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:

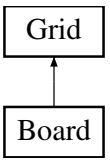
- src/lib/Array1D.hpp

4.2 Board Class Reference

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

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

Inheritance diagram for Board:



Public Member Functions

- Board (int N_row, int N_col)
- void init_from_motherboard (Grid *motherboard, int row_low, int row_upp, int col_left, int col_right)
- void set_bottom_ghost_row (Array1D *target)
- void set_upper_ghost_row (Array1D *target)
- void set_left_ghost_col (Array1D *target)
- void set_right_ghost_col (Array1D *target)
- void store_neighbour_row (Array1D *store, int n_row)
- void store_upper_ghost_neighbour_row (Array1D *store)
- void store_bottom_ghost_neighbour_row (Array1D *store)
- void ghost_display ()
- void update_board ()

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
- Array1D upper_ghost_row
- Array1D left_ghost_col
- Array1D right_ghost_col

  The right ghost column is not needed for the computation of the next generation.
- Array1D temp1
- 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 for a 2D grid that contains the entire board for the Game of Life.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Board()

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

4.2.3 Member Function Documentation

4.2.3.1 ghost_display()

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

4.2.3.2 init_from_motherboard()

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

4.2.3.3 set_bottom_ghost_row()

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

4.2.3.4 set_left_ghost_col()

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

4.2.3.5 set_right_ghost_col()

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

4.2.3.6 set_upper_ghost_row()

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

4.2.3.7 store_bottom_ghost_neighbour_row()

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

4.2.3.8 store_neighbour_row()

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

4.2.3.9 store_upper_ghost_neighbour_row()

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

4.2.3.10 update_board()

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

4.2.4 Member Data Documentation

4.2.4.1 bottom_ghost_row

```
Array1D Board::bottom_ghost_row
```

4.2.4.2 left_ghost_col

```
Array1D Board::left_ghost_col
```

4.2.4.3 right_ghost_col

```
Array1D Board::right_ghost_col
```

The right ghost column is not needed for the computation of the next generation.

4.2.4.4 temp1

```
Array1D Board::temp1
```

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 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 ()
- void readParams (const std::string &filename)
- void display () const

Public Attributes

- int board_size {10}
- int N_critical {3}
- int save_interval {1}
- int evolve_steps {20}
- int random_data {1}
- int num_threads {1}
- double prob_live {0.5}
- std::string board_file {"examples/"}
- std::string output_path {"examples/"}

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]
```

4.3.3 Member Function Documentation

4.3.3.1 display()

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

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/"}
```

