2; ++i) {
00062                 bottom_ghost_row(i) = (*target)(i);
00063             }
00064         }
00066
00069         void set_upper_ghost_row(Array1D* target) {
00070             assert (target->size == N_col+2);
00071             #pragma omp parallel for
00072             for (int i = 0; i < N_col+2; ++i) {
00073                 upper_ghost_row(i) = (*target)(i);
00074             }
00075         }
00077
00080         void set_left_ghost_col(Array1D* target) {
00081             assert (target->size == N_row);
00082             #pragma omp parallel for
00083             for (int i = 0; i < N_col; ++i) {
00084                 left_ghost_col(i) = (*target)(i);
00085             }
00086         }
00088
00091         void set_right_ghost_col(Array1D* target) {
00092             assert (target->size == N_row);
00093             #pragma omp parallel for
00094             for (int i = 0; i < N_col; ++i) {
00095                 right_ghost_col(i) = (*target)(i);
00096             }
00097         }
00099
00103         void store_neighbour_row(Array1D* store, int n_row) {
00104             (*store)(0) = left_ghost_col(n_row) + data[n_row * N_col + 0] + data[n_row * N_col + 1];
00105             #pragma omp parallel for
00106             for (int i = 1; i < N_col-1; ++i) {
00107                 (*store)(i) = data[n_row * N_col + i-1] + data[n_row * N_col + i]
00108                                 + data[n_row * N_col + i + 1];
00109             }
00110             (*store)(N_col - 1) = data[n_row * N_col + N_col - 2] + data[n_row * N_col + N_col - 1] +
      right_ghost_col(n_row);
00111         }
```

```
00113
00116         void store_upper_ghost_neighbour_row(Array1D* store) {
00117             #pragma omp parallel for
00118             for (int i = 0; i < N_col; ++i) {
00119                 (*store)(i) = upper_ghost_row(i) + upper_ghost_row(i+1) + upper_ghost_row(i+2);
00120             }
00121         }
00123
00126         void store_bottom_ghost_neighbour_row(Array1D* store) {
00127             #pragma omp parallel for
00128             for (int i = 0; i < N_col; ++i) {
00129                 (*store)(i) = bottom_ghost_row(i) + bottom_ghost_row(i+1) + bottom_ghost_row(i+2);
00130             }
00131         }
00133         void ghost_display(){
00134             upper_ghost_row.display();
00135             for (int i = 0; i < N_row; ++i) {
00136                 std::cout « left_ghost_col(i) « " ";
00137                 for (int j = 0; j < N_col; ++j) {
00138                     std::cout « data[i*N_col+j] « " ";
00139                 }
00140                 std::cout « right_ghost_col(i) « std::endl;
00141             }
00142             bottom_ghost_row.display();
00143         }
00145         void update_board(){
00146
00147             // Storage
00148             int N_nb {0};
00149             int val {0};
00150
00151             // Start with the top row, which requires the neighbours of the upper ghost row
00152             store_upper_ghost_neighbour_row(&temp1);
00153             store_neighbour_row(&temp2, 0);
00154             store_neighbour_row(&temp3, 1);
00155             #pragma omp parallel for
00156                 for (int j = 0; j < N_col; ++j) {
00157                     val = data[j];
00158                     N_nb = temp1(j) + temp2(j) + temp3(j) - val;
00159                     data[j] = (1 - val) * (N_nb == N_nb_crit) + val * (N_nb == N_nb_crit || N_nb ==
      N_nb_crit - 1);
00160                 }
00161
00162             // Then the middle rows
00163             for (int i = 1; i < N_row -1; ++i){
00164                 temp1.copy_into(&temp2);
00165                 temp2.copy_into(&temp3);
00166                 store_neighbour_row(&temp3, i+1);
00167                 #pragma omp parallel for
00168                     for (int j = 0; j < N_col; ++j) {
00169                         val = data[i*N_col + j];
00170                         N_nb = temp1(j) + temp2(j) + temp3(j) - val;
00171                         data[i*N_col + j] = (1 - val) * (N_nb == N_nb_crit) + val * (N_nb == N_nb_crit
      || N_nb == N_nb_crit - 1);
00172                     }
00173             }
00174
00175             // Finally the bottom row, which requires the neighbours of the bottom ghost row
00176             temp1.copy_into(&temp2);
00177             temp2.copy_into(&temp3);
00178             store_bottom_ghost_neighbour_row(&temp3);
00179             #pragma omp parallel for
00180                 for (int j = 0; j < N_col; ++j) {
00181                     val = data[(N_row - 1)*N_col + j];
00182                     N_nb = temp1(j) + temp2(j) + temp3(j) - val;
00183                     data[(N_row - 1)*N_col + j] = (1 - val) * (N_nb == N_nb_crit) + val * (N_nb ==
      N_nb_crit || N_nb == N_nb_crit - 1);
00184                 }
00185         }
00186 };
00187
00188 #endif
```

5.5 src/lib/Functions.cpp File Reference

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <random>
```

```
#include <omp.h>
#include "Board.hpp"
#include "GameParams.hpp"
#include "Grid.hpp"
#include "Array1D.hpp"
```

