Game Of Life

Reference Manual

Benjamin Grothe

1 Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

4 Namespace Documentation

4.1 gol Namespace Reference

Data Structures

- class Cell
- class Figure
- class GameOfLife
- class Grid
- class Neighbours
- struct Rule

  structure for Game of Life Rules.
- struct RuleColor

  structure for Game of Life Color Rules
- class UI
- struct Vec2

  structure defines a 2 dimensional Vector

Typedefs

- typedef std::list< Figure * > Figure_Stack

  includes Figures and is used for step back
  Stack size is defined by STACK_SIZE
- typedef unsigned int UI_Flag

  UI_Flags includes information if UI Windows are activated or not.
- typedef unsigned int UI_Options_Flag

  UI_Options_Flag includes some informations about some UI options.

Enumerations

- enum UI_Flag_ {
  UI_Flag_none = 0 , UI_Flag_enable_GridUI = 1<<0 , UI_Flag_enable_SetupUI = 1<<1 , UI_Flag_enable_LoadUI = 1<<2 ,
  UI_Flag_enable_LoadExampleUI = 1<<3 , UI_Flag_enable_SaveUI = 1<<4 , UI_Flag_enable_MessageUI = 1<<5 }

  UI_Flag_ defines Flags for UI_Flag typedef;.
- enum UI_Options_Flag_ {
  UI_Options_Flag_none = 0 , UI_Options_Flag_Run = 1<<0 , UI_Options_Flag_enable_AutoZoom = 1<<2 , UI_Options_Flag_enable_AutoScroll = 1<<3 ,
  UI_Options_Flag_enable_ColorizeAll = 1<<4 }

  UI_Options_Flag_ defines Flags for UI_Options_Flag typedef.

4.1.1 Typedef Documentation

4.1.1.1 Figure_Stack `typedef std::list<Figure*> gol::Figure_Stack`

includes Figures and is used for step back
Stack size is defined by STACK_SIZE

Definition at line 54 of file game_of_life.h.

4.1.1.2 UI_Flag `typedef unsigned int gol::UI_Flag`

UI_Flags includes information if UI Windows are activated or not.

Note

Flag values are defined in enumeration UI_Flag_

Definition at line 59 of file game_of_life.h.

4.1.1.3 UI_Options_Flag `typedef unsigned int gol::UI_Options_Flag`

UI_Options_Flag includes some informations about some UI options.

Note

Flag values are defined in enumeration UI_Options_Flag_

Definition at line 77 of file game_of_life.h.

4.1.2 Enumeration Type Documentation

4.1.2.1 UI_Flag_ `enum gol::UI_Flag_`

UI_Flag_ defines Flags for UI_Flag typedef;.

Enumerator

UI_Flag_none	
UI_Flag_enable_GridUI	
UI_Flag_enable_SetupUI	
UI_Flag_enable_LoadUI	
UI_Flag_enable_LoadExampleUI	
UI_Flag_enable_SaveUI	
UI_Flag_enable_MessageUI	

Definition at line 63 of file game_of_life.h.

```
00063                 {
00064         UI_Flag_none = 0,                       // no UI_Flag is set
00065         UI_Flag_enable_GridUI = 1«0,           // UI_Flag which is used for enabling Grid_UI
00066         UI_Flag_enable_SetupUI = 1«1,          // UI_Flag which is used for enabling Setting_UI
00067         UI_Flag_enable_LoadUI = 1«2,           // UI_Flag which is used for enabling Load_UI
00068         UI_Flag_enable_LoadExampleUI = 1«3,    // UI_Flag which is used for enabling Load_Exp_UI
00069         UI_Flag_enable_SaveUI = 1«4,           // UI_Flag which is used for enabling Save_UI
00070         UI_Flag_enable_MessageUI = 1«5         // UI_Flag which is used for enabling Message_UI
00071     };
```

4.1.2.2 UI_Options_Flag_ enum gol::UI_Options_Flag_

UI_Options_Flag_ defines Flags for UI_Options_Flag typedef.

Enumerator

UI_Options_Flag_none	
UI_Options_Flag_Run	
UI_Options_Flag_enable_AutoZoom	
UI_Options_Flag_enable_AutoScroll	
UI_Options_Flag_enable_ColorizeAll	

Definition at line 79 of file game_of_life.h.

```
00079                         {
00080         UI_Options_Flag_none = 0,                  // no UI_Options_Flag is set
00081         UI_Options_Flag_Run = 1«0,                // UI_Options_Flag which shows if Game of life is
      running
00082         UI_Options_Flag_enable_AutoZoom = 1«2,     // UI_Options_Flag which enabling auto zoom
00083         UI_Options_Flag_enable_AutoScroll = 1«3,   // UI_Options_Flag which enabling auto scroll
00084         UI_Options_Flag_enable_ColorizeAll = 1«4   // UI_Options_Flag which shows if all alive cells
      have the same color
00085     };
```

5 Data Structure Documentation

5.1 gol::Cell Class Reference

`#include "game_of_life.h"`

Public Member Functions

- Cell (Grid *_grid, Vec2 _position)

  Constructor for object cell.
- ~Cell ()=default

  Destructor for object cell.
- Vec2 get_position () const

  getter function for private member position
- bool get_state () const

  getter function for private member state
- unsigned int get_color () const

  getter function for private member color

- Grid * get_grid () const

  getter function for private member grid
- Neighbours * get_neighbours () const

  getter function for private member neighbours
- void set_state (bool _state)

  setter function for private member state
- void set_color (unsigned int _color)

  setter function for private member color
- void set_neighbours (Neighbours *_neighbours)

  setter function for private member neighbours

Private Attributes

- Vec2 position
- bool state
- unsigned int color
- Grid * grid
- Neighbours * neighbours

5.1.1 Detailed Description

Definition at line 105 of file game_of_life.h.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Cell() `gol::Cell::Cell (`

```
Grid * _grid,
Vec2 _position )
```

Constructor for object cell.

Parameters

_grid	: gol::Grid -> owner grid
_position	: gol::Vec2 -> position in grid

Definition at line 14 of file game_of_life.cpp.

```
00014                                       {
00015         grid = _grid;
00016         position = _position;
00017         state = false;              // initialize Cell object as death
00018         color = IM_COL32_WHITE;     // initialize Cell color as white
00019     }
```

5.1.2.2 ~Cell() `gol::Cell::~Cell ( ) [default]`

Destructor for object cell.

5.1.3 Member Function Documentation

5.1.3.1 get_color() `unsigned int gol::Cell::get_color ( ) const [inline]`

getter function for private member color

Returns

color of cell as const unsigned int

Definition at line 138 of file game_of_life.h.
```
00138 {return color; };
```

5.1.3.2 get_grid() `Grid * gol::Cell::get_grid ( ) const [inline]`

getter function for private member grid

Returns

owner grid as gol::Grid*

Definition at line 143 of file game_of_life.h.
```
00143 { return grid; }
```

5.1.3.3 get_neighbours() `Neighbours * gol::Cell::get_neighbours ( ) const [inline]`

getter function for private member neighbours

Returns

neighbour as gol::Neighbours*

Definition at line 148 of file game_of_life.h.
```
00148 { return neighbours; }
```

5.1.3.4 get_position() `Vec2 gol::Cell::get_position ( ) const [inline]`

getter function for private member position

Returns

position of cell in grid as const gol::Vec2

Definition at line 128 of file game_of_life.h.
```
00128 { return position; }
```

5.1.3.5 get_state() `bool gol::Cell::get_state ( ) const [inline]`

getter function for private member state

Returns

shows if cell is alive or dead as const bool

Definition at line 133 of file game_of_life.h.

```
00133 { return state; }
```

5.1.3.6 set_color() `void gol::Cell::set_color ( unsigned int _color ) [inline]`

setter function for private member color

Parameters

_color	: unsigned int -> set new cell color

Definition at line 160 of file game_of_life.h.

```
00160 { color = _color; }
```

5.1.3.7 set_neighbours() `void gol::Cell::set_neighbours ( Neighbours * _neighbours ) [inline]`

setter function for private member neighbours

Parameters

_neighbours	: gol::Neighbours* -> set new Neighbours Object which includes all neighbour cells

Definition at line 165 of file game_of_life.h.

```
00165 { neighbours = _neighbours; }
```

5.1.3.8 set_state() `void gol::Cell::set_state ( bool _state ) [inline]`

setter function for private member state

Parameters

_state	: bool -> new state [true = alive, false = dead]

Definition at line 155 of file game_of_life.h.

```
00155 { state = _state; }
```

5.1.4 Field Documentation

5.1.4.1 color `unsigned int gol::Cell::color [private]`

Definition at line 109 of file game_of_life.h.

5.1.4.2 grid `Grid* gol::Cell::grid [private]`

Definition at line 110 of file game_of_life.h.

5.1.4.3 neighbours `Neighbours* gol::Cell::neighbours [private]`

Definition at line 111 of file game_of_life.h.

5.1.4.4 position `Vec2 gol::Cell::position [private]`

Definition at line 107 of file game_of_life.h.

5.1.4.5 state `bool gol::Cell::state [private]`

Definition at line 108 of file game_of_life.h.

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

- game_of_life.h
- game_of_life.cpp

5.2 gol::Figure Class Reference

```
#include "game_of_life.h"
```

Public Member Functions

- Figure (std::vector< bool > _states, Vec2 &_dimensions)
  explict Constructor for Figure object
- Figure (std::string &str, Vec2 &_dimensions)
  explict Constructor for Figure object
- Figure (std::string &path)
  constructor which loads object figure from .lif file
- Figure (GameOfLife *gameOfLife)
  constructor to save Game Of Lifes grid into a figure
- ~Figure ()=default
  Desturctor for object Figure.
- Vec2 get_dimension () const
  getter function for private member dimension
- bool get_state (Vec2 _position) const
  getter function for explicit state in private member states
- int save_to_file (std::string &path)
  save all states of this object into a .lif file

Private Member Functions

- std::vector< bool > convert_string_to_states (std::string str)
  convert a string in .lif format to an boolean vector
- std::string convert_dimension_to_string ()
  convert the real figure dimensions to string
- std::string convert_states_to_string ()
  convert states to a single string in .lif format

Static Private Member Functions

- static Vec2 convert_string_to_dimensions (std::string str)
  filter dimensions with format like "x=100 , y=150" from std::string

Private Attributes

- Vec2 dimensions
- std::vector< bool > states

5.2.1 Detailed Description

Definition at line 276 of file game_of_life.h.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Figure() [1/4]

```
gol::Figure::Figure (
        std::vector< bool > _states,
        Vec2 & _dimensions ) [inline], [explicit]
```

explict Constructor for Figure object

Parameters

_states	: std::vector<bool> -> include all new Figure cell states
_dimensions	: gol::Vec2 -> includes dimension of new Figure object

Definition at line 308 of file game_of_life.h.

```
00308 {dimensions = _dimensions; states = std::move(_states);  }
```

5.2.2.2 Figure() [2/4] gol::Figure::Figure (std::string & *str*, Vec2 & *_dimensions*) [inline], [explicit]

explict Constructor for Figure object

Parameters

str	: std::string& -> string in .lif format which will be converted into private member states
_dimensions	: gol::Vec2 -> includes dimension of new Figure object

Definition at line 314 of file game_of_life.h.

```
00314 { dimensions = _dimensions; states = convert_string_to_states(str); }
```

5.2.2.3 Figure() [3/4] gol::Figure::Figure (std::string & *path*) [explicit]

constructor which loads object figure from .lif file

Parameters

path	: std::string -> full path to .lif file

Definition at line 163 of file game_of_life.cpp.

```
00163                                       {
00164         std::fstream file;
00165
00166         std::string code;   // whole code as string
00167         std::string dim;    // dimension as string
00168
00169
00170         file.open(path);
00171             // check if file exist else throw std::logic_error
00172         if(!file.is_open()){ throw std::logic_error("ERROR: could not load file"); }
00173
00174         while(file){
00175
00176             std::string row;
00177             // read line in file and write line into row
00178             std::getline(file, row);
00179             // if line starts with a number, 'b', 'o' , '$' or '!' write line into code
00180             // if line starts with '#' ignore line
00181             // else write line into dim 'dimension'
00182             if(((row[0] >= 0x30) && (row[0] <= 0x39)) || (row[0] == 'b') || (row[0] == 'o') || (row[0]
      == '$') || ((row[0] == '!'))){
00183                 for(auto c: row){
00184                     code.push_back(c);
```

```
00185                 }
00186             }else if(row[0] != '#'){
00187                 for(auto c : row){
00188                     dim.push_back(c);
00189                 }
00190             }
00191         }
00192 
00193         // get dimension of figure from dim
00194         dimensions = convert_string_to_dimensions(dim);
00195         // fill figure_table with converted code
00196         states = convert_string_to_states(code);
00197 
00198 
00199     }
```

5.2.2.4 Figure() [4/4] gol::Figure::Figure (GameOfLife * *gameOfLife*) [explicit]

constructor to save Game Of Lifes grid into a figure

Parameters

gameOfLife	: gol::GameOfLife

Definition at line 200 of file game_of_life.cpp.

```
00200                                           {
00201         // get dimensions from GameOfLife object
00202         dimensions = gameOfLife->get_grid()->get_dimension();
00203         // get every cell state in grid
00204         for(auto c: gameOfLife->get_grid()->get_cells()){
00205             states.push_back(c->get_state());
00206         }
00207     }
```

5.2.2.5 ~Figure() gol::Figure::~Figure ( ) [default]

Desturctor for object Figure.

