CellAutoCpp Document

Contents

Built-in Function

countSurroundingCellsWithValue

```
int countSurroundingCellsWithValue(const std::vector<Cell *> &neighbors,
const state_name &state)
```

description:

count number of neighboring cells has states value greater than or equal to 1

parameters:

neighbors: The neighbors of the cell.

state: the state to be counted.

return:

the number of neighboring cells has states value greater than or equal to 1.

getSurroundingCellsAverageValue

```
int getSurroundingCellsAverageValue(const std::vector<Cell *> &neighbors,
const state_name &state);
```

description:

get average value of neighboring cells

parameters:

neighbors: The neighbors of the cell.

state: the state to be counted.

return:

the average value of neighboring cells

get_neighbors

```
std::vector<Cell*> get_neighbors(const grid_type &grid, int x, int y);
```

description:

find the neighbors of the cell given by grid[x][y]

parameters:

grid: the grid the cell automata

x: x coordinate of the cell

y: y coordinate of the cell

return:

an 1d array of the neighbor cells.

Exception

nonexist_type

description

inherited from out_of_range exception. nonexist_type would be throwed when a cell is attempted to be set to be any type that does not exist.

combine_error

description

combine_error will appear when users attempt to specify a block of time stamp array to be combined, which exceed the size of time stamp array.

percentage_error

description

percentage_error will appear when the summation of generation probability in percentage of all cell types is not equal to 100.

CAWorld_param_error

description

CAWorld_param_error will appear when user attempt to build a cell automata of impractical grid size such as a size of negative width.

internal_error

description

interl_error will appear when the CAWorld engine comes across some errors.

Type

neighborindex

```
enum neighborindex
{
    TOPLEFT = 0,
    TOP = 1,
    TOPRIGHT = 2,
    LEFT = 3,
    RIGHT = 4,
    BOTTOMLEFT = 5,
    BOTTOM = 6,
    BOTTOMRIGHT = 7
};
```

description

Evolving depending on neighbors is prevalent. One can get the neighbors for a cell by calling the function get_neighbors, which returns a 1d array of cells. neighborindex indexing the cell array to the relevant position of neighbors.

world_param_type

```
typedef std::tuple<unsigned, unsigned, unsigned> world_param_type;
```

description

parameter for the size of cell automata gird. It is a tuple consisting of three unsigned int which is width, height and connection type of the grid respectively.

type_name

```
typedef std::string type_name;
```

description

name of the cell type

percentage

```
typedef unsigned percentage;
```

description

generation probability in percentage

gird_type

```
typedef std::vector<std::vector<Cell*>> grid_type;
```

description

2d array for grid. It stores pointers to cells in the grid.

frame_type

```
typedef std::vector<std::vector<Cell>> frame_type;
```

description

2d array for a frame of history.

reset_type

```
typedef std::function<void(Cell *)> reset_type;
```

description

type of the rule lambda for reset

init_type

```
typedef std::function<void(Cell *)> init_type;
```

description

type of the rule lambda for initialize

process_type

```
typedef std::function<void(grid_type &, Cell*)> process_type;
```

description

type of the rule lambda for process

getcolor_type

```
typedef std::function<int(Cell *)> getcolor_type;
```

description

type of the rule lambda for getcolor

gettypeind_type

```
typedef std::function<std::string(Cell *)> gettypeind_type;
```

description

type of the rule lambda for gettypeind

grid_param_type

```
typedef std::tuple<type_name, percentage, process_type, reset_type,
init_type> grid_param_type;
```

description

cell type tuple. A cell type tuple describes the name, generation probability and rule sets for initialize, reset and process of a cell type.

state_name

```
typedef std::string state_name;
```

description

name of state

state_value

```
typedef int state_value;
```

description

value of state

class model

Public function:

```
Model(world_param_type param, std::vector<grid_param_type> types = {},
unsigned size = 1, getcolor_type getcolor_func = nullptr);

void add_grid_type(grid_param_type type);

void add_grid_type(const type_name &name, percentage percent,
process_type process, reset_type reset, init_type init);
```

Private type:

```
typedef std::tuple<percentage, process_type, reset_type, init_type>
grid_param_type_no_name;
```