4.3.4.2 board_size

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

4.3.4.3 evolve_steps

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

4.3.4.4 N_critical

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

4.3.4.5 num_threads

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

4.3.4.6 output_path

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

4.3.4.7 prob_live

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

4.3.4.8 random_data

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

4.3.4.9 save_interval

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

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

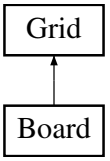
- src/lib/GameParams.hpp

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"
```

Classes

- class Board

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

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
00007 #ifndef BOARD_HPP
00008 #define BOARD_HPP
00009
00011 class Board : public Grid{
00012     public:
00013         /*The ghost rows include the corners and are therefore wider than the board*/
00014         Array1D bottom_ghost_row;
00015         Array1D upper_ghost_row;
00016         Array1D left_ghost_col;
00018         Array1D right_ghost_col;
00019
00020         Array1D temp1, temp2, temp3;
00021
00022         // Constructor and initialization of size of the arrays
00023         Board(int N_row, int N_col) : Grid(N_row, N_col),
00024         bottom_ghost_row(N_col+2), upper_ghost_row(N_col+2), left_ghost_col(N_row),
      right_ghost_col(N_row),
00025          temp1(N_col), temp2(N_col), temp3(N_col) {
00026             /*do something that demands N_row to be at least 3*/
00027         }
00028
00029         void init_from_motherboard(Grid* motherboard, int row_low, int row_upp, int col_left, int
      col_right){
00030             N_nb_crit = (*motherboard).N_nb_crit;
00031             #pragma omp parallel for collapse(2)
00032                 for (int i = 0; i < N_row; ++i) {
00033                     for (int j = 0; j < N_col; ++j) {
00034                         data[i*N_col+j] = (*motherboard)(row_low + i, col_left + j);
00035                     }
00036                 }
00037
00038         }
00039
00040         void set_bottom_ghost_row(Array1D* target) {
00041             for (int i = 0; i < N_col+2; ++i) {
00042                 bottom_ghost_row(i) = (*target)(i);
00043             }
00044         }
00045         void set_upper_ghost_row(Array1D* target) {
00046             for (int i = 0; i < N_col+2; ++i) {
00047                 upper_ghost_row(i) = (*target)(i);
00048             }
00049         }
00050
00051         void set_left_ghost_col(Array1D* target) {
00052             for (int i = 0; i < N_col; ++i) {
00053                 left_ghost_col(i) = (*target)(i);
00054             }
00055         }
00056         void set_right_ghost_col(Array1D* target) {
00057             for (int i = 0; i < N_col; ++i) {
00058                 right_ghost_col(i) = (*target)(i);
00059             }
00060         }
00061
00062         void store_neighbour_row(Array1D* store, int n_row) {
00063             (*store)(0) = left_ghost_col(n_row) + data[n_row * N_col + 0] + data[n_row * N_col + 1];
00064             for (int i = 1; i < N_col-1; ++i) {
00065                 (*store)(i) = data[n_row * N_col + i-1] + data[n_row * N_col + i]
00066                                 + data[n_row * N_col + i + 1];
00067             }
00068             (*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);
00069         }
00070
00071         void store_upper_ghost_neighbour_row(Array1D* store) {
00072             for (int i = 0; i < N_col; ++i) {
00073                 (*store)(i) = upper_ghost_row(i) + upper_ghost_row(i+1) + upper_ghost_row(i+2);
00074             }
00075         }
00076         void store_bottom_ghost_neighbour_row(Array1D* store) {
00077             for (int i = 0; i < N_col; ++i) {
00078                 (*store)(i) = bottom_ghost_row(i) + bottom_ghost_row(i+1) + bottom_ghost_row(i+2);
00079             }
00080         }
00081
```

```
00082         void ghost_display(){
00083             upper_ghost_row.display();
00084             for (int i = 0; i < N_row; ++i) {
00085                 std::cout « left_ghost_col(i) « " ";
00086                 for (int j = 0; j < N_col; ++j) {
00087                     std::cout « data[i*N_col+j] « " ";
00088                 }
00089                 std::cout « right_ghost_col(i) « std::endl;
00090             }
00091             bottom_ghost_row.display();
00092         }
00093 
00094         void update_board(){
00095             int N_nb {0};
00096             int val {0};
00097             store_upper_ghost_neighbour_row(&temp1);
00098             store_neighbour_row(&temp2, 0);
00099             store_neighbour_row(&temp3, 1);
00100             #pragma omp parallel for
00101                 for (int j = 0; j < N_col; ++j) {
00102                     val = data[j];
00103                     N_nb = temp1(j) + temp2(j) + temp3(j) - val;
00104                     data[j] = (1 - val) * (N_nb == N_nb_crit) + val * (N_nb == N_nb_crit || N_nb ==
    N_nb_crit - 1);
00105                 }
00106 
00107             for (int i = 1; i < N_row -1; ++i){
00108                 temp1.copy_into(&temp2);
00109                 temp2.copy_into(&temp3);
00110                 store_neighbour_row(&temp3, i+1);
00111                 #pragma omp parallel for
00112                     for (int j = 0; j < N_col; ++j) {
00113                         val = data[i*N_col + j];
00114                         N_nb = temp1(j) + temp2(j) + temp3(j) - val;
00115                         data[i*N_col + j] = (1 - val) * (N_nb == N_nb_crit) + val * (N_nb == N_nb_crit
    || N_nb == N_nb_crit - 1);
00116                     }
00117             }
00118             temp1.copy_into(&temp2);
00119             temp2.copy_into(&temp3);
00120             store_bottom_ghost_neighbour_row(&temp3);
00121             #pragma omp parallel for
00122                 for (int j = 0; j < N_col; ++j) {
00123                     val = data[(N_row - 1)*N_col + j];
00124                     N_nb = temp1(j) + temp2(j) + temp3(j) - val;
00125                     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);
00126                 }
00127         }
00128 };
00129 
00130 #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"
```

Functions

- void initialize_random (Grid *grid, GameParams *params)
- void initialize_from_file (Grid *grid, GameParams *params, std::string file)
- void iteration_one_board (Board *board, GameParams *params, Array1D *store_row, Array1D *store_col)

5.5.1 Function Documentation

5.5.1.1 initialize_from_file()

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

5.5.1.2 initialize_random()

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

5.5.1.3 iteration_one_board()

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

5.6 src/lib/Functions.hpp File Reference

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

Functions

- void initialize_random (Grid *grid, GameParams *params)
- void initialize_from_file (Grid *grid, GameParams *params, std::string file)
- void iteration_one_board (Board *board, GameParams *params, Array1D *store_row, Array1D *store_col)

5.6.1 Function Documentation

5.6.1.1 initialize_from_file()

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

5.6.1.2 initialize_random()

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

5.6.1.3 iteration_one_board()