5.2.3 Member Function Documentation

5.2.3.1 convert_dimension_to_string() std::string gol::Figure::convert_dimension_to_string ( ) [private]

convert the real figure dimensions to string

Definition at line 228 of file game_of_life.cpp.

```
00228                                                   {
00229         std::string str_dim;
00230 
00231         int x = 0;
00232         int y = 0;
00233 
00234         // check real row of figure
00235         int t_y = 0;
00236         for(int _y = 0; _y < dimensions.y; _y++){
```

```
00237             int t_x = 0;
00238             for(int _x = 0; _x < dimensions.y; _x++){
00239                 if(states[(_y*dimensions.x)+_x]){t_x = _x;}
00240             }
00241 
00242             if(x < t_x){ x = t_x; }
00243             if(t_x > 0){ y = _y; }
00244             if((t_y == 0) && (_y > t_y) && (t_x > 0)){
00245                 t_y = _y;
00246             }
00247         }
00248 
00249         int column_f = 0;
00250         bool b = false;
00251         for(int _x=0; _x < dimensions.x; _x++){
00252             for(int _y=0; _y < dimensions.y; _y++){
00253                 if(states[(_y*dimensions.x+_x)]){ b = true; }
00254             }
00255             column_f = _x;
00256             if(b){break;}
00257 
00258         }
00259         // convert real dimensions into a string
00260         str_dim += "x=" + std::to_string(x-column_f+1) + " , y=" + std::to_string(y-t_y+2);
00261 
00262 
00263         return str_dim;
00264     }
```

5.2.3.2 convert_states_to_string() std::string gol::Figure::convert_states_to_string ( ) [private]

convert states to a single string in .lif format

Definition at line 315 of file game_of_life.cpp.

```
00315                                                 {
00316         std::string str_states;
00317 
00318         // convert states to string
00319         for(int y=0; y < dimensions.y; y++){
00320             for(int x=0; x < dimensions.x; x++){
00321                 if(states[(y*dimensions.x)+x]){ str_states += 'o'; }else{ str_states += 'b'; }
00322             }
00323             str_states += '$';
00324         }
00325         // count first empty columns
00326         int column_f = 0;
00327         bool b = false;
00328         for(int _x=0; _x < dimensions.x; _x++){
00329             for(int _y=0; _y < dimensions.y; _y++){
00330                 if(states[(_y*dimensions.x+_x)]){ b = true; }
00331             }
00332             column_f = _x;
00333             if(b){break;}
00334 
00335         }
00336         // filter first columns
00337         b = true;
00338         int counter = 0;
00339         std::string _str_states;
00340         for(auto c: str_states){
00341             if(c == '$'){ _str_states.push_back(c); b = true;}else
00342             if(b == true){
00343                 counter++;
00344                 if(counter >= column_f){ b = false; counter = 0;}
00345             }else
00346             { _str_states.push_back(c);}
00347 
00348         }
00349 
00350         str_states = _str_states;
00351 
00352         // filter last death columns
00353         counter = 0;
00354         char last = 0x00;
00355         _str_states.clear();
00356         for(auto c: str_states){
00357             if(last == 0x00){last = c;}
00358             if(c == last){ counter++;}
00359             if(c != last){
```

```
00360                 if((c == '$') && (last == 'b')){ counter = 1; last = 0x00; }
00361                 if(counter > 1){ _str_states += std::to_string(counter) + last; counter = 1; } else
    {_str_states += last;}
00362                 last = c;
00363             }
00364         }
00365
00366         int counter_f = 0; // counter first empty rows
00367         int counter_l = 0; // counter last empty rows
00368         b = false;    // first row marker
00369         for(auto c: _str_states){
00370             if((c=='$')&&(b == false)){ counter_f+=2; } //count first empty rows
00371             if((c=='$')&&(b == true)){ counter_l+=2; }  //count last empty rows
00372             if((c == 'o') || (c == 'b')){counter_l = 0; b = 1; }
00373         }
00374         // delete first empty rows
00375         _str_states.erase(0,counter_f);
00376
00377         // delete last empty rows
00378         while(counter_l > 0){
00379             _str_states.pop_back();
00380             counter_l--;
00381         }
00382         _str_states += '!';
00383
00384         str_states.clear();
00385         for(auto c: _str_states){
00386             if(c != 0x00){str_states.push_back(c);}
00387         }
00388         return str_states;
00389     }
```

5.2.3.3 convert_string_to_dimensions()

`Vec2 gol::Figure::convert_string_to_dimensions ( std::string str ) [static], [private]`

filter dimensions with format like "x=100 , y=150" from std::string

Parameters

str	: std::string

Returns

dimensions as gol::Vec2

Definition at line 209 of file game_of_life.cpp.

```
00209                                                                 {
00210         std::string x, y;
00211
00212         int comma = 0;
00213
00214         for(auto c: str){
00215
00216             // filter numbers
00217             if((c >= 0x30) && (c <= 0x39)){
00218                 if(comma == 0){ x.push_back(c); }
00219                 if(comma == 1){ y.push_back(c); }
00220             }
00221             // filter comma
00222             if(c == ','){ comma++; }
00223             if(comma > 1){ throw std::logic_error("ERROR: there are to much comma's in dimension
    line"); }
00224         }
00225
00226         return (Vec2){std::stoi(x), std::stoi(y)};
00227     }
```

5.2.3.4 convert_string_to_states() std::vector< bool > gol::Figure::convert_string_to_states (std::string *str*) [private]

convert a string in .lif format to an boolean vector

Note

used to load figure from file

Parameters

str	: std::string

Returns

std::vector<bool>

Definition at line 266 of file game_of_life.cpp.

```
00266                                                                   {
00267         std::vector<bool> _states;
00268         std::string _num;
00269         int x = 0;
00270         int y = 0;
00271
00272         // prepare _states for the right dimensions
00273         for(int i=0; i < (dimensions.x * dimensions.y); i++){
00274             _states.push_back(false);
00275         }
00276
00277         // check each character in string
00278         for(auto c: str){
00279             // if character is a number add to _num buffer
00280             if((c >=0x30) && (c <= 0x39)){ _num.push_back(c); }
00281             // if character is a 'b' set state on position [x;y](*_num) to false and increment x
00282             if(c == 'b'){
00283                 if(_num.empty()){ _num.push_back('1'); }
00284                 for(int b=0; b < std::stoi(_num); b++){
00285                     _states[(y *dimensions.x)+(x)]=false;
00286                     x++;
00287                 }
00288                 // reset _num buffer
00289                 _num.clear();
00290             }
00291             // if character is a 'o' set state on position [x;y](*_num) to true and increment x
00292             if(c == 'o'){
00293                 if(_num.empty()){ _num.push_back('1'); }
00294                 for(int o=0; o < std::stoi(_num); o++){
00295                     _states[(y*dimensions.x)+(x)]=true;
00296                     x++;
00297                 }
00298                 // reset _num buffer
00299                 _num.clear();
00300             }
00301             // if character is a '$' increment y; reset _num buffer; set x to 0
00302             if(c == '$'){
00303                 if(_num.empty()){ _num.push_back('1');}
00304                 for(int r=0; r < std::stoi(_num); r++){
00305                     y++;
00306                 }
00307                 _num.clear();
00308                 x=0;
00309             }
00310             // EOF "End Of File"
00311             if(c == '!'){ break; }
00312         }
00313         return _states;
00314     }
```

5.2.3.5 get_dimension() `Vec2 gol::Figure::get_dimension ( ) const [inline]`

getter function for private member dimension

Returns

dimension of figure as gol::Vec2

Definition at line 332 of file game_of_life.h.

```
00332 { return dimensions; }
```

5.2.3.6 get_state() `bool gol::Figure::get_state ( Vec2 _position ) const [inline]`

getter function for explicit state in private member states

Parameters

_position	: gol::Vec2 position of state in

Returns

Definition at line 339 of file game_of_life.h.

```
00339 { return states[(_position.y*dimensions.x)+_position.x]; }
```

5.2.3.7 save_to_file() `int gol::Figure::save_to_file ( std::string & path )`

save all states of this object into a .lif file

Parameters

path	: std::string -> full filepath of new .lif file

Returns

for Error handling

Definition at line 391 of file game_of_life.cpp.

```
00391                                           {
00392         std::fstream file;
00393
00394         // create file and open
00395         file.open(path , std::ios_base::out);
00396         // Error Handling
00397         if(!file.is_open()){ throw std::logic_error("ERROR: failed"); }
00398
00399         file « convert_dimension_to_string() « "\n";
```

```
00400         file « convert_states_to_string();
00401
00402         // close file
00403         file.close();
00404
00405         return 0;
00406     }
```

5.2.4 Field Documentation

5.2.4.1 dimensions `Vec2 gol::Figure::dimensions [private]`

Definition at line 278 of file game_of_life.h.

5.2.4.2 states `std::vector<bool> gol::Figure::states [private]`

Definition at line 279 of file game_of_life.h.

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

- game_of_life.h
- game_of_life.cpp

5.3 gol::GameOfLife Class Reference

`#include "game_of_life.h"`

Public Member Functions

- GameOfLife (Vec2 _dimensions, Rule _rules, RuleColor _rulesColor)
  constructor of GameOfLife object
- ~GameOfLife ()=default
  Destructor of GameOfLife object.
- Grid * get_grid ()
- Rule * get_rules ()
- RuleColor * get_color_rules ()
- void set_rules (Rule rule)
- int run ()
  MASTER-function to run game of life with ImGui interface.
- void populate_figure (Figure *_figure, Vec2 position, int angle)
  populate grid in this object with informations from figure object
- void populate_random ()
  populate grid object with random cell states
- void refresh_grid ()
  check all cells grid and refresh grid object
- void clear_grid ()
  kill all cells in grid object

Private Member Functions

- bool check_rules_in_cell (Cell *_cell)

  check neighbours in Cell and return new state for cell
- void check_rules_in_grid ()

  check rules for every Cell in Grid and refresh Grid after checking rules

Static Private Member Functions

- static bool random_bool ()

Private Attributes

- Grid * grid
- Rule rules
- RuleColor rules_color

5.3.1 Detailed Description

Definition at line 354 of file game_of_life.h.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 GameOfLife() `gol::GameOfLife::GameOfLife (`

```
Vec2 _dimensions,
Rule _rules,
RuleColor _rulesColor )
```

constructor of GameOfLife object

Parameters

_dimensions	: gol::Vec2 -> dimensions of grid in this object
_rules	: gol::Rules

Definition at line 408 of file game_of_life.cpp.

```
00408                                                                    {
00409         rules = _rules;                                        // initialize rules
00410         grid = new Grid(this, _dimensions); // initialize new grid
00411         rules_color = _rulesColor;                             // initialize color Rules
00412     }
```

5.3.2.2 ~GameOfLife() `gol::GameOfLife::~GameOfLife ( ) [default]`

Destructor of GameOfLife object.