Private attributes:

```
world_param_type world_param;

std::unordered_map<type_name, grid_param_type_no_name> grid_types;
unsigned buffersize;

getcolor_type getcolor;
```

Private function:

```
inline void _add_grid_type(grid_param_type &type);
```

Description:

Model class packages parameters needed to describe a cell automata model. These parameters are classified into necessary parameters and optional parameters. Necessary parameters include the width and height of the grid and cell types list which contains type name and rule set for initialize, reset, reset for each cell type. The optional parameters include the length of the time stamp to be stored and rules for visualizing the automata. Model object serves as the argument for the constructor of the class CAWorld.

Documentation

constructor & destructor

```
Model(world_param_type param, std::vector<grid_param_type> types = {},
unsigned size = 1, getcolor_type getcolor_func = nullptr)
```

parameters:

param: size parameters of the gird. It is a world_param_type structs which contains the width and height and the connection type of the grid.

types: cell types list. It consists of gird_param_type tuple which specifies the type

name, the generated probability in percentage and rules for initialize, reset and process for each type of cell. Rules sets are implemented as lambdas. These rule lambdas must be defined as specific type: process_type, reset_type, init_type to make sure input and output format is what the CAWorld engine expects.
size: the size of the time stamp storage. The CAWorld engine supports storing the simulation history. How long the history would be stored is configurable. By setting the size to be 0, the engine is configured to store the whole history from the beginning of the simulation. By setting the size to be 1, the engine is configured to not store any history. And when the size to equal any number other than 0 and 1, the engine will maintain a fixed length of the history whose length is equal to size. Size is 1 by default.

getcolor_func: the rule for generating the color index of each cell for visualization. Like the rule for process and initialize, it is also implemented as lambda of type getcolor_type.

Member Function

```
void add_grid_type(grid_param_type type)
```
description

add a new cell type to the model

parameters

type: a cell type tuple that describes the type name, generated probability and rule set.

```
void add_grid_type(const type_name &name, percentage percent, process_type process, reset_type reset, init_type init)
```
description

add a new cell type to the model

parameters

name: name of the cell type
percent: generation probability
process: rule for process
reset: rule for reset
init: rule for initialize

```
inline void _add_grid_type(grid_param_type &type)
```
description

append the cell type list to the type list maintain the Model object.

parameters

type: cell type list of type tuples.

Class Cell

Public function

```
Cell()
Cell(const Cell &)

Cell(Cell &&) noexcept

Cell& operator=(const Cell &)

Cell& operator=(Cell &&) noexcept

virtual ~Cell()

inline state_value &operator[](const state_name &state)

inline void set_type(const type_name &rhs_type)

inline const type_name &get_type() const

inline const std::unordered_map<state_name, state_value> &
get_states() const
```

Public attributes:

```
int x,y;
```

Protected functions

```
static inline const type_name &_add_type(const std::pair<type_name,
Model::grid_param_type_no_name> &pair)

inline void _set_type(const type_name &rhs_type)

inline const process_type&_call_process() const

inline const reset_type&_call_reset() const

inline const init_type&_call_init() const

virtual void prepare_process()

virtual inline const unsigned timestamp_size() const

virtual inline void timestamp_resize(unsigned size)

virtual inline Cell get_frame(unsigned i) const

virtual inline Cell *_clone() const &

virtual inline Cell *_clone() &&

virtual inline void _move(Cell *cell)

virtual inline void _move(CellHistBounded *cell)

virtual inline void _move(CellHistUnbounded *cell)
```

Protected attributes

```
static std::unordered_map<type_name, std::tuple<process_type,
reset_type, init_type>> type_aux_funcs
```

```
std::unordered_map<state_name, state_value> states

type_name type
```

Description

Cell is the building block of cell automata. Cell automata maintains a certain number of cells, which are stored and organized as a grid. In each iteration of simulation, cell automata engine traverses the gird and applies the rule of process for each cell in the grid. The class Cell serves as the base class which is inherited by class CellHistBounded and class CellHistUnbounded. These three types of cell differ in the way to store the simulation history:

cell type	history storage	container for history
Cell	no history storage	
CellHistBounded	stores a limited length of history	std::deque
CellHistUnbounded	stores a unlimited length of history	std::vector

All cell objects share a table type_aux_funcs that maps the rule function for different types of cell.