Functions

- void initialize_random (Grid *grid, GameParams *params)

  Initialize the board with random data.
- void initialize_from_file (Grid *grid, GameParams *params, std::string file)

  Initialize the board from a file.
- void iteration_one_board (Board *board, GameParams *params, Array1D *store_row, Array1D *store_col)

  Update the board for a given number of steps.

5.5.1 Function Documentation

5.5.1.1 initialize_from_file()

```
void initialize_from_file (
            Grid * grid,
            GameParams * params,
            std::string file )
```

Initialize the board from a file.

Parameters

grid	The grid to be initialized
params	The parameters for the game
file	The file to read the data from

5.5.1.2 initialize_random()

```
void initialize_random (
            Grid * grid,
            GameParams * params )
```

Initialize the board with random data.

Parameters

grid	The grid to be initialized
params	The parameters for the game

5.5.1.3 iteration_one_board()

```
void iteration_one_board (
            Board * board,
            GameParams * params,
            Array1D * store_row,
            Array1D * store_col )
```

Update the board for a given number of steps.

Parameters

board	The board to be updated
params	The parameters for the game, including the number of evolve steps
store_row	An array to store ghost rows
store_col	An array to store ghost columns

5.6 src/lib/Functions.hpp File Reference

```
#include "Board.hpp"
#include "GameParams.hpp"
```

Functions

- void initialize_random (Grid *grid, GameParams *params)

  Initialize the board with random data.
- void initialize_from_file (Grid *grid, GameParams *params, std::string file)

  Initialize the board from a file.
- void iteration_one_board (Board *board, GameParams *params, Array1D *store_row, Array1D *store_col)

  Update the board for a given number of steps.

5.6.1 Function Documentation

5.6.1.1 initialize_from_file()

```
void initialize_from_file (
            Grid * grid,
            GameParams * params,
            std::string file )
```

Initialize the board from a file.

Parameters

grid	The grid to be initialized
params	The parameters for the game
file	The file to read the data from

5.6.1.2 initialize_random()

```
void initialize_random (
            Grid * grid,
            GameParams * params )
```

Initialize the board with random data.

Parameters

grid	The grid to be initialized
params	The parameters for the game

5.6.1.3 iteration_one_board()

```
void iteration_one_board (
            Board * board,
            GameParams * params,
            Array1D * store_row,
            Array1D * store_col )
```

Update the board for a given number of steps.

Parameters

board	The board to be updated
params	The parameters for the game, including the number of evolve steps
store_row	An array to store ghost rows
store_col	An array to store ghost columns

5.7 Functions.hpp

Go to the documentation of this file.

```
00001 #ifndef FUNCTIONS_HPP
00002 #define FUNCTIONS_HPP
00003
00004 #include "Board.hpp"
00005 #include "GameParams.hpp"
00006
00007
00008 void initialize_random(Grid* grid, GameParams* params);
00009
00010 void initialize_from_file(Grid* grid, GameParams* params, std::string file);
00011
00012 void iteration_one_board(Board* board, GameParams* params, Array1D* store_row, Array1D* store_col);
00013
00014
00015
00016
00017
00018
00019
00020
00021
00022 #endif
```

5.8 src/lib/GameParams.hpp File Reference

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
```

Classes

- class GameParams

  A class that stores the parameters for the Game of Life.

5.9 GameParams.hpp

Go to the documentation of this file.