5.3.3 Member Function Documentation

5.3.3.1 check_rules_in_cell() bool gol::GameOfLife::check_rules_in_cell (Cell * _cell) [private]

check neighbours in Cell and return new state for cell

Parameters

_cell	: Cell*

Returns

new state as bool;

Definition at line 414 of file game_of_life.cpp.

```
00414                                                   {
00415         int counter = 0;
00416         // count every true cell in neighbours
00417         for(auto n: _cell->get_neighbours()->get_cells()){
00418             if(n->get_state()){ counter++; }
00419         }
00420         // check rules
00421         if (rules.alive[counter]){ return true; }
00422         if (rules.death[counter]){ return false; }
00423
00424
00425
00426         // returns new cell state
00427         return _cell->get_state();
00428     }
```

5.3.3.2 check_rules_in_grid() void gol::GameOfLife::check_rules_in_grid ( ) [private]

check rules for every Cell in Grid and refresh Grid after checking rules

Definition at line 429 of file game_of_life.cpp.

```
00429                                           {
00430         std::vector<bool> _states;  // buffer for cell states
00431
00432         // check rules for every cell in grid and push result into state buffer
00433         for(auto c: grid->get_cells()){
00434             _states.push_back(check_rules_in_cell(c));
00435         }
00436         // write new state from buffer to cells in grid
00437         for(auto c: grid->get_cells()){
00438             c->set_state(_states[(c->get_position().y*grid->get_dimension().x)+c->get_position().x]);
00439         }
00440
00441         // change color
00442         for(auto c: grid->get_cells()){
00443             c->get_neighbours()->set_n_alive(0);
00444             for(auto n: c->get_neighbours()->get_cells()){
00445                 if(n->get_state()){
      c->get_neighbours()->set_n_alive(c->get_neighbours()->get_n_alive()+1); }
00446             }
00447             c->set_color(rules_color.color[c->get_neighbours()->get_n_alive()]);
00448         }
00449
00450     }
```

5.3.3.3 clear_grid() void gol::GameOfLife::clear_grid ( )

kill all cells in grid object

Definition at line 452 of file game_of_life.cpp.

```
00452                                {
00453         // set every cell state in grit to false
00454         for(auto c: grid->get_cells()){
00455             c->set_state(false);
00456         }
00457     }
```

5.3.3.4 get_color_rules() RuleColor * gol::GameOfLife::get_color_rules ( ) [inline]

Definition at line 386 of file game_of_life.h.

```
00386 { return &rules_color; }
```

5.3.3.5 get_grid() Grid * gol::GameOfLife::get_grid ( ) [inline]

Definition at line 384 of file game_of_life.h.

```
00384 { return grid; }
```

5.3.3.6 get_rules() Rule * gol::GameOfLife::get_rules ( ) [inline]

Definition at line 385 of file game_of_life.h.

```
00385 { return &rules; }
```

5.3.3.7 populate_figure() void gol::GameOfLife::populate_figure (

```
Figure * _figure,
Vec2 position,
int angle )
```

populate grid in this object with informations from figure object

Parameters

_figure	: gol::Figure
position	: gol::Vec2 -> position of figure in grid
angle	: Int -> 0=[0°];1=[90°];2=[180°];3=[270°]

Definition at line 458 of file game_of_life.cpp.

```
00458                                                                                {
00459         int x_c = 0;
00460
00461         // Error Handler
00462         if((angle < 0)||(angle > 3)){throw std::logic_error("ERROR wrong angle in
      rotate_and_populate_figure()"); }
00463         // no roation
```

```
00464         if(angle == 0){
00465             // Error handling
00466             if(_figure->get_dimension().x+position.x > grid->get_dimension().x){throw
      std::logic_error("ERROR: figure dimension x is larger than grid dimension x");}
00467             if(_figure->get_dimension().y+position.y > grid->get_dimension().y){throw
      std::logic_error("ERROR: figure dimension y is larger than grid dimension y");}
00468             // insert figure states into grid
00469             for(int _y=0; _y < _figure->get_dimension().y; _y++){
00470                 for(int _x=0; _x < _figure->get_dimension().x; _x++){
00471                     grid->get_cell((Vec2){_x + position.x, _y +
      position.y})->set_state(_figure->get_state((Vec2){_x, _y}));
00472                 }
00473             }
00474         }
00475 
00476         // rotate 180°
00477         if(angle == 2){
00478             // Error handling
00479             if(_figure->get_dimension().x > grid->get_dimension().x){throw std::logic_error("ERROR:
      figure dimension x is larger than grid dimension x");}
00480             if(_figure->get_dimension().y > grid->get_dimension().y){throw std::logic_error("ERROR:
      figure dimension y is larger than grid dimension y");}
00481             // insert figure states into grid
00482             for(int _y=_figure->get_dimension().y-1; _y >= 0; _y--){
00483                 int y_c = 0;
00484                 for(int _x=_figure->get_dimension().x-1; _x >= 0; _x--){
00485                     grid->get_cell((Vec2){y_c + position.x, x_c +
      position.y})->set_state(_figure->get_state((Vec2){_x, _y}));
00486                     y_c++;
00487                 }
00488                 x_c++;
00489             }
00490         }
00491 
00492         // rotate 90°
00493         if(angle == 1){
00494             // Error Handling
00495             if(_figure->get_dimension().x > grid->get_dimension().y){throw std::logic_error("ERROR:
      figure dimension x is larger than grid dimension y");}
00496             if(_figure->get_dimension().y > grid->get_dimension().x){throw std::logic_error("ERROR:
      figure dimension y is larger than grid dimension x");}
00497             // insert figure states into grid
00498             for(int _x=0; _x < _figure->get_dimension().x; _x++){
00499                 int y_c = 0;
00500                 for(int _y=_figure->get_dimension().y-1; _y >= 0; _y--){
00501                     grid->get_cell((Vec2){y_c + position.x, x_c +
      position.y})->set_state(_figure->get_state((Vec2){_x,_y}));
00502                     y_c++;
00503                 }
00504                 x_c++;
00505             }
00506         }
00507 
00508         // rotate 270°
00509         if(angle == 3){
00510             // Error Handling
00511             if(_figure->get_dimension().x > grid->get_dimension().y){throw std::logic_error("ERROR:
      figure dimension x is larger than grid dimension y");}
00512             if(_figure->get_dimension().y > grid->get_dimension().x){throw std::logic_error("ERROR:
      figure dimension y is larger than grid dimension x");}
00513             // insert figure states into grid
00514             for(int _x=_figure->get_dimension().x-1; _x >= 0; _x--){
00515                 int y_c = 0;
00516                 for(int _y=0; _y < _figure->get_dimension().y; _y++){
00517                     grid->get_cell((Vec2){y_c + position.x, x_c +
      position.y})->set_state(_figure->get_state((Vec2){_x, _y}));
00518                     y_c++;
00519                 }
00520                 x_c++;
00521             }
00522         }
00523     }
```

5.3.3.8 populate_random() `void gol::GameOfLife::populate_random ( )`

populate grid object with random cell states

Definition at line 529 of file game_of_life.cpp.

```
00529                                    {
```

```
00530         for(int _y=0; _y < grid->get_dimension().y; _y++){
00531             for(int _x=0; _x < grid->get_dimension().x; _x++){
00532                 grid->get_cell((Vec2){_x, _y})->set_state(random_bool());
00533             }
00534         }
00535         // Colorize Grid
00536         for(auto c: grid->get_cells()){
00537             c->get_neighbours()->set_n_alive(0);
00538             for(auto n: c->get_neighbours()->get_cells()){
00539                 if(n->get_state()){
      c->get_neighbours()->set_n_alive(c->get_neighbours()->get_n_alive()+1); }
00540             }
00541             c->set_color(rules_color.color[c->get_neighbours()->get_n_alive()]);
00542         }
00543     }
```

5.3.3.9 random_bool() `bool gol::GameOfLife::random_bool ( )  [static], [private]`

Returns

random boolean value (50:50)

Definition at line 525 of file game_of_life.cpp.

```
00525                                 {
00526             if (rand() % 2 == 0){return true;}
00527             return false;
00528     }
```

5.3.3.10 refresh_grid() `void gol::GameOfLife::refresh_grid ( )  [inline]`

check all cells grid and refresh grid object

Definition at line 405 of file game_of_life.h.

```
00405 { check_rules_in_grid(); } // FIXME: its to slow ( ca  « 9(x*y)++ » loops on 1 core ) <- Optimazion
      flag -O1 !! <- eventually save only alive cells in grid calculate neighbours and change them "Stack
      style"
```

5.3.3.11 run() `int gol::GameOfLife::run ( )`

MASTER-function to run game of life with ImGui interface.

Returns

for error handling

Definition at line 544 of file game_of_life.cpp.

```
00544                         {
00545
00546
00547         // Setup SDL // https://github.com/ocornut/imgui.git
00548         if (SDL_Init(SDL_INIT_VIDEO | SDL_INIT_TIMER | SDL_INIT_GAMECONTROLLER) != 0)
00549         {
00550             printf("Error: %s\n", SDL_GetError());
00551             return -1;
00552         }
00553
00554
00555         // From 2.0.18: Enable native IME. // https://github.com/ocornut/imgui.git
00556         #ifdef SDL_HINT_IME_SHOW_UI
```

```
00557         SDL_SetHint(SDL_HINT_IME_SHOW_UI, "1");
00558         #endif
00559
00560
00561         // Setup window // https://github.com/ocornut/imgui.git
00562         SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);
00563         SDL_GL_SetAttribute(SDL_GL_DEPTH_SIZE, 24);
00564         SDL_GL_SetAttribute(SDL_GL_STENCIL_SIZE, 8);
00565         SDL_GL_SetAttribute(SDL_GL_CONTEXT_MAJOR_VERSION, 2);
00566         SDL_GL_SetAttribute(SDL_GL_CONTEXT_MINOR_VERSION, 2);
00567         SDL_WindowFlags window_flags = (SDL_WindowFlags)(SDL_WINDOW_OPENGL | SDL_WINDOW_RESIZABLE |
      SDL_WINDOW_ALLOW_HIGHDPI);
00568         SDL_Window* window = SDL_CreateWindow("Cornway's Game Of Life", SDL_WINDOWPOS_CENTERED,
      SDL_WINDOWPOS_CENTERED, 1280, 720, window_flags);
00569         SDL_GLContext gl_context = SDL_GL_CreateContext(window);
00570
00571         SDL_GL_MakeCurrent(window, gl_context);
00572         SDL_GL_SetSwapInterval(1); // Enable vsync
00573
00574         // Setup Dear ImGui context // https://github.com/ocornut/imgui.git
00575         IMGUI_CHECKVERSION();
00576         ImGui::CreateContext();
00577
00578         ImGuiIO& io = ImGui::GetIO(); (void)io;
00579         io.ConfigFlags |= ImGuiConfigFlags_NavEnableKeyboard;     // Enable Keyboard Controls
00580         io.ConfigFlags |= ImGuiConfigFlags_NavEnableGamepad;      // Enable Gamepad Controls
00581         io.ConfigFlags |= ImGuiConfigFlags_DockingEnable;         // Enable Docking
00582         io.ConfigFlags |= ImGuiConfigFlags_ViewportsEnable;       // Enable Multi-Viewport / Platform
      Windows
00583         //io.ConfigViewportsNoAutoMerge = true;
00584         //io.ConfigViewportsNoTaskBarIcon = true;
00585
00586         // Setup Dear ImGui style
00587         ImGui::StyleColorsDark();
00588         //ImGui::StyleColorsLight();
00589
00590         // When viewports are enabled we tweak WindowRounding/WindowBg so platform windows can look
      identical to regular ones. // https://github.com/ocornut/imgui.git
00591         ImGuiStyle& style = ImGui::GetStyle();
00592         if (io.ConfigFlags & ImGuiConfigFlags_ViewportsEnable)
00593         {
00594             style.WindowRounding = 0.0f;
00595             style.Colors[ImGuiCol_WindowBg].w = 1.0f;
00596             style.ChildBorderSize = 0.0f;
00597         }
00598
00599         // Setup Platform/Renderer backends // https://github.com/ocornut/imgui.git
00600         ImGui_ImplSDL2_InitForOpenGL(window, gl_context);
00601         ImGui_ImplOpenGL2_Init();
00602
00603         ImVec4 clear_color = ImVec4(0.0f, 0.0f, 0.0f, 1.00f);
00604         // Main loop
00605         bool done = false;
00606         // ------------------------------------------------------------------------------
00607         // USER CODE GLOBAL BEGIN -------------------------------------------------------
00608         UI_Flag uiFlag = UI_Flag_none;
00609         uiFlag |= UI_Flag_enable_GridUI | UI_Flag_enable_SetupUI;
00610
00611         UI_Options_Flag uiOptFlag = UI_Options_Flag_none;
00612
00613         std::string message;                                    // message for Message User Interface
00614
00615         float zoom = 2.f;                                           // zoom variable
00616         int speed = 0, count=0;                                 // speed variables
00617
00618         Figure* reset_figure = new Figure(this);     // Reset figure object
00619         Figure_Stack stack = {};                                // Stack needed for step back -> max
      size defined ba STACK_SIZE
00620
00621         std::string p;                                          // filepath as string
00622
00623         // USER CODE GLOBAL END ---------------------------------------------------------
00624         // ------------------------------------------------------------------------------
00625         // main loop begin
00626         while (!done) {
00627
00628             // Poll and handle events (inputs, window resize, etc.) //
      https://github.com/ocornut/imgui.git
00629             // You can read the io.WantCaptureMouse, io.WantCaptureKeyboard flags to tell if dear
      imgui wants to use your inputs.
00630             // - When io.WantCaptureMouse is true, do not dispatch mouse input data to your main
      application, or clear/overwrite your copy of the mouse data.
00631             // - When io.WantCaptureKeyboard is true, do not dispatch keyboard input data to your main
      application, or clear/overwrite your copy of the keyboard data.
00632             // Generally you may always pass all inputs to dear imgui, and hide them from your
      application based on those two flags.
00633             SDL_Event event;
```

```
00634             while (SDL_PollEvent(&event))
00635             {
00636                 ImGui_ImplSDL2_ProcessEvent(&event);
00637                 if (event.type == SDL_QUIT)
00638                     done = true;
00639                 if (event.type == SDL_WINDOWEVENT && event.window.event == SDL_WINDOWEVENT_CLOSE &&
      event.window.windowID == SDL_GetWindowID(window))
00640                     done = true;
00641             }
00642 
00643             ImGui_ImplOpenGL2_NewFrame();
00644             ImGui_ImplSDL2_NewFrame();
00645             ImGui::NewFrame();
00646 
      //-----------------------------------------------------------------------------------------------------------------
00647             // USER CODE FRAME BEGIN
      ------------------------------------------------------------------------------
00648             // initialize Grid User Interface if Grid_UI is true
00649             UI::Dockspace_UI();
00650 
00651             if(uiFlag & UI_Flag_enable_GridUI) {
00652 
00653                 UI::grid_UI(/*io,*/ uiFlag, uiOptFlag, *this, stack, zoom, speed, count);
00654             }
00655             // initialize Setting User Interface if Setting_UI is true
00656             if(uiFlag & UI_Flag_enable_SetupUI) {
00657                 UI::setting_UI(uiFlag, uiOptFlag, *this, *reset_figure, stack, zoom, speed);
00658             }
00659             // initialize Load User Interface if Load_UI is true
00660             if(uiFlag & UI_Flag_enable_LoadUI){
00661                 UI::load_UI(uiFlag, uiOptFlag, *this, *reset_figure, message, zoom); // TODO remove
      zoom input variable
00662             }
00663             // initialize Message User Interface if Message_UI is true
00664             if(uiFlag & UI_Flag_enable_MessageUI){
00665                 UI::messageBox_UI(uiFlag, message);
00666             }
00667             // initialize Screenshot User Interface if Save_UI is true
00668             if(uiFlag & UI_Flag_enable_SaveUI){
00669                 UI::screenshot_UI(uiFlag ,*this, message );
00670             }
00671             if(uiFlag & UI_Flag_enable_LoadExampleUI) {
00672                 UI::load_example_UI(uiFlag, uiOptFlag, *this, *reset_figure, zoom, speed);
00673             }
00674             // USER CODE FRAME END
      ------------------------------------------------------------------------------
00675 
      //-----------------------------------------------------------------------------------------------------------------
00676 
00677 
00678             // Rendering // https://github.com/ocornut/imgui.git
00679             ImGui::Render();
00680             glViewport(0, 0, (int)io.DisplaySize.x, (int)io.DisplaySize.y);
00681             glClearColor(clear_color.x * clear_color.w, clear_color.y * clear_color.w, clear_color.z *
      clear_color.w, clear_color.w);
00682             glClear(GL_COLOR_BUFFER_BIT);
00683             //glUseProgram(0); // You may want this if using this code in an OpenGL 3+ context where
      shaders may be bound
00684             ImGui_ImplOpenGL2_RenderDrawData(ImGui::GetDrawData());
00685 
00686             // Update and Render additional Platform Windows // https://github.com/ocornut/imgui.git
00687             // (Platform functions may change the current OpenGL context, so we save/restore it to
      make it easier to paste this code elsewhere.
00688             //  For this specific demo app we could also call SDL_GL_MakeCurrent(window, gl_context)
      directly)
00689             if (io.ConfigFlags & ImGuiConfigFlags_ViewportsEnable)
00690             {
00691                 SDL_Window* backup_current_window = SDL_GL_GetCurrentWindow();
00692                 SDL_GLContext backup_current_context = SDL_GL_GetCurrentContext();
00693                 ImGui::UpdatePlatformWindows();
00694                 ImGui::RenderPlatformWindowsDefault();
00695                 SDL_GL_MakeCurrent(backup_current_window, backup_current_context);
00696             }
00697 
00698             SDL_GL_SwapWindow(window);
00699         }
00700 
00701 
00702 
00703         // Cleanup // https://github.com/ocornut/imgui.git
00704         ImGui_ImplOpenGL2_Shutdown();
00705         ImGui_ImplSDL2_Shutdown();
00706         ImGui::DestroyContext();
00707         //SDL_GLContext gl_context = SDL_GL_CreateContext(window);
00708         SDL_GL_DeleteContext(gl_context);
00709         SDL_DestroyWindow(window);
00710         SDL_Quit(); //
```

```
00711
00712         return 0;
00713
00714     }
```