Documentation

constructor & destructor

`Cell()`
description

default constructor

`Cell(const Cell &)`
description

copy constructor

`Cell(Cell &&) noexcept`
description

move constructor

`virtual ~Cell()`
description

destructor

member function

`Cell& operator=(const Cell &)`
description

assignment operation

`Cell& operator=(Cell &&) noexcept`
description

assignment operation

```
inline state_value &operator[](const state_name &state)
```
description

get the state value of the specified state

parameters

state: the lookup table

return

state value

```
inline void set_type(const type_name &rhs_type)
```
description

set the cell type

parameters

rhs_type: type name

```
inline const type_name &get_type() const
```
description

get the cell type

return

type name

```
inline const std::unordered_map<state_name, state_value> & get_states()
const
```
description

get the state dictionary of the cell

return

dictionary of the cell

```
static inline const type_name &_add_type(const std::pair<type_name,
Model::grid_param_type_no_name> &pair)
```
description

add the rule sets to the rule set function table type_aux_funcs

parameters

pair tuple that describes the rule set for a cell type

return

the name of cell type whose rule set is added to the rule set table.

```
inline void _set_type(const type_name &rhs_type)
```
description

set the type of the cell

parameters

rhs_type the name the cell type to be set

```
inline const process_type&_call_process() const
```
description

call the process function to apply the process rule to the cell. The process function is stored in the rule function table type_aux_funcs. The process function corresponding to the type of the cell is mapped in the running time.

```
inline const reset_type&_call_reset() const
```
description

call the reset function to apply the process rule to the cell. The reset function is stored in the rule function table type_aux_funcs. The reset function corresponding to the type of the cell is mapped in the running time.

```
inline const init_type&_call_init() const
```
description

call the init function to apply the initialize rule to the cell. The init function is stored in the rule function table type_aux_funcs. The init function corresponding to the type of the cell is mapped in the running time.

```
virtual void prepare_process()
```
description

store the current type and state into time stamp container as the simulation history. The prepare_process function do nothing in the class Cell which is the kind of cell storing no history.

```
virtual inline const unsigned timestamp_size() const
```
description

return the size of the time stamp, which is the number of history simulation step stored in the stamp. timestemp_size function for the class cell always return 0 because class Cell does not store simulation history

return

the size of time stamp container, which is always 0.

```
virtual inline void timestamp_resize(unsigned size)
```
description

change the size of the time stamp. Ang attempts to call the timestamp_resize for class cell will throw an exception since class Cell does not have any time stamp container.

```
virtual inline Cell get_frame(unsigned i) const
```
description

get the ith frame of history snapshot from the time stamp container. For class Cell, the frame index i must be 0, which means the current snapshot or an exception will be throwed because class Cell does not store history.

parameter

i index of frame

return

a snapshot of history type and state value of the cell.

```
virtual inline Cell *_clone() const &
```
description

copy function of class Cell

```
virtual inline Cell *_clone() &&
```

description

move copy function of class Cell

```
virtual inline void _move(Cell *cell)
virtual inline void _move(CellHistBounded *cell)
virtual inline void _move(CellHistUnbounded *cell)
```

description

move function of class Cell

Class CellHistBounded

Public function

```
CellHistBounded() = default;
CellHistBounded(unsigned buffersize)
CellHistBounded(const CellHistBounded &) = default;
CellHistBounded(CellHistBounded &&) noexcept = default;
CellHistBounded& operator=(const CellHistBounded &) = default;
CellHistBounded& operator=(CellHistBounded &&) noexcept = default;
virtual ~CellHistBounded() final = default;
```

Private function

```
virtual inline void prepare_process() final;
virtual inline const unsigned timestamp_size() const final;
virtual inline void timestamp_resize(unsigned size) final;
virtual inline Cell get_frame(unsigned i) const final;

virtual inline CellHistBounded *_clone() const & final;
virtual inline CellHistBounded *_clone() && final;
virtual inline void _move(Cell *cell) final;
virtual inline void _move(CellHistBounded *cell) final;
virtual inline void _move(CellHistUnbounded *cell) final;
```

private attributes

```
std::deque<type_name> type_hist;
std::deque<std::unordered_map<state_name, state_value>> states_hist;
```

Description

CellHistBounded is a kind of cell class that store a limited length of simulation history. It maintains a pair of queue, type_hist and states_hist that store the simulation history of type and state value respectively. It inherited from the class Cell and most of its function is set to be default, which will be omitted.