```
00001 #ifndef GAMEPARAMS_HPP
00002 #define GAMEPARAMS_HPP
00003
00004 #include <iostream>
00005 #include <fstream>
00006 #include <sstream>
00007 #include <string>
00008
00010 class GameParams{
00011     public:
00012
00014         int board_size {10};
00016         int N_critical {3};
00018         int save_interval {1};
00020         int evolve_steps {20};
00022         int random_data {1};
00024         int num_threads {1};
00026         double prob_live {0.5};
00028         std::string board_file {"examples/"};
00030         std::string output_path {"examples/"};
00031
00033         GameParams(){}
00034
00037         void readParams(const std::string& filename){
00038
00039             std::ifstream inputFile(filename); // Open the text file for reading
00040
00041             if (!inputFile) { // Check if the file was opened successfully
00042                 std::cerr « "Unable to open file " « filename « std::endl;
00043                 return;
00044             }
00045
00046             // Read parameters from the file and set member variables
00047             std::string line;
00048             while (std::getline(inputFile, line)) {
00049
00050                 if (line.empty() || line[0] == '#' || line.substr(0, 2) == "//") {
00051                     continue;
00052                 }
00053
00054                 std::istringstream iss(line);
00055                 std::string paramName, equalsSign, paramValue;
00056
00057                 // Parse the line into parameter name, '=', and parameter value
00058                 if (iss » paramName » equalsSign » paramValue && equalsSign == "=") {
00059                     // Set member variables based on parameter name
00060                     if (paramName == "board_size") {
00061                         std::istringstream(paramValue) » board_size;
00062                     } else if (paramName == "N_critical") {
00063                         std::istringstream(paramValue) » N_critical;
00064                     } else if (paramName == "save_interval") {
00065                         std::istringstream(paramValue) » save_interval;
00066                     } else if (paramName == "num_evolve_steps") {
00067                         std::istringstream(paramValue) » evolve_steps;
00068                     } else if (paramName == "random_data") {
00069                         std::istringstream(paramValue) » random_data;
```

```
00070                     } else if (paramName == "prob_live") {
00071                         std::istringstream(paramValue) » prob_live;
00072                     } else if (paramName == "board_file") {
00073                         std::istringstream(paramValue) » board_file;
00074                     }else if (paramName == "output_path") {
00075                         std::istringstream(paramValue) » output_path;
00076                     } else if (paramName == "num_threads") {
00077                         std::istringstream(paramValue) » num_threads;
00078                     }
00079                 }
00080             }
00081
00082             // Close the file
00083             inputFile.close();
00084         }
00085
00087         void display() const {
00088             std::cout « "board size: " « board_size « std::endl;
00089             std::cout « "N_critical: " « N_critical « std::endl;
00090             std::cout « "save interval: " « save_interval « std::endl;
00091             std::cout « "evolve steps: " « evolve_steps « std::endl;
00092             std::cout « "num omp threads: " « num_threads « std::endl;
00093             std::cout « "probability to live: " « prob_live « std::endl;
00094             if (random_data) {
00095                 std::cout « "initialization: random" « std::endl;
00096             } else {
00097                 std::cout « "initialization: " « board_file « std::endl;
00098             }
00099         }
00100
00101 };
00102
00103 #endif
```

5.10 src/lib/Grid.hpp File Reference