5.3.3.12 set_rules() void gol::GameOfLife::set_rules (Rule *rule*) [inline]

Definition at line 388 of file game_of_life.h.

```
00388 { rules = rule; }
```

5.3.4 Field Documentation

5.3.4.1 grid Grid* gol::GameOfLife::grid [private]

Definition at line 356 of file game_of_life.h.

5.3.4.2 rules Rule gol::GameOfLife::rules [private]

Definition at line 357 of file game_of_life.h.

5.3.4.3 rules_color RuleColor gol::GameOfLife::rules_color [private]

Definition at line 358 of file game_of_life.h.

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

- game_of_life.h
- game_of_life.cpp

5.4 gol::Grid Class Reference

```
#include "game_of_life.h"
```

Public Member Functions

- Grid (GameOfLife *_gameOfLife, Vec2 _dimension)
  constuctor for object grid
- ~Grid ()=default
  Desturctor for object grid.
- Vec2 get_dimension () const
- std::vector< Cell * > get_cells ()
  return all cells in this grid
- Cell * get_cell (Vec2 position)
  return explicit cell
- Vec2 get_real_grid_dimension ()
  calculate dimension of Grid with alive cells,
- float get_scrollx_position ()
  calculate x-axis scroll position which will be used in autoscroll mode by Grid_UI
- float get_scrolly_position ()
  calculate y-axis scroll position which will be used in autoscroll mode by Grid_UI
- float get_auto_zoom_factor ()
  calculate autozoom factor which will be used by Grid_UI

Private Member Functions

- Vec2 get_real_grid_position ()
  calculate the position of left upper corner of real_grid_dimension "Figure"

Private Attributes

- Vec2 dimension
- std::vector< Cell * > cells
- GameOfLife * gameOfLife

5.4.1 Detailed Description

Definition at line 173 of file game_of_life.h.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 Grid()

```
gol::Grid::Grid (
        GameOfLife * _gameOfLife,
        Vec2 _dimension )
```

constuctor for object grid

Parameters

_gameOfLife	: gol::GameOfLife -> owner GameOfLife
_dimension	: gol::Vec2 -> dimensions of this grid

Definition at line 21 of file game_of_life.cpp.

```
00021                                                   {
00022         gameOfLife = _gameOfLife;
00023         dimension = _dimension;
00024 
00025         // initialize Grid object with x * y death Cell objects
00026         for(int y = 0; y < dimension.y; y++){
00027             for(int x = 0; x < dimension.x; x++){
00028                 cells.push_back(new Cell(this, (Vec2){x,y}));
00029             }
00030         }
00031 
00032         // create for each Cell object in Grid object a Neighbours object
00033         for(int y = 0; y < dimension.y; y++){
00034             for(int x = 0; x < dimension.x; x++){
00035                 new Neighbours(cells[(y*dimension.x)+x]);
00036             }
00037         }
00038     }
```

5.4.2.2 ~Grid() gol::Grid::~Grid ( ) [default]

Desturctor for object grid.

5.4.3 Member Function Documentation

5.4.3.1 get_auto_zoom_factor() float gol::Grid::get_auto_zoom_factor ( )

calculate autozoom factor which will be used by Grid_UI

Returns

float

Definition at line 80 of file game_of_life.cpp.

```
00080                                   {
00081         float d = 1;
00082         if(get_real_grid_dimension().x > get_real_grid_dimension().y){ d = (float)dimension.x /
      (float)get_real_grid_dimension().x;}else{
00083             d = (float)dimension.y/(float)get_real_grid_dimension().y;
00084         }
00085         if(d <= 2.f){ return 2.f; }
00086         return d;
00087     }
```

5.4.3.2 get_cell() Cell * gol::Grid::get_cell (Vec2 position) [inline]

return explicit cell

Parameters

position	: gol::Vec2 -> position of cell in grid

Returns

single cell on position as gol::Cell

Definition at line 206 of file game_of_life.h.

```
00206 { return cells[((position.y* this->dimension.x)+position.x)]; }
```

5.4.3.3 get_cells() std::vector< Cell * > gol::Grid::get_cells ( ) [inline]

return all cells in this grid

Returns

all cells in grid as std::vector<gol::Cell*>

Definition at line 200 of file game_of_life.h.

```
00200 { return cells;}
```

5.4.3.4 get_dimension() Vec2 gol::Grid::get_dimension ( ) const [inline]

Definition at line 195 of file game_of_life.h.

```
00195 { return dimension; }
```

5.4.3.5 get_real_grid_dimension() Vec2 gol::Grid::get_real_grid_dimension ( )

calculate dimension of Grid with alive cells,

the borders (up,down,left,right) with dead cells will be cut

Returns

gol::Vec2

Definition at line 40 of file game_of_life.cpp.

```
00040                                       {
00041         int first_row  = 0; bool first_row_flag = false;
00042         int last_row   = 0;
00043
00044         // filter first and last row
00045         for(auto c: cells){
00046             if(c->get_state()){
00047                 if(!first_row_flag){
00048                     first_row_flag = true;
00049                     first_row = c->get_position().y;
00050                     last_row = c->get_position().y;
00051                 }else{
00052                     last_row = c->get_position().y;
00053                 }
00054             }
00055         }
00056
00057         int first_column  = 0; bool first_column_flag = false;
00058         int last_column   = 0;
00059         // filter first and last column
00060         for(int x=0; x < dimension.x; x++){
00061             for(int y=0; y < dimension.y; y++){
```

```
00062                 if(cells[(y*dimension.x)+x]->get_state()){
00063                     if(!first_column_flag){
00064                         first_column_flag = true;
00065                         first_column = cells[(y*dimension.x)+x]->get_position().x;
00066                         last_column = cells[(y*dimension.x)+x]->get_position().x;
00067                     }else{
00068                         last_column = cells[(y*dimension.x)+x]->get_position().x;
00069                     }
00070                 }
00071             }
00072         }
00073 
00074         int y = last_row-first_row+1;
00075         int x = last_column-first_column+1;
00076 
00077 
00078         return Vec2{x,y};
00079     }
```

5.4.3.6 get_real_grid_position() Vec2 gol::Grid::get_real_grid_position ( ) [private]

calculate the position of left upper corner of real_grid_dimension "Figure"

Returns

gol::Vec2

Definition at line 88 of file game_of_life.cpp.

```
00088                                       {
00089         //TODO get center of real grid dimension
00090 
00091         //TODO first living cell in row
00092         Vec2 position_A = Vec2{0,0};
00093         Vec2 position = Vec2{0,0};
00094         for(auto c: cells){
00095             if(c->get_state()) {
00096                 position_A.y = c->get_position().y;
00097                 break;
00098             }
00099 
00100         }
00101 
00102         // TODO first living cell in column
00103         bool b = false;
00104         for(int x=0; x < dimension.x; x++){
00105             for(int y=0; y < dimension.y; y++){
00106                 if(cells[(y*dimension.x)+x]->get_state()) {
00107                     position_A.x = cells[(y*dimension.x)+x]->get_position().x;
00108                     b = true;
00109                     break;
00110                 }
00111             }
00112             if(b){break;}
00113         }
00114         position.x = position_A.x;// + (get_real_grid_dimension().x/2);
00115         position.y = position_A.y;// + (get_real_grid_dimension().y/2);
00116 
00117         return position; // Center of figure
00118     }
```

5.4.3.7 get_scrollx_position() float gol::Grid::get_scrollx_position ( )

calculate x-axis scroll position which will be used in autoscroll mode by Grid_UI

Returns

float

Definition at line 119 of file game_of_life.cpp.

```
00119                                       {
00120         float x = (float)get_real_grid_position().x;//+((float)get_real_grid_dimension().x/2);
00121         auto x_max = (float)dimension.x;
00122         float real_x = ((float)get_real_grid_dimension().x+2)*x/x_max;
00123         return (x+real_x)/x_max;
00124     }
```

5.4.3.8 get_scrolly_position() `float gol::Grid::get_scrolly_position ( )`

calculate y-axis scroll position which will be used in autoscroll mode by Grid_UI

Returns

float

Definition at line 125 of file game_of_life.cpp.

```
00125                                   {
00126
00127         float y = (float)get_real_grid_position().y;//+((float)get_real_grid_dimension().y/2);
00128         auto y_max = (float)dimension.y;
00129         float real_y = ((float)get_real_grid_dimension().y+1)*y/y_max;
00130         return (y+real_y)/y_max;
00131     }
```

5.4.4 Field Documentation

5.4.4.1 cells `std::vector<Cell*> gol::Grid::cells [private]`

Definition at line 176 of file game_of_life.h.

5.4.4.2 dimension `Vec2 gol::Grid::dimension [private]`

Definition at line 175 of file game_of_life.h.

5.4.4.3 gameOfLife `GameOfLife* gol::Grid::gameOfLife [private]`

Definition at line 177 of file game_of_life.h.

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

- game_of_life.h
- game_of_life.cpp

5.5 gol::Neighbours Class Reference

```
#include "game_of_life.h"
```

Public Member Functions

- Neighbours (Cell ∗_owner)

  constructor of object Neighbours
- ∼Neighbours ()=default

  Desturctor for object Neighbours.
- std::vector< Cell ∗ > get_cells () const

  getter function for all neighbour cells from owner
- int get_n_alive () const

  getter function for private member n_alive
- void set_n_alive (int _n)

  setter function for private member n_alive

Private Attributes

- Cell ∗ owner
- std::vector< Cell ∗ > cells
- int n_alive

5.5.1 Detailed Description

Definition at line 237 of file game_of_life.h.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 Neighbours() `gol::Neighbours::Neighbours ( Cell * _owner ) [explicit]`

constructor of object Neighbours

Parameters

_owner	: gol::Cell -> cell which owns this object

Definition at line 134 of file game_of_life.cpp.

```
00134                                       {
00135         owner = _owner;
00136
00137         owner->set_neighbours(this);        // associate this neighbour to owner Cell
00138
00139         Vec2 pos_o = owner->get_position(); // position owner
00140         Grid* grid = owner->get_grid();     // grid
00141
00142         // add all cells next to this object
00143         for(int y= -1; y <= 1; y++){
00144             for(int x= -1; x <= 1; x++){
00145
00146                 // position of next_to cell
00147                 Vec2 pos_n = (Vec2){pos_o.x+x,pos_o.y + y};
00148                 // check x position of next_to cell
00149                 if( (pos_n.x >= 0) && (pos_n.x < owner->get_grid()->get_dimension().x) ){
00150                     // check y position of next_to cell
```

```
00151                     if( (pos_n.y >= 0) && (pos_n.y < owner->get_grid()->get_dimension().y) ){
00152                         // filter owner cell object
00153                         if((pos_o.x != pos_n.x) || (pos_o.y != pos_n.y)){
00154                             cells.push_back(grid->get_cell(pos_n));
00155                         }
00156                     }
00157                 }
00158             }
00159         }
00160
00161     }
```

5.5.2.2 ~Neighbours() gol::Neighbours::~Neighbours ( ) [default]

Desturctor for object Neighbours.

5.5.3 Member Function Documentation

5.5.3.1 get_cells() std::vector< Cell * > gol::Neighbours::get_cells ( ) const [inline]

getter function for all neighbour cells from owner

Returns

all neighbour cells from owner as std::vector<Cell*>

Definition at line 257 of file game_of_life.h.

```
00257 { return cells; }
```

5.5.3.2 get_n_alive() int gol::Neighbours::get_n_alive ( ) const [inline]

getter function for private member n_alive

Returns

number of all alive cells as int

Definition at line 263 of file game_of_life.h.

```
00263 { return n_alive; }
```

5.5.3.3 set_n_alive() void gol::Neighbours::set_n_alive (int _n) [inline]

setter function for private member n_alive

Parameters

↩ _↩ *n*	: int -> new value of all alive cells

Definition at line 268 of file game_of_life.h.

```
00268 { n_alive = _n; }
```

5.5.4 Field Documentation

5.5.4.1 cells `std::vector<Cell*> gol::Neighbours::cells [private]`

Definition at line 240 of file game_of_life.h.

5.5.4.2 n_alive `int gol::Neighbours::n_alive [private]`

Definition at line 242 of file game_of_life.h.

5.5.4.3 owner `Cell* gol::Neighbours::owner [private]`

Definition at line 239 of file game_of_life.h.

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

- game_of_life.h
- game_of_life.cpp

5.6 gol::Rule Struct Reference

structure for Game of Life Rules.

```
#include "game_of_life.h"
```

Data Fields

- bool alive [9]
- bool death [9]

5.6.1 Detailed Description

structure for Game of Life Rules.

shows what to do if n neighbour cells are alive or dead

Definition at line 93 of file game_of_life.h.

5.6.2 Field Documentation

5.6.2.1 alive `bool gol::Rule::alive[9]`

Definition at line 93 of file game_of_life.h.

5.6.2.2 death `bool gol::Rule::death[9]`

Definition at line 93 of file game_of_life.h.

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

- game_of_life.h

5.7 gol::RuleColor Struct Reference

structure for Game of Life Color Rules

```
#include "game_of_life.h"
```

Data Fields

- unsigned int color [9]

5.7.1 Detailed Description

structure for Game of Life Color Rules

shows which cell color should be used if n neighbour cells are alive

Definition at line 98 of file game_of_life.h.

5.7.2 Field Documentation

5.7.2.1 color `unsigned int gol::RuleColor::color[9]`

Definition at line 98 of file game_of_life.h.

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

- game_of_life.h

5.8 gol::UI Class Reference

`#include "game_of_life.h"`

Static Public Member Functions

- static void grid_UI (UI_Flag &uiFlag, UI_Options_Flag &uiOptionsFlag, GameOfLife &gameOfLife, Figure_Stack &figureStack, float &zoom, int &speed, int &count)

  initialize Grid user interface
- static void setting_UI (UI_Flag &uiFlag, UI_Options_Flag &uiOptionsFlag, GameOfLife &gameOfLife, Figure &reset_figure, Figure_Stack &figureStack, float &zoom, int &speed)

  initialize Setting user interface
- static void load_UI (UI_Flag &uiFlag, UI_Options_Flag &uiOptionsFlag, GameOfLife &gameOfLife, Figure &reset_figure, std::string &message, float &zoom)

  initialize Load user interface
- static void screenshot_UI (UI_Flag &uiFlag, GameOfLife &gameOfLife, std::string &message)

  initialize Screenshot user interface
- static void messageBox_UI (UI_Flag &uiFlag, std::string &message)

  initialize MessageBox user interface
- static void ColorPicker (GameOfLife &gameOfLife, const char *label, ImU32 *color)

  creates colorized rectangle with color picker function
- static void load_example_UI (UI_Flag &uiFlag, UI_Options_Flag &uiOptionsFlag, GameOfLife &gameOfLife, Figure &reset_figure, float &zoom, int &speed)

  initialize load example user interface
- static void Dockspace_UI ()

  initialize Dockspace Window for user interface windows like Grid_UI or Setting_UI

Static Private Member Functions

- static void print (GameOfLife *_gameOfLife, float zoom)

  print rectangles with size of zoom << used by gol::UI::grid_UI() >>
- static void button_setting_zoom (UI_Options_Flag &uiOptionsFlag, float &factor)

  create zoom in/out buttons << used by gol::UI::setting_UI() >>
- static void button_setting_speed (int &factor)

  create speed up/down buttons << used by gol::UI::setting_UI() >>
- static void button_setting_step (GameOfLife &gameOfLife, Figure_Stack &_stack)

  create step for/back buttons << used by gol::UI::setting_UI() >>
- static void button_setting_run (UI_Options_Flag &uiOptionsFlag)

  create run button << used by gol::UI::setting_UI() >>
- static void button_setting_reset (GameOfLife &gameOfLife, Figure &reset_figure, Figure_Stack &figureStack)

  create stop button << used by gol::UI::setting_UI() >>

- static void button_setting_clear (GameOfLife &gameOfLife, Figure &reset_figure, Figure_Stack &figureStack)

  Create clear button << used by gol::UI::setting_UI() >>
- static void button_setting_random (GameOfLife &gameOfLife, Figure &reset_figure, Figure_Stack &figure↩ Stack)

  create random button << used by gol::UI::setting_UI() >>
- static void button_setting_load (UI_Flag &uiFlag)

  create load button << used by gol::UI::setting_UI() >>
- static void button_setting_load_exp (UI_Flag &uiFlag)

  create example load button << used by gol::UI::setting_UI() >>
- static void button_setting_screenshot (UI_Flag &uiFlag)

  create screenshot button << used by gol::UI::setting_UI() >>
- static void button_setting_reset_rules (Rule *_rule)

  create reset rules button << used by gol::UI::setting_UI() >>
- static void button_setting_reset_colors (RuleColor *_rule)

  create reset colors button << used by gol::UI::setting_UI() >>
- static void setup_rules (UI_Options_Flag &uiOptionsFlag, GameOfLife &gameOfLife)

  create checkboxes for rule and color modifications << used by gol::UI::setting_UI() >>
- static void button_load (UI_Flag &uiFlag, UI_Options_Flag &uiOptionsFlag, GameOfLife &gameOfLife, Figure &reset_figure, std::string path, Vec2 position, int &angle, std::string &message, float &zoom)

  creates load button to load figure from file into grid << used by gol::UI::load_UI() >>
- static void button_save (UI_Flag &uiFlag, GameOfLife &gameOfLife, std::string path)

  creates save button to save figure into a file from grid << used by gol::UI::save_UI() >>

5.8.1 Detailed Description

Definition at line 415 of file game_of_life.h.

5.8.2 Member Function Documentation

5.8.2.1 button_load()

```
void gol::UI::button_load (
        UI_Flag & uiFlag,
        UI_Options_Flag & uiOptionsFlag,
        GameOfLife & gameOfLife,
        Figure & reset_figure,
        std::string path,
        Vec2 position,
        int & angle,
        std::string & message,
        float & zoom )  [static], [private]
```

creates load button to load figure from file into grid << used by gol::UI::load_UI() >>

Parameters

uiFlag	: &UI_Flag -> includes load_UI & message_UI flags to enable/disable window
uiOptionsFlag	: &UI_Options_Flag -> used to enable auto scroll and auto zoom
gameOfLife	: &GameOfLife -> includes grid & cell information
reset_figure	: &Figure -> figure which includes reset grid states
path	: std::string -> full filepath to .lif data
position	: Vec2 -> position upper left corner of Figure in grid
angle	: &int -> angle of figure in grid << 0=0°,1=90°,2=180°,3=270° >>
message	: &std::string -> error message << used by gol::UI::message_UI() >>

Definition at line 894 of file game_of_life.cpp.

```
00894
    {
00895         ImGui::PushID(2);
00896         //color green
00897         ImGui::PushStyleColor(ImGuiCol_Button, (ImVec4)ImColor::HSV(2 / 7.0f, 0.6f, 0.6f));
00898         ImGui::PushStyleColor(ImGuiCol_ButtonHovered, (ImVec4)ImColor::HSV(2 / 7.0f, 0.7f, 0.7f));
00899         ImGui::PushStyleColor(ImGuiCol_ButtonActive, (ImVec4)ImColor::HSV(2 / 7.0f, 0.8f, 0.8f));
00900         if (ImGui::Button("Load")) {
00901
00902             try {
00903                 std::string _path = (std::string) std::move(path);
00904                 auto *_f = new gol::Figure(_path);
00905
00906                 // ERROR HANDLING
00907                 if(_f->get_dimension().x+position.x > gameOfLife.get_grid()->get_dimension().x){throw
    std::logic_error("ERROR: Figure Dimension x in comination with position x are to large");}
00908                 if(_f->get_dimension().y+position.y > gameOfLife.get_grid()->get_dimension().y){throw
    std::logic_error("ERROR: Figure Dimension y in comination with position y are to large");}
00909
00910
00911                 gameOfLife.populate_figure(_f, position, (angle / 90));
00912                 reset_figure = Figure(&gameOfLife);
00913                 uiFlag &= ~UI_Flag_enable_LoadUI;
00914                 delete _f;
00915
00916                 //Colorize grid
00917                 for(auto c: gameOfLife.get_grid()->get_cells()){
00918                     c->get_neighbours()->set_n_alive(0);
00919                     for(auto n: c->get_neighbours()->get_cells()){
00920                         if(n->get_state()){
    c->get_neighbours()->set_n_alive(c->get_neighbours()->get_n_alive()+1); }
00921                     }
00922
    c->set_color(gameOfLife.get_color_rules()->color[c->get_neighbours()->get_n_alive()]);
00923                 }
00924                 //zoom = gameOfLife.get_grid()->get_auto_zoom_factor();
00925                 uiOptionsFlag |= UI_Options_Flag_enable_AutoZoom | UI_Options_Flag_enable_AutoScroll;
00926             }catch(std::logic_error &err){
00927                 message = err.what();
00928                 uiFlag |= UI_Flag_enable_MessageUI;
00929             }
00930
00931
00932
00933         }
00934         ImGui::PopStyleColor(3);
00935         ImGui::PopID();
00936     }
```

5.8.2.2 button_save() `void gol::UI::button_save (`

`UI_Flag & uiFlag,`

`GameOfLife & gameOfLife,`

`std::string path ) [static], [private]`

creates save button to save figure into a file from grid << used by gol::UI::save_UI() >>

Parameters

uiFlag	: &UI_Flag -> includes save_UI flags to enable/disable window
gameOfLife	: &GameOfLife -> includes grid & cell information
path	: std::string -> full filepath to new .lif data

Definition at line 937 of file game_of_life.cpp.

```
00937                                                                                                 {
00938         //color green
00939         ImGui::PushID(2);
00940         ImGui::PushStyleColor(ImGuiCol_Button, (ImVec4)ImColor::HSV(2 / 7.0f, 0.6f, 0.6f));
00941         ImGui::PushStyleColor(ImGuiCol_ButtonHovered, (ImVec4)ImColor::HSV(2 / 7.0f, 0.7f, 0.7f));
00942         ImGui::PushStyleColor(ImGuiCol_ButtonActive, (ImVec4)ImColor::HSV(2 / 7.0f, 0.8f, 0.8f));
00943         if (ImGui::Button("Save")) {
00944             std::string Filename(path);
```

```
00945
00946             //Grid* grid = gameOfLife.get_grid();
00947
00948             auto *_fig = new gol::Figure(&gameOfLife);
00949
00950             _fig->save_to_file(path);
00951
00952             uiFlag &= ~UI_Flag_enable_SaveUI;
00953
00954         }
00955         ImGui::PopStyleColor(3);
00956         ImGui::PopID();
00957     }
```

5.8.2.3 button_setting_clear() void gol::UI::button_setting_clear (GameOfLife & *gameOfLife*, Figure & *reset_figure*, Figure_Stack & *figureStack*) [static], [private]