Documentation

```
virtual void prepare_process()
```

description

store the current type and state into time stamp container as the simulation history.

```
virtual inline const unsigned timestamp_size() const
```

description

return the size of the time stamp, which is the number of history simulation step stored in the stamp.

return

the size of time stamp container.

```
virtual inline void timestamp_resize(unsigned size)
```

description

change the size of the time stamp.

```
virtual inline Cell get_frame(unsigned i) const
```

description

get the ith frame of history snapshot from the time stamp container.

parameter

i index of frame

return

a snapshot of history type and state value of the cell.

Class CellHistUnbounded

Public function

```
CellHistUnbounded() = default;
CellHistUnbounded(const CellHistUnbounded &) = default;
CellHistUnbounded(CellHistUnbounded &&) noexcept = default;
CellHistUnbounded& operator=(const CellHistUnbounded &) = default;
CellHistUnbounded& operator=(CellHistUnbounded &&) noexcept =
default;
virtual ~CellHistUnbounded() final = default;
```

Private function

```
virtual inline void prepare_process() final;
virtual inline const unsigned timestamp_size() const final;
virtual inline void timestamp_resize(unsigned size) final;
virtual inline Cell get_frame(unsigned i) const final;

virtual inline CellHistUnbounded *_clone() const & final;
virtual inline CellHistUnbounded *_clone() && final;
virtual inline void _move(Cell *cell) final;
virtual inline void _move(CellHistBounded *cell) final;
virtual inline void _move(CellHistUnbounded *cell) final;
```

Private attributes

```
std::vector<type_name> type_hist;
std::vector<std::unordered_map<state_name, state_value>> states_hist;
```

Description

CellHistUnbounded is a kind of cell class that store the whole simulation history right from the beginning of the simulation. It maintains a pair of vector, type_hist and states_hist

that store the simulation history of type and state value respectively. It inherited from the class Cell and most of its function is set to be default, which will be omitted.

Documentation

```
virtual void prepare_process()
```

description

store the current type and state into time stamp container as the simulation history.

```
virtual inline const unsigned timestamp_size() const
```

description

return the size of the time stamp, which is the number of history simulation step stored in the stamp.

return

the size of time stamp container.

```
virtual inline void timestamp_resize(unsigned size)
```

description

change the size of the time stamp.

```
virtual inline Cell get_frame(unsigned i) const
```

description

get the ith frame of history snapshot from the time stamp container.

parameter

i index of frame

return

a snapshot of history type and state value of the cell.

Class CAWorld

Public function

```
CAWorld(const Model &model);
CAWorld(const CAWorld &rhs);
CAWorld(CAWorld &&rhs) noexcept;
CAWorld &operator=(const CAWorld &rhs);
CAWorld &operator=(CAWorld &&rhs) noexcept;
~CAWorld();

CAWorld &step(unsigned x, unsigned y);

CAWorld &forall_step(unsigned steps);

std::vector<int> print_world();

void save2file(const char * filename);

void loadfromfile(const char * filename);
```

```
std::vector<std::vector<std::string>> getgridref(gettypeind_type
gettypeind);

void initgridfromgridref(std::vector<std::vector<std::string>> &
gridref);

CAWorld &combine(const CAWorld &world, unsigned r_low, unsigned
r_high, unsigned c_low, unsigned c_high);

CAWorld &combine(CAWorld &&world, unsigned r_low, unsigned r_high,
unsigned c_low, unsigned c_high);

std::vector<frame_type> get_timestamps();

frame_type get_timestamp();

std::vector<CAMeasure*>& GetMeasures()

void AddMeasure(CAMeasure* n)

void AddMeasureAndRun(CAMeasure* n);
```