```
#include <iostream>
#include <fstream>
#include <omp.h>
#include "Array1D.hpp"
```

Classes

- class Grid

  A class for a 2D grid that contains the entire board for the Game of Life.

5.11 Grid.hpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include <fstream>
00003 #include <omp.h>
00004 #include "Array1D.hpp"
00005
00006 #ifndef GRID_HPP
00007 #define GRID_HPP
00008
00010 class Grid{
00011     public:
00013         int N_row;
00015         int N_col;
00017         int* data;
00019         int N_nb_crit;
00021         int size;
00022
00024         Grid(int N_row, int N_col, int N_nb_crit = 3){
00025             this -> N_row = N_row;
```

```
00026             this -> N_col = N_col;
00027             this -> N_nb_crit = N_nb_crit;
00028             this -> data = new int[N_row*N_col];
00029             size = N_row * N_col;
00030         }
00032         ~Grid(){
00033             delete[] this -> data;
00034         }
00036         int& operator()(int i, int j){
00037             return this -> data[i*N_col+j];
00038         }
00043         void store_row(Array1D* store, int n_row, int shift = 0) {
00044             #pragma omp parallel for
00045             for (int i = 0; i < N_col; ++i) {
00046                 (*store)(i + shift) = data[n_row * N_col + i];
00047             }
00048         }
00052         void store_col(Array1D* store, int n_col) {
00053             #pragma omp parallel for
00054             for (int i = 0; i < N_row; ++i) {
00055                 (*store)(i) = data[i * N_col + n_col];
00056             }
00057         }
00058
00063         Array1D sub_row(int n_row, int i_low, int i_upp){
00064             Array1D temp(N_col);
00065             store_row(&temp, n_row);
00066             return temp.sub_arr(i_low, i_upp);
00067         }
00068
00073         Array1D sub_col(int n_col, int i_low, int i_upp){
00074             Array1D temp(N_row);
00075             store_col(&temp, n_col);
00076             return temp.sub_arr(i_low, i_upp);
00077         }
00078
00080         void display() {
00081             for (int i = 0; i < N_row; ++i) {
00082                 for (int j = 0; j < N_col; ++j) {
00083                     std::cout << data[i*N_col+j] << " ";
00084                 }
00085                 std::cout << std::endl;
00086             }
00087         }
00088
00091         Array1D periodic_row(int n_row){
00092             Array1D temp(N_col + 2);
00093             temp(0) = data[n_row * N_col + N_col - 1];
00094             store_row(&temp, n_row, 1);
00095             temp(N_col + 1) = data[n_row * N_col];
00096             return temp;
00097         }
00098
00101         void save(std::string file) {
00102             std::ofstream outputFile(file);
00103
00104             if (!outputFile.is_open()) {
00105                 std::cerr << "Error opening file for writing!" << std::endl;
00106             }
00107
00108             for (int i = 0; i < N_row; ++i) {
00109                 for (int j = 0; j < N_col-1; ++j) {
00110                     outputFile << data[i*N_col+j] << " ";
00111                 }
00112                 outputFile << data[i*N_col+N_col - 1];
00113                 outputFile << std::endl;
00114             }
00115
00116             outputFile.close();
00117         }
00118
00121         void store_data(int* arr){
00122             #pragma omp parallel for
00123             for (int i = 0; i < size; i++){
00124                 arr[i] = data[i];
00125             }
00126         }
00127
00130         void read_data(int* arr){
00131             #pragma omp parallel for
00132             for (int i = 0; i < size; i++){
00133                 data[i] = arr[i];
00134             }
00135         }
00136
00143         void overwrite_sub_board(int* arr, int row_low, int row_upp, int col_low, int col_upp){
00144             int n_rows = row_upp - row_low;
```

```
00145             int n_cols = col_upp - col_low;
00146             #pragma omp parallel for collapse(2)
00147                 for (int i = 0; i < n_rows; i++){
00148                     for (int j = 0; j < n_cols; j++){
00149                         data[(row_low + i)*N_col + col_low + j] = arr[i*n_cols + j];
00150                     }
00151                 }
00152         }
00153 };
00154
00155 #endif
```

5.12 src/main_parallel.cpp File Reference

```
#include <iostream>
#include <omp.h>
#include <mpi.h>
#include "lib/Board.hpp"
#include "lib/GameParams.hpp"
#include "lib/Functions.hpp"
#include "lib/Array1D.hpp"
#include "lib/Grid.hpp"
```

Functions

- int main (int argc, char ∗argv[ ])

5.12.1 Function Documentation

5.12.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

5.13 src/main_simple.cpp File Reference

```
#include <iostream>
#include <omp.h>
#include "lib/Board.hpp"
#include "lib/GameParams.hpp"
#include "lib/Functions.hpp"
#include "lib/Array1D.hpp"
#include "lib/Grid.hpp"
```

Functions

- int main (int argc, char ∗argv[ ])

5.13.1 Function Documentation

5.13.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Index