Create clear button << used by gol::UI::setting_UI() >>

Parameters

gameOfLife	: &GameOfLife -> includes grid & cell information
reset_figure	: &Figure -> figure which includes reset grid states
figureStack	: &Figure_Stack -> STACK_SIZE pre Figures for reverse step

Definition at line 859 of file game_of_life.cpp.

```
00859
      {
00860         const ImVec2 size_clr = ImVec2(44,20.);
00861         if (ImGui::Button("clear",size_clr)) {
00862             gameOfLife.clear_grid();
00863             reset_figure = Figure(&gameOfLife);
00864             figureStack.clear();
00865         }
00866     }
```

5.8.2.4 button_setting_load() void gol::UI::button_setting_load (UI_Flag & *uiFlag*) [static], [private]

create load button << used by gol::UI::setting_UI() >>

Parameters

uiFlag	: &UI_Flag -> includes load_ui flag to enable/disable window

Definition at line 876 of file game_of_life.cpp.

```
00876                                           {
00877         const ImVec2 size_load = ImVec2(200.,20.);
00878         if (ImGui::Button("load from file", size_load)) {
00879             uiFlag |= UI_Flag_enable_LoadUI;
00880         }
00881     }
```

5.8.2.5 button_setting_load_exp() void gol::UI::button_setting_load_exp (UI_Flag & *uiFlag*) [static], [private]

create example load button << used by gol::UI::setting_UI() >>

Parameters

uiFlag	: &UI_Flag -> includes load_exp_ui flag to enable/disable window

Definition at line 882 of file game_of_life.cpp.

```
00882                                                   {
00883         const ImVec2 size_load = ImVec2(200.,20.);
00884         if (ImGui::Button("load example", size_load)) {
00885             uiFlag |= UI_Flag_enable_LoadExampleUI;
00886         }
00887     }
```

5.8.2.6 button_setting_random() void gol::UI::button_setting_random (GameOfLife & *gameOfLife*, Figure & *reset_figure*, Figure_Stack & *figureStack*) [static], [private]

create random button << used by gol::UI::setting_UI() >>

Parameters

gameOfLife	: &GameOfLife -> includes grid & cell information
reset_figure	: &Figure -> figure which includes reset grid states
figureStack	: &Figure_Stack -> STACK_SIZE pre Figures for reverse step

Definition at line 867 of file game_of_life.cpp.

```
00867
      {
00868         const ImVec2 size_rand = ImVec2(200.,20.);
00869         if (ImGui::Button("random", size_rand)) {
00870             gameOfLife.clear_grid();
00871             gameOfLife.populate_random();
00872             reset_figure = Figure(&gameOfLife);
00873             figureStack.clear();
00874         }
00875     }
```

5.8.2.7 button_setting_reset() void gol::UI::button_setting_reset (GameOfLife & *gameOfLife*, Figure & *reset_figure*, Figure_Stack & *figureStack*) [static], [private]

create stop button << used by gol::UI::setting_UI() >>

Parameters

uiOptionsFlag	: &UI_Options_Flag -> includes run option flag

Create reset button << used by gol::UI::setting_UI() >>

Parameters

gameOfLife	: &GameOfLife -> includes grid & cell information
reset_figure	: &Figure -> figure which includes reset grid states
figureStack	: &Figure_Stack -> STACK_SIZE pre Figures for reverse step

Definition at line 814 of file game_of_life.cpp.

```
00814
      {
00815         const ImVec2 size_res = ImVec2(44,20.);
00816         // create button
00817         if (ImGui::Button("reset", size_res)) {
00818             gameOfLife.clear_grid();
00819             gameOfLife.populate_figure(&reset_figure, (Vec2){0,0},0);
00820             //Colorize grid
00821             for(auto c: gameOfLife.get_grid()->get_cells()){
00822                 c->get_neighbours()->set_n_alive(0);
00823                 for(auto n: c->get_neighbours()->get_cells()){
00824                     if(n->get_state()){
      c->get_neighbours()->set_n_alive(c->get_neighbours()->get_n_alive()+1); }
00825                 }
00826                 c->set_color(gameOfLife.get_color_rules()->color[c->get_neighbours()->get_n_alive()]);
00827             }
00828
00829             figureStack.clear();
00830         }
00831     }
```

5.8.2.8 button_setting_reset_colors() void gol::UI::button_setting_reset_colors (RuleColor * *_rule*) [static], [private]

create reset colors button << used by gol::UI::setting_UI() >>

Parameters

_rule	: *RuleColor -> rules which will be reset to all white

Definition at line 1295 of file game_of_life.cpp.

```
01295                                                   {
01296
01297         const ImVec2 size = ImVec2(200.,20.);
01298         if (ImGui::Button("Reset colors", size)) {
01299             // all colors are white
01300             *_rule = gol::RuleColor{
01301                     IM_COL32(255,0,0,255),
01302                     IM_COL32(198,57,0,255),
01303                     IM_COL32(141,114,0,255),
01304                     IM_COL32(84,171,0,255),
01305                     IM_COL32(0,255,0,255),
01306                     IM_COL32(0,198,57,255),
01307                     IM_COL32(0,141,114,255),
01308                     IM_COL32(0,84,171,255),
01309                     IM_COL32(0,0,255,255)
01310             };
01311         }
01312
01313     }
```

5.8.2.9 button_setting_reset_rules() void gol::UI::button_setting_reset_rules (Rule * *_rule*) [static], [private]

create reset rules button << used by gol::UI::setting_UI() >>

Parameters

_rule	: *Rule -> rules which will be reset to Cornways Game of life [23/3]

Definition at line 1261 of file game_of_life.cpp.

```
01261                                                {
01262         const ImVec2 size = ImVec2(200.,20.);
01263         if (ImGui::Button("Cornways", size)) {
01264             for(int r=0; r < 9; r++){
01265                 // reset game of life rules to cornways (23/3)
01266                 if(r == 3){_rule->alive[r] = true; }else{ _rule->alive[r] = false; }
01267                 if((r == 2) || (r == 3)){ _rule->death[r] = false; }else{ _rule->death[r] = true; }
01268             }
01269         }
01270         if (ImGui::Button("Anti Cornways", size)) {
01271             for(int r=0; r < 9; r++){
01272                 // reset game of life rules to cornways (56/5)
01273                 if(r == 5){_rule->death[r] = true; }else{_rule->death[r] = false; }
01274                 if((r == 5) || (r == 6)){_rule->alive[r] = false; }else{ _rule->alive[r] = true; }
01275
01276             }
01277         }
01278         if (ImGui::Button("Kopiersystem", size)) {
01279             for(int r=0; r < 9; r++){
01280                 // reset game of life rules to (1357/1357)
01281                 if((r==1)||(r==3)||(r==5)||(r==7)){_rule->alive[r] = true; }else{_rule->alive[r] =
      false; }
01282                 if((r==1)||(r==3)||(r==5)||(r==7)){_rule->death[r] = false; }else{_rule->death[r] =
      true; }
01283             }
01284         }
01285         if (ImGui::Button("Anti Kopiersystem", size)) {
01286             for(int r=0; r < 9; r++){
01287                 // reset game of life rules to (1357/1357)
01288                 if((r==1)||(r==3)||(r==5)||(r==7)){_rule->death[r] = true; }else{_rule->death[r] =
      false; }
01289                 if((r==1)||(r==3)||(r==5)||(r==7)){_rule->alive[r] = false; }else{_rule->alive[r] =
      true; }
01290             }
01291         }
01292
01293
01294     }
```

5.8.2.10 button_setting_run() `void gol::UI::button_setting_run (`

`UI_Options_Flag & uiOptionsFlag ) [static], [private]`

create run button << used by gol::UI::setting_UI() >>

Parameters

uiOptionsFlag	: &UI_Options_Flag -> includes run option flag

Definition at line 772 of file game_of_life.cpp.

```
00772                                                        {
00773         const ImVec2 size_run = ImVec2(96,20.);
00774         if(!(uiOptionsFlag & UI_Options_Flag_Run)) {
00775             ImGui::PushID(2);
00776             //color green
00777             ImGui::PushStyleColor(ImGuiCol_Button, (ImVec4) ImColor::HSV(2 / 7.0f, 0.6f, 0.6f));
00778             ImGui::PushStyleColor(ImGuiCol_ButtonHovered, (ImVec4) ImColor::HSV(2 / 7.0f, 0.7f,
      0.7f));
00779             ImGui::PushStyleColor(ImGuiCol_ButtonActive, (ImVec4) ImColor::HSV(2 / 7.0f, 0.8f, 0.8f));
00780             // create button
00781             if (ImGui::Button("RUN", size_run)) {
00782                 uiOptionsFlag |= UI_Options_Flag_Run;
00783             }
00784             ImGui::PopStyleColor(3);
00785             ImGui::PopID();
00786         }else{
```

```
00787             ImGui::PushID(0);
00788             //color red
00789             ImGui::PushStyleColor(ImGuiCol_Button, (ImVec4)ImColor::HSV(0 / 7.0f, 0.6f, 0.6f));
00790             ImGui::PushStyleColor(ImGuiCol_ButtonHovered, (ImVec4)ImColor::HSV(0 / 7.0f, 0.7f, 0.7f));
00791             ImGui::PushStyleColor(ImGuiCol_ButtonActive, (ImVec4)ImColor::HSV(0 / 7.0f, 0.8f, 0.8f));
00792             // create button
00793             if (ImGui::Button("STOP", size_run)) {
00794                 uiOptionsFlag &= ~UI_Options_Flag_Run;
00795             }
00796             ImGui::PopStyleColor(3);
00797             ImGui::PopID();
00798         }
00799     }
```

5.8.2.11 button_setting_screenshot() void gol::UI::button_setting_screenshot (UI_Flag & *uiFlag*) [static], [private]

create screenshot button << used by gol::UI::setting_UI() >>

Parameters

uiFlag	: &UI_Flag -> includes ui flag to enable/disable window

Definition at line 888 of file game_of_life.cpp.

```
00888                                                 {
00889         const ImVec2 size = ImVec2(200.,20.);
00890         if (ImGui::Button("screenshot to .lif file", size)) {
00891             uiFlag |= UI_Flag_enable_SaveUI;
00892         }
00893     }
```

5.8.2.12 button_setting_speed() void gol::UI::button_setting_speed (int & *factor*) [static], [private]

create speed up/down buttons << used by gol::UI::setting_UI() >>

Parameters

factor	: &int -> speed factor

Definition at line 760 of file game_of_life.cpp.

```
00760                                             {
00761         // speed up button
00762         const ImVec2 size = ImVec2(96,20.);
00763         if (ImGui::Button("Speed +",size)) {
00764             if(factor > 0){factor--;}
00765         }
00766         ImGui::SameLine();
00767         // speed down button
00768         if (ImGui::Button("speed -",size)) {
00769             factor++;
00770         }
00771     }
```

5.8.2.13 button_setting_step() void gol::UI::button_setting_step (GameOfLife & *gameOfLife*, Figure_Stack & *_stack*) [static], [private]

create step for/back buttons << used by gol::UI::setting_UI() >>

Parameters

gameOfLife	: &GameOfLife -> includes grid & cell information
_stack	: &Figure_Stack -> includes STACK_SIZE pre Figures for reverse step

Definition at line 832 of file game_of_life.cpp.

```
00832                                                                                {
00833         const ImVec2 size = ImVec2(96,20.);
00834         // create back button
00835         if (ImGui::Button("« Step", size)) {
00836             if(!_stack.empty()){
00837 
00838                 gameOfLife.populate_figure(_stack.back(), Vec2{0, 0}, 0);
00839                 _stack.pop_back();
00840                 // COLORIZE
00841                 for(auto c: gameOfLife.get_grid()->get_cells()){
00842                     c->get_neighbours()->set_n_alive(0);
00843                     for(auto n: c->get_neighbours()->get_cells()){
00844                         if(n->get_state()){
    c->get_neighbours()->set_n_alive(c->get_neighbours()->get_n_alive()+1);}
00845                     }
00846 
    c->set_color(gameOfLife.get_color_rules()->color[c->get_neighbours()->get_n_alive()]);
00847                 }
00848             }
00849         }
00850         ImGui::SameLine();
00851         // step for button
00852         if (ImGui::Button("Step »", size)) {
00853             _stack.push_back(new Figure(&gameOfLife));
00854             if(_stack.size() > STACK_SIZE){ _stack.pop_front(); }
00855             gameOfLife.refresh_grid();
00856 
00857         }
00858     }
```

5.8.2.14 button_setting_zoom()

```
void gol::UI::button_setting_zoom (
        UI_Options_Flag & uiOptionsFlag,
        float & factor )  [static], [private]
```

create zoom in/out buttons << used by gol::UI::setting_UI() >>

Parameters

factor	: &float -> zoom factor
uiOptionsFlag	: &UI_Options_Flag -> includes auto zoom enabled option flag

Definition at line 739 of file game_of_life.cpp.

```
00739                                                                                {
00740 
00741         static bool auto_zoom;
00742         if(uiOptionsFlag & UI_Options_Flag_enable_AutoZoom){ auto_zoom = true; }else{ auto_zoom =
    false; }
00743         ImGui::Checkbox("Auto Zoom", &auto_zoom);
00744         if(auto_zoom){uiOptionsFlag |= UI_Options_Flag_enable_AutoZoom; }else{uiOptionsFlag &=
    ~UI_Options_Flag_enable_AutoZoom; }
00745 
00746         // Zoom in button
00747         const ImVec2 size = ImVec2(96,20.);
00748         if(!(uiOptionsFlag & UI_Options_Flag_enable_AutoZoom)) {
00749             if (ImGui::Button("Zoom +", size)) {
00750 
00751                 factor +=2;
00752             }
00753             ImGui::SameLine();
```

```
            // Zoom out button
            if (ImGui::Button("Zoom -", size)) {
                if (factor > 2) { factor -=2 ; }else{ factor = 2; }
            }
        }
    }
```

5.8.2.15 ColorPicker()

`void gol::UI::ColorPicker ( GameOfLife & gameOfLife, const char * label, ImU32 * color ) [static]`

creates colorized rectangle with color picker function

Parameters

gameOfLife	: gol::GameOfLife -> includes grid & cell information
label	: const char* -> label of rectangle
color	: unsigned integer* -> color of rectangle

Definition at line 1175 of file game_of_life.cpp.

```
01175                                                                               {
01176         //ImGui::Begin("Color Picker");
01177             float col[4];
01178             col[0] = (float)((*color      ) & 0xFF) / 255.0f;
01179             col[1] = (float)((*color » 8 ) & 0xFF) / 255.0f;
01180             col[2] = (float)((*color » 16) & 0xFF) / 255.0f;
01181             col[3] = (float)((*color » 24) & 0xFF) / 255.0f;
01182 
01183             static bool drag_and_drop = false;
01184             static bool hdr = true;
01185             static bool alpha_preview = true;
01186             static bool alpha_half_preview = true;
01187             static bool options_menu = true;
01188             ImGuiColorEditFlags misc_flags = (hdr ? ImGuiColorEditFlags_HDR : 0) | (drag_and_drop ? 0
      : ImGuiColorEditFlags_NoDragDrop) | (alpha_half_preview ? ImGuiColorEditFlags_AlphaPreviewHalf :
      (alpha_preview ? ImGuiColorEditFlags_AlphaPreview : 0)) | (options_menu ? 0 :
      ImGuiColorEditFlags_NoOptions);
01189             //ImGui::ColorPicker4("##picker", &col[0], misc_flags | ImGuiColorEditFlags_NoSidePreview
      | ImGuiColorEditFlags_NoSmallPreview);
01190             //ImGui::ColorButton("##current", (ImVec4){col[0],col[1],col[2],col[3]},
      ImGuiColorEditFlags_NoPicker | ImGuiColorEditFlags_AlphaPreviewHalf, ImVec2(60, 40));
01191             //ImGuiColorEditFlags palette_button_flags = ImGuiColorEditFlags_NoAlpha |
      ImGuiColorEditFlags_NoPicker | ImGuiColorEditFlags_NoTooltip;
01192             //ImGui::ColorButton("##palette", (ImVec4){col[0],col[1],col[2],col[3]},
      palette_button_flags, ImVec2(20, 20));
01193 
01194             ImGui::ColorEdit4(label, &col[0], ImGuiColorEditFlags_NoInputs | misc_flags);
01195 
01196 
01197             ImU32 _col = ((ImU32)(col[0] * 255.0f)       ) |
01198                          ((ImU32)(col[1] * 255.0f) «  8) |
01199                          ((ImU32)(col[2] * 255.0f) « 16) |
01200                          ((ImU32)(col[3] * 255.0f) « 24);
01201 
01202             *color = _col;
01203 
01204 
01205 
01206 
01207         //ImGui::End();
01208     }
```

5.8.2.16 Dockspace_UI()

`void gol::UI::Dockspace_UI ( ) [static]`

initialize Dockspace Window for user interface windows like Grid_UI or Setting_UI

Definition at line 999 of file game_of_life.cpp.

```
00999                           {
01000         static ImGuiDockNodeFlags dockspace_flags = ImGuiDockNodeFlags_None |
      ImGuiDockNodeFlags_PassthruCentralNode;
01001
01002         ImGuiWindowFlags window_flags = ImGuiWindowFlags_NoDocking;
01003         window_flags |= ImGuiWindowFlags_NoTitleBar | ImGuiWindowFlags_NoCollapse |
      ImGuiWindowFlags_NoResize | ImGuiWindowFlags_NoMove;
01004         window_flags |= ImGuiWindowFlags_NoBringToFrontOnFocus | ImGuiWindowFlags_NoNavFocus;
01005         window_flags |= ImGuiWindowFlags_NoBackground ;
01006
01007         const ImGuiViewport* viewport = ImGui::GetMainViewport();
01008         ImGui::SetNextWindowPos(viewport->WorkPos);
01009         ImGui::SetNextWindowSize(viewport->WorkSize);
01010
01011         ImGui::Begin("DockSpace", nullptr, window_flags);
01012
01013         ImGuiID dockspace_id = ImGui::GetID("MyDockSpace");
01014         ImGui::DockSpace(dockspace_id, ImVec2(0.0f, 0.0f), dockspace_flags);
01015
01016         ImGui::End();
01017     }
```

5.8.2.17 grid_UI() void gol::UI::grid_UI (UI_Flag & *uiFlag*, UI_Options_Flag & *uiOptionsFlag*, GameOfLife & *gameOfLife*, Figure_Stack & *figureStack*, float & *zoom*, int & *speed*, int & *count*) [static]

initialize Grid user interface

Parameters

uiFlag	: &UI_Flag -> includes Grid_UI & setting_UI flags to enable/disable window
uiOptionsFlag	: &UI_Options_Flag -> used to enable auto scroll, auto zoom and run
gameOfLife	: gol::GameOfLife -> includes grid & cell information
zoom	: &float -> zoom factor
speed	: &int -> speed factor
count	: &int -> counter required by speed

Definition at line 1018 of file game_of_life.cpp.

```
01018
      {
01019
01020
01021
01022
01023         ImGui::Begin("GRID",NULL,ImGuiWindowFlags_HorizontalScrollbar /*| ImGuiWindowFlags_NoMove */|
      ImGuiWindowFlags_MenuBar | ImGuiWindowFlags_NoTitleBar);
01024
01025         //Auto Scroll Begin
01026         if(uiOptionsFlag & UI_Options_Flag_enable_AutoScroll) {
01027
01028             ImGui::SetScrollX(gameOfLife.get_grid()->get_scrollx_position()*(ImGui::GetScrollMaxX()));
      //TODO
01029             ImGui::SetScrollY(gameOfLife.get_grid()->get_scrolly_position()*(ImGui::GetScrollMaxY()));
      //TODO
01030
01031             //std::cerr << ImGui::GetContentRegionMax().x << "x" << ImGui::GetContentRegionMax().y << " <-
      ";
01032             //std::cerr << ImGui::GetScrollMaxX() << "x" << ImGui::GetScrollMaxY() << "\n"; //TODO debug
01033         }
01034         //Auto Scroll End
01035         //Auto Zoom Begin
01036         if(uiOptionsFlag & UI_Options_Flag_enable_AutoZoom) {
```

```
01037             if ((gameOfLife.get_grid()->get_real_grid_dimension().x > 0) &&
      (gameOfLife.get_grid()->get_real_grid_dimension().y > 0)) {
01038                 zoom = gameOfLife.get_grid()->get_auto_zoom_factor();
01039             }
01040         }
01041         //Auto Zoom End
01042
01043
01044
01045         ImGuiIO& io = ImGui::GetIO(); (void)io;
01046         ImGui::Text("Application average %.3f ms/frame (%.2f FPS)", 1000.0f / io.Framerate,
      io.Framerate); //TODO: FOR DEBUG
01047
01048         // Menubar
01049         static bool setting = false;
01050         static bool run;
01051         if(uiFlag & UI_Flag_enable_SetupUI){ setting = true; }
01052         if(uiOptionsFlag & UI_Options_Flag_Run){ run = true; }else{run = false; }
01053         if (ImGui::BeginMenuBar()){
01054             if (ImGui::BeginMenu("Options")){
01055                 ImGui::MenuItem("Settings", NULL, &setting);
01056                 ImGui::MenuItem("run", NULL, &run); //TODO
01057                 ImGui::EndMenu();
01058             }
01059
01060         }
01061         ImGui::EndMenuBar();
01062         if(setting){ uiFlag |= UI_Flag_enable_SetupUI; }else{ uiFlag &= ~UI_Flag_enable_SetupUI; }
01063         if(run){ uiOptionsFlag |= UI_Options_Flag_Run; }else{ uiOptionsFlag &= ~UI_Options_Flag_Run; }
01064         // print grid to imGUI figure
01065         UI::print(&gameOfLife, zoom);
01066
01067         // run
01068         if (uiOptionsFlag & UI_Options_Flag_Run) {
01069             if (count > speed) { count = speed; }
01070             if (count == speed) {
01071                 figureStack.push_back(new Figure(&gameOfLife));
01072                 if(figureStack.size() > STACK_SIZE){ figureStack.pop_front(); }
01073                 gameOfLife.refresh_grid();
01074                 count = 0;
01075             } else { count++; }
01076         }
01077
01078         ImGui::End();
01079     }
```

5.8.2.18 load_example_UI()

```
void gol::UI::load_example_UI (
        UI_Flag & uiFlag,
        UI_Options_Flag & uiOptionsFlag,
        GameOfLife & gameOfLife,
        Figure & reset_figure,
        float & zoom,
        int & speed )  [static]
```

initialize load example user interface

Parameters

uiFlag	: &UI_Flag -> includes UI Flags to enable/disable window
uiOptionsFlag	: &UI_Options_Flag -> used to enable auto scroll, auto zoom
gameOfLife	: gol::GameOfLife -> includes grid & cell information
reset_figure	: gol::Figure -> reference required for resetting grid
zoom	: &float -> zoom factor
speed	: &int -> speed factor

Definition at line 1315 of file game_of_life.cpp.

```
01315
      {
```

```
01316         static int x_pos = 0;
01317         static int y_pos = 0;
01318         static int angle = 0;
01319 
01320         static bool example_flags[] = {true,false,false,false,false,false,false,false};
01321         bool last_flags[8];
01322 
01323 
01324 
01325 
01326         for(int i=0; i < 8; i++){
01327             last_flags[i] = example_flags[i];
01328         }
01329 
01330         ImGui::Begin("Load example figures", nullptr, 0/*ImGuiWindowFlags_NoMove*/);
01331 
01332         ImGui::Text("Cornway's example figures");
01333         ImGui::Checkbox("Glider", &example_flags[0]);
01334         ImGui::Checkbox("Light-Weight Spaceship", &example_flags[1]);
01335         ImGui::Checkbox("Middle-Weight Spaceship", &example_flags[2]);
01336         ImGui::Checkbox("Heavy-Weight Spaceship", &example_flags[3]);
01337         ImGui::Checkbox("Gosper Gun", &example_flags[4]);
01338         ImGui::Checkbox("Eater", &example_flags[5]);
01339 
01340         ImGui::Text("\nother example figures");
01341         ImGui::Checkbox("HTW Logo", &example_flags[6]);
01342         ImGui::Checkbox("Pacman", &example_flags[7]);
01343 
01344         bool equal = true;
01345         for(int i=0; i < 8; i++){
01346             if(example_flags[i] != last_flags[i]){
01347                 equal = false;
01348             }
01349         }
01350 
01351         if(!equal){
01352             for(int i=0; i < 8; i++){
01353                 example_flags[i] =  (example_flags[i] + last_flags[i])%2; // XOR
01354             }
01355         }
01356 
01357         ImGui::Text("enter x and y position ");
01358         ImGui::InputInt("x", &x_pos); // Textfield x position
01359         if(x_pos > gameOfLife.get_grid()->get_dimension().x){x_pos =
      gameOfLife.get_grid()->get_dimension().x; }
01360         if(x_pos < 0){x_pos = 0;}
01361 
01362         ImGui::InputInt("y", &y_pos); // Textfield y position
01363         if(y_pos > gameOfLife.get_grid()->get_dimension().y){ y_pos =
      gameOfLife.get_grid()->get_dimension().y; }
01364         if(y_pos < 0){y_pos = 0;}
01365 
01366 
01367         ImGui::Text("enter angle of figure");
01368         ImGui::InputInt("angle", &angle, 90, 90, 0); // Textfield x position
01369         if(angle > 270) {
01370             angle = 270;
01371         }
01372         if(angle < 0){
01373             angle = 0;
01374         }
01375 
01376         ImGui::PushID(2);
01377         //color green
01378         ImGui::PushStyleColor(ImGuiCol_Button, (ImVec4)ImColor::HSV(2 / 7.0f, 0.6f, 0.6f));
01379         ImGui::PushStyleColor(ImGuiCol_ButtonHovered, (ImVec4)ImColor::HSV(2 / 7.0f, 0.7f, 0.7f));
01380         ImGui::PushStyleColor(ImGuiCol_ButtonActive, (ImVec4)ImColor::HSV(2 / 7.0f, 0.8f, 0.8f));
01381         if (ImGui::Button("Load")) {
01382             gol::Rule r_cornway = (gol::Rule){{false,false, false, true, false, false, false, false,
      false},
01383                                               {true, true, false, false, true, true, true, true,
      true}};
01384 
01385             gol::Rule r_copy = (gol::Rule){{false,true, false, true, false, true, false, true, false},
01386                                            {true, false, true, false, true, false, true, false,
      true}};
01387 
01388             if(example_flags[0]){
01389                 //Figure *f = new Figure(glider, glider_dim);
01390 
01391                 uiOptionsFlag |= UI_Options_Flag_enable_AutoScroll | UI_Options_Flag_enable_AutoZoom;
01392                 std::string glider_string = "bob$2bo$3o!";
01393                 Vec2 glider_dim = {3, 3};
01394                 auto *f = new Figure(glider_string, glider_dim);
01395                 gameOfLife.populate_figure(f,(Vec2){x_pos,y_pos}, angle/90);
01396                 gameOfLife.set_rules(r_cornway);
01397                 zoom = gameOfLife.get_grid()->get_auto_zoom_factor();
```

```
01398             }
01399             if(example_flags[1]){
01400                 uiOptionsFlag |= UI_Options_Flag_enable_AutoScroll | UI_Options_Flag_enable_AutoZoom;
01401                 //Figure *f = new Figure(lws, lws_dim);
01402                 std::string lws_string = "b4o$o3bo$4bo$o2b1o!";
01403                 Vec2 lws_dim = {5, 4};
01404                 auto *f = new Figure(lws_string, lws_dim);
01405                 gameOfLife.populate_figure(f,(Vec2){x_pos,y_pos}, angle/90);
01406                 gameOfLife.set_rules(r_cornway);
01407                 zoom = gameOfLife.get_grid()->get_auto_zoom_factor();
01408 
01409             }
01410             if(example_flags[2]){
01411                 uiOptionsFlag |= UI_Options_Flag_enable_AutoScroll | UI_Options_Flag_enable_AutoZoom;
01412                 //Figure *f = new Figure(mws, mws_dim);
01413                 std::string mws_string = "b5o$o4bo$5bo$o3bob$2bo3b!";
01414                 Vec2 mws_dim = {6, 5};
01415                 auto *f = new Figure(mws_string, mws_dim);
01416                 gameOfLife.populate_figure(f,(Vec2){x_pos,y_pos}, angle/90);
01417                 gameOfLife.set_rules(r_cornway);
01418                 zoom = gameOfLife.get_grid()->get_auto_zoom_factor();
01419 
01420             }
01421             if(example_flags[3]){
01422                 uiOptionsFlag |= UI_Options_Flag_enable_AutoScroll | UI_Options_Flag_enable_AutoZoom;
01423 
01424                 //Figure *f = new Figure(hws, hws_dim);
01425                 std::string hws_string = "b6o$o5bo$6bo$o4bob$2b2o2b!";
01426                 Vec2 hws_dim = {7, 5};
01427                 auto *f = new Figure(hws_string, hws_dim);
01428                 gameOfLife.populate_figure(f,(Vec2){x_pos,y_pos}, angle/90);
01429                 gameOfLife.set_rules(r_cornway);
01430                 zoom = gameOfLife.get_grid()->get_auto_zoom_factor();
01431             }
01432             if(example_flags[4]){
01433                 uiOptionsFlag |= UI_Options_Flag_enable_AutoScroll | UI_Options_Flag_enable_AutoZoom;
01434 
01435                 Vec2 gosper_dim = {36, 9};
01436                 std::string gosper_string =
    "24bo$22bobo$12b2o6b2o12b2o$11bo3bo4b2o12b2o$2o8bo5bo3b2o$2o8bo3bob2o4bobo$10bo5bo7bo$11bo3bo$12b2o!";
01437 
01438                 auto *f = new Figure(gosper_string, gosper_dim);
01439                 gameOfLife.populate_figure(f,(Vec2){x_pos,y_pos}, angle/90);
01440                 gameOfLife.set_rules(r_cornway);
01441                 zoom = gameOfLife.get_grid()->get_auto_zoom_factor();
01442             }
01443             if(example_flags[5]){
01444                 uiOptionsFlag |= UI_Options_Flag_enable_AutoScroll | UI_Options_Flag_enable_AutoZoom;
01445                 //Figure *f = new Figure(eater, eater_dim);
01446                 std::string eater_string = "2o2b$obob$2bob$2b2o!";
01447                 Vec2 eater_dim = {4, 4};
01448                 auto *f = new Figure(eater_string, eater_dim);
01449                 gameOfLife.populate_figure(f,(Vec2){x_pos,y_pos}, angle/90);
01450                 gameOfLife.set_rules(r_cornway);
01451                 zoom = gameOfLife.get_grid()->get_auto_zoom_factor();
01452             }
01453             if(example_flags[6]){
01454                 uiOptionsFlag |= UI_Options_Flag_enable_AutoScroll | UI_Options_Flag_enable_AutoZoom;
01455                 //Figure *f = new Figure(htw, htw_dim);
01456                 std::string htw_string =
    "10o30b10o$10o30b10o$10o30b10o$10o30b10o$10o30b10o$10o30b10o$10o30b10o$10o30b10o$10o30b10o$10o30b10o$10o30b10o$10b11o19b20o10b10o1
01457                 Vec2 htw_dim = {130, 40};
01458                 auto *f = new Figure(htw_string, htw_dim);
01459                 gameOfLife.populate_figure(f,(Vec2){x_pos,y_pos}, angle/90);
01460                 gameOfLife.set_rules(r_copy);
01461                 zoom = gameOfLife.get_grid()->get_auto_zoom_factor();
01462             }
01463             if(example_flags[7]){
01464                 uiOptionsFlag |= UI_Options_Flag_enable_AutoScroll | UI_Options_Flag_enable_AutoZoom;
01465                 Vec2 pacman_dim = {13, 13};
01466                 //Figure *f = new Figure(pacman, pacman_dim);
01467                 std::string pacman_string =
    "5b3o$3b2o3b2o$2bo7bo$bo4b3o2bo$bo4b3o2bo$o11bo$o5b7o$o6bo$bo6bo$bo7bo$2bo7bo$3b2o3b2o$5b3o!";
01468                 auto *f = new Figure(pacman_string, pacman_dim);
01469                 gameOfLife.populate_figure(f,(Vec2){x_pos,y_pos}, angle/90);
01470                 gameOfLife.set_rules(r_copy);
01471                 zoom = gameOfLife.get_grid()->get_auto_zoom_factor();
01472             }
01473 
01474             for(auto c: gameOfLife.get_grid()->get_cells()){
01475                 c->get_neighbours()->set_n_alive(0);
01476                 for(auto n: c->get_neighbours()->get_cells()){
01477                     if(n->get_state()){
    c->get_neighbours()->set_n_alive(c->get_neighbours()->get_n_alive()+1); }
01478                 }
01479                 c->set_color(gameOfLife.get_color_rules()->color[c->get_neighbours()->get_n_alive()]);
01480             }
```

```
01481             reset_figure = Figure(&gameOfLife);
01482             uiFlag &= ~UI_Flag_enable_LoadExampleUI;
01483         }
01484         ImGui::PopStyleColor(3);
01485         ImGui::PopID();
01486         // abort button
01487         ImGui::SameLine();
01488         //color red
01489         ImGui::PushID(0);
01490         ImGui::PushStyleColor(ImGuiCol_Button, (ImVec4)ImColor::HSV(0 / 7.0f, 0.6f, 0.6f));
01491         ImGui::PushStyleColor(ImGuiCol_ButtonHovered, (ImVec4)ImColor::HSV(0 / 7.0f, 0.7f, 0.7f));
01492         ImGui::PushStyleColor(ImGuiCol_ButtonActive, (ImVec4)ImColor::HSV(0 / 7.0f, 0.8f, 0.8f));
01493         if (ImGui::Button("Abort")) {
01494             uiFlag &= ~UI_Flag_enable_LoadExampleUI;
01495         }
01496         ImGui::PopStyleColor(3);
01497         ImGui::PopID();
01498
01499         ImGui::End();
01500     }
```

5.8.2.19 load_UI() void gol::UI::load_UI (UI_Flag & *uiFlag*, UI_Options_Flag & *uiOptionsFlag*, GameOfLife & *gameOfLife*, Figure & *reset_figure*, std::string & *message*, float & *zoom*) [static]

initialize Load user interface

Parameters

uiFlag	: &UI_Flag -> includes UI flags to enable/disable window
uiOptionsFlag	: &UI_Options_Flag -> used to enable auto scroll, auto zoom
gameOfLife	: gol::GameOfLife -> includes grid & cell information
reset_figure	: gol::Figure -> reference required for resetting grid
load_UI	: bool -> load user interface trigger
message_UI	: bool -> message user interface trigger
message	: std::string -> required for error handling

Definition at line 1080 of file game_of_life.cpp.

```
01080
      {
01081         static char path[256] =
      "/Users/user404/Studium/CE27_Softwaretechnik/Game_Of_Life2.0/figures/toggle.lif";
01082         static int x_pos = 0;
01083         static int y_pos = 0;
01084         static int angle = 0;
01085
01086         ImGui::Begin("Load from File", nullptr, 0/*ImGuiWindowFlags_NoMove*/  );
01087         ImGui::Text("enter full filepath");
01088         ImGui::InputText("filepath", path, IM_ARRAYSIZE(path)); //Text input path
01089
01090
01091         ImGui::Text("enter x and y position of figure in grid");
01092         ImGui::InputInt("x", &x_pos); // Textfield x position
01093         if(x_pos > gameOfLife.get_grid()->get_dimension().x){x_pos =
      gameOfLife.get_grid()->get_dimension().x; }
01094         if(x_pos < 0){x_pos = 0;}
01095
01096         ImGui::InputInt("y", &y_pos); // Textfield y position
01097         if(y_pos > gameOfLife.get_grid()->get_dimension().y){ y_pos =
      gameOfLife.get_grid()->get_dimension().y; }
01098         if(y_pos < 0){y_pos = 0;}
01099
01100
```

```
01101         ImGui::Text("enter angle of figure in grid");
01102         ImGui::InputInt("angle", &angle, 90, 90, 0); // Textfield x position
01103         if(angle > 270) {
01104             angle = 270;
01105         }
01106         if(angle < 0){
01107             angle = 0;
01108         }
01109 
01110         // Button load from file
01111         try {
01112             button_load(uiFlag, uiOptionsFlag, gameOfLife, reset_figure, (std::string) path, (Vec2)
      {x_pos, y_pos}, angle, message, zoom);
01113         }catch(std::logic_error &err){
01114             message = (std::string)err.what();
01115             uiFlag |= UI_Flag_enable_MessageUI;
01116         }
01117         // abort button
01118         ImGui::SameLine();
01119         //color red
01120         ImGui::PushID(0);
01121         ImGui::PushStyleColor(ImGuiCol_Button, (ImVec4)ImColor::HSV(0 / 7.0f, 0.6f, 0.6f));
01122         ImGui::PushStyleColor(ImGuiCol_ButtonHovered, (ImVec4)ImColor::HSV(0 / 7.0f, 0.7f, 0.7f));
01123         ImGui::PushStyleColor(ImGuiCol_ButtonActive, (ImVec4)ImColor::HSV(0 / 7.0f, 0.8f, 0.8f));
01124         if (ImGui::Button("Abort")) {
01125             uiFlag &= ~UI_Flag_enable_LoadUI;
01126         }
01127         ImGui::PopStyleColor(3);
01128         ImGui::PopID();
01129 
01130         ImGui::End();
01131 
01132         //TODO add some standard figures
01133     }
```

5.8.2.20 messageBox_UI()

```
void gol::UI::messageBox_UI (
        UI_Flag & uiFlag,
        std::string & message )  [static]
```

initialize MessageBox user interface

Parameters

uiFlag	: &UI_Flag -> includes UI flags to enable/disable window
message	: std::string -> message to display

Definition at line 1163 of file game_of_life.cpp.

```
01163                                                                   {
01164         ImGui::Begin("Message");
01165 
01166         ImGui::Text("%s", message.c_str());
01167 
01168         if (ImGui::Button("OK")) {
01169             uiFlag &= ~UI_Flag_enable_MessageUI;
01170             message.clear();
01171         }
01172 
01173         ImGui::End();
01174     }
```

5.8.2.21 print()

```
void gol::UI::print (
        GameOfLife * _gameOfLife,
        float zoom )  [static], [private]
```

print rectangles with size of zoom << used by gol::UI::grid_UI() >>

Parameters

_gameOfLife	: &GameOfLife -> includes grid & cell information
zoom	: float -> zoom factor

Definition at line 717 of file game_of_life.cpp.

```
00717                                                   {
00718
00719         ImVec2 p = ImGui::GetCursorScreenPos();
00720
00721         // Create new child for scrolling
00722         ImVec2 scrolling_child_size = ImVec2(ImGui::GetFrameHeight() +
      (float)_gameOfLife->get_grid()->get_dimension().x*zoom,
      (float)_gameOfLife->get_grid()->get_dimension().y*zoom);
00723         ImGui::BeginChild("scrolling", scrolling_child_size, true,
      ImGuiWindowFlags_HorizontalScrollbar