Private function

```
void combine_error_check(const CAWorld &world, unsigned r_low,
unsigned r_high, unsigned c_low, unsigned c_high)

const type_name _add_type(std::pair<type_name,
Model::grid_param_type_no_name> &pair)

type_name type_initializer(const std::vector<std::pair<type_name,
percentage>> &accum_dist)

void _forall_step()

void _step(unsigned x, unsigned y)

void copy_grid(const CAWorld &rhs)

void delete_grid()
```

Private attributes

```
unsigned width, height, grid_size;

grid_type grid;

bool empty_reset = 1;

unsigned buffersize;

getcolor_type getcolor;

std::vector<CAMeasure*> measures;
```

Description

CAWorld serves as the engine of the simulator. It maintains a 2d array of cells called grid. In each round of iteration, the engine looks up the rule set table according to cell type

for each cell and applies the founded rule set function for cells in the grid. The engine provides series of functionalities to monitor the evolving. The functionality includes measuring overall statistics, storing and print history snapshots and visualization. CAWorld should be initialized with model object, which conveys the parameters for the simulation such as the size of the grid, types of cell and their rule sets.

Documentation

Constructor & Destructor

```
CAWorld(const Model &model)
```

Description

initialize CAWrold object from model

parameters

model: model parameters

```
CAWorld(const CAWorld &rhs)
CAWorld(CAWorld &&rhs) noexcept
CAWorld &operator=(const CAWorld &rhs)
CAWorld &operator=(CAWorld &&rhs) noexcept;
~CAWorld()
```

Member function

```
CAWorld &step(unsigned x, unsigned y)
```

description

run a step of simulation for a single cell

parameters

x: x coordinates of the cell

y: y coordinates of the cell

return

reference to the CAWorld object

```
CAWorld &forall_step(unsigned steps)
```

description

run steps of simulation for all cells in the grid

parameters

steps: number of simulation step

return

reference to the CAWorld object

```
std::vector<int> print_world()
```

description

print color index of cells for visualization

return

color index array

```
void save2file(const char * filename)
```

description

save grid snapshot in file

parameters

filename: file path

```
void loadfromfile(const char * filename);
```

description

load grid snapshot from the file

parameters

filename: path of the file

```
std::vector<std::vector<std::string>> getgridref(gettypeind_type
gettypeind);
```

description

getgirdref export the current Cellular automaton model for building new cellular automaton model in the future. The function return an 2d array of index to cell types. Each element in the array corresponds to a cell in the grid. The function will call gettypeind to determine the type index for each cell. It depends on the user to interpret the index.

parameters

gettypeind: the rule set for generating the index.

return

the 2d array of indices.

```
void initgridfromgridref(std::vector<std::vector<std::string>> & gridref);
```

description

initialize the grid based on the index array generated by function getgridref().

parameters:

gridref: the 2d array of indices which can be generated by function getgridre().

```
CAWorld &combine(const CAWorld &world, unsigned r_low, unsigned r_high,
unsigned c_low, unsigned c_high);
```

description

combine the time stamp history of another world. The time stamp history will be copied and keep the another world to be combined inact. The position at which time stamp block to be placed is specified by the four parameters: r_low, r_high, c_low, c_high.

parameters

world: the world to be combined

r_low: the lower column bound for the position at which time stamp block to be placed.

r_hight: the higher column bound for the position at which time stamp block to be placed.

c_low: the lower row bound for the position at which time stamp block to be placed.

c_high: the higher row bound for the position at which time stamp block to be placed.

return

the reference to this world

```
CAWorld &combine(CAWorld &&world, unsigned r_low, unsigned r_high, unsigned c_low, unsigned c_high)
```

description

combine the time stamp history of another world. The time stamp history will be moved from the world to be combined. The position at which time stamp block to be placed is specified by the four parameters: r_low, r_high, c_low, c_high.

parameters

world: the world to be combined

r_low: the lower column bound for the position at which time stamp block to be placed.

r_hight: the higher column bound for the position at which time stamp block to be placed.

c_low: the lower row bound for the position at which time stamp block to be placed.

c_high: the higher row bound for the position at which time stamp block to be placed.