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

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
00013         int board_size {10};
00014         int N_critical {3};
00015         int save_interval {1};
00016         int evolve_steps {20};
00017         int random_data {1};
00018         int num_threads {1};
00019         double prob_live {0.5};
00020         std::string board_file {"examples/"};
00021         std::string output_path {"examples/"};
00022
00023         GameParams(){}
00024
00027         void readParams(const std::string& filename){
00028
00029             std::ifstream inputFile(filename); // Open the text file for reading
00030
00031             if (!inputFile) { // Check if the file was opened successfully
00032                 std::cerr « "Unable to open file " « filename « std::endl;
00033                 return;
00034             }
00035
00036             // Read parameters from the file and set member variables
00037             std::string line;
00038             while (std::getline(inputFile, line)) {
00039
00040                 if (line.empty() || line[0] == '#' || line.substr(0, 2) == "//") {
00041                     continue;
00042                 }
00043
00044                 std::istringstream iss(line);
00045                 std::string paramName, equalsSign, paramValue;
00046
00047                 // Parse the line into parameter name, '=', and parameter value
00048                 if (iss » paramName » equalsSign » paramValue && equalsSign == "=") {
00049                     // Set member variables based on parameter name
00050                     if (paramName == "board_size") {
00051                         std::istringstream(paramValue) » board_size;
00052                     } else if (paramName == "N_critical") {
00053                         std::istringstream(paramValue) » N_critical;
00054                     } else if (paramName == "save_interval") {
00055                         std::istringstream(paramValue) » save_interval;
00056                     } else if (paramName == "num_evolve_steps") {
00057                         std::istringstream(paramValue) » evolve_steps;
00058                     } else if (paramName == "random_data") {
00059                         std::istringstream(paramValue) » random_data;
00060                     } else if (paramName == "prob_live") {
00061                         std::istringstream(paramValue) » prob_live;
00062                     } else if (paramName == "board_file") {
00063                         std::istringstream(paramValue) » board_file;
00064                     }else if (paramName == "output_path") {
00065                         std::istringstream(paramValue) » output_path;
00066                     } else if (paramName == "num_threads") {
00067                         std::istringstream(paramValue) » num_threads;
00068                     }
00069                 }
00070             }
00071
00072             // Close the file
00073             inputFile.close();
00074         }
00075
00076         void display() const {
00077             std::cout « "board size: " « board_size « std::endl;
00078             std::cout « "N_critical: " « N_critical « std::endl;
00079             std::cout « "save interval: " « save_interval « std::endl;
00080             std::cout « "evolve steps: " « evolve_steps « std::endl;
00081             std::cout « "num omp threads: " « num_threads « std::endl;
00082             std::cout « "probability to live: " « prob_live « std::endl;
00083             if (random_data) {
00084                 std::cout « "initialization: random" « std::endl;
00085             } else {
```

```
00086                 std::cout « "initialization: " « board_file « std::endl;
00087             }
00088         }
00089
00090 };
00091
00092 #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             for (int i = 0; i < N_col; ++i) {
00045                 (*store)(i + shift) = data[n_row * N_col + i];
00046             }
00047         }
00051         void store_col(Array1D* store, int n_col) {
00052             for (int i = 0; i < N_row; ++i) {
00053                 (*store)(i) = data[i * N_col + n_col];
00054             }
00055         }
00056
00061         Array1D sub_row(int n_row, int i_low, int i_upp){
00062             Array1D temp(N_col);
00063             store_row(&temp, n_row);
00064             return temp.sub_arr(i_low, i_upp);
```

```
00065         }
00066
00071         Array1D sub_col(int n_col, int i_low, int i_upp){
00072             Array1D temp(N_row);
00073             store_col(&temp, n_col);
00074             return temp.sub_arr(i_low, i_upp);
00075         }
00076
00078         void display() {
00079             for (int i = 0; i < N_row; ++i) {
00080                 for (int j = 0; j < N_col; ++j) {
00081                     std::cout « data[i*N_col+j] « " ";
00082                 }
00083                 std::cout « std::endl;
00084             }
00085         }
00086
00089         Array1D periodic_row(int n_row){
00090             Array1D temp(N_col + 2);
00091             temp(0) = data[n_row * N_col + N_col - 1];
00092             store_row(&temp, n_row, 1);
00093             temp(N_col + 1) = data[n_row * N_col];
00094             return temp;
00095         }
00096
00099         void save(std::string file) {
00100             std::ofstream outputFile(file);
00101
00102             if (!outputFile.is_open()) {
00103                 std::cerr « "Error opening file for writing!" « std::endl;
00104             }
00105
00106             for (int i = 0; i < N_row; ++i) {
00107                 for (int j = 0; j < N_col-1; ++j) {
00108                     outputFile « data[i*N_col+j] « " ";
00109                 }
00110                 outputFile « data[i*N_col+N_col - 1];
00111                 outputFile « std::endl;
00112             }
00113
00114             outputFile.close();
00115         }
00116
00119         void store_data(int* arr){
00120             for (int i = 0; i < size; i++){
00121                 arr[i] = data[i];
00122             }
00123         }
00124
00127         void read_data(int* arr){
00128             for (int i = 0; i < size; i++){
00129                 data[i] = arr[i];
00130             }
00131         }
00132
00139         void overwrite_sub_board(int* arr, int row_low, int row_upp, int col_low, int col_upp){
00140             int n_rows = row_upp - row_low;
00141             int n_cols = col_upp - col_low;
00142             #pragma omp parallel for collapse(2)
00143                 for (int i = 0; i < n_rows; i++){
00144                     for (int j = 0; j < n_cols; j++){
00145                         data[(row_low + i)*N_col + col_low + j] = arr[i*n_cols + j];
00146                     }
00147                 }
00148         }
00149 };
00150
00151 #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"
```

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 "lib/Board.hpp"
#include "lib/GameParams.hpp"
#include "lib/Functions.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