return

the reference to this world

```
std::vector<frame_type> get_timestamps()
```

description

get the time stamp stored in the world

return

the time stamp array

```
frame_type get_timestamp();
```

description

get a single frame of time stamp

return

a frame of the stamp array

```
std::vector<CAMeasure*>& GetMeasures()
```

description

get all CAMeasure objects that are currently attached to the CAWorld object

return

an array of CAMeassure pointers

```
void AddMeasure(CAMeasure* n)
```

description

attach a new CAMeasure object to the CAWorld object; from then on every cell's evolvement will update the measurement

parameters

n the measure object to be used

```
void AddMeasureAndRun(CAMeasure* n);
```

description

attach a new CAMeasure object to the CAWorld object and run it on the whole grid once

parameters

n the measure object to be used

CAMeasure

Public function

```
CAMeasure(const std::string &_name)

virtual void Init()

virtual void NewRecord()

virtual void Update(Cell *cell)

virtual std::string Str_Current()

virtual std::string Str_All()

std::string GetName()
```

Protected attributes

```
std::string name;
```

Description

This is the base class for all measurements. A measurement is a metric that is attached to any object of CAWorld class, monitors the status of the object from the moment attached and provides APIs for users to access recorded measurement data. A measurement is intended to keep one **record** for one moment (or step) of the world. A correct and desirable process of measurement consists of three steps: 1. Call function **`NewRecord()`** before the advance of the world; 2. Whenever a cell object in the grid is updated (and finalized), call the function **`Update()`** on that cell; 3. Retrieve the results by either accessing the data by specific APIs or call the stringify function **`Str_Current()`** or **`Str_All()`**. This pipeline is now fixed during the process of **`CAWorld::forall_step()`**.

Documentation

Constructor & Destructor

`CAMeasure(const std::string &_name)`

description

Construct a measurement object and give it a name

parameters

_name: the name of the measurement object

return

Return the constructed object

Member function

`virtual void Init()`

description

Initialize all statistics before start.

```
virtual void NewRecord()
```
description

Notify the measurement to start a new record for next moment, thus finalize the current result updated by current moment.

```
virtual void Update(Cell *cell)
```
description

Update the current record with states in a specific (and updated) cell.

parameters

cell: the cell used to update the record

```
virtual std::string Str_Current()
```
description

Get the stringified result of the current record.

Return

The stringified result of the current record

```
virtual std::string Str_All()
```
description

Get the stringified result of all records. (Each record is used to measure one moment of the world)

return

The stringified result of all records

```
std::string GetName()
```
description

Get the name of the measurement

return

The name of the measurement

CADistributionMeasure

Public function

```
CADistributionMeasure(std::string _name="State Value Distribution")

virtual void Init()

virtual void NewRecord()

virtual void Update(Cell *cell)

virtual std::string Str_Current()

virtual std::string Str_All()
```

Private function

```
std::string Str_Index(int i, std::string prefix="")
```

Private attributes

```
typedef std::unordered_map<type_name, std::unordered_map<state_name,
```

```
std::unordered_map<state_value, int> > > stats_frame_type;
std::vector<stats_frame_type> stats;
```

Description

This is a derived class from **`CAMeasure`**. This measurement is used to monitor the distribution of values in the world and works like a histogram. For every cell type and for every type of state of that type, it records the distribution of values in a hash map.

Documentation

```
virtual void Init()
```

description

Initialize all statistics before start. In this derived class, it clears the three-dimensional map.

```
virtual void NewRecord()
```

description

Notify the measurement to start a new record for next moment, thus finalize the current result updated by current moment. In this derived class, it pushes a new `stats_frame_type` type object to the three dimensional hash map.

```
virtual void Update(Cell *cell)
```

description

Update the current record with states in a specific (and updated) cell. In this derived class, it increments the histogram by looking at the type of cell and all its states.

parameters

cell: the cell used to update the record

```
virtual std::string Str_Current()
```

description

Get the stringified result of the current record.

Return

The stringified result of the current record

```
virtual std::string Str_All()
```

description

Get the stringified result of all records. (Each record is used to measure one moment of the world)

return

The stringified result of all records

```
std::string Str_Index(int i, std::string prefix="")
```

description

Get the stringified result of a specific record.

return

The stringified result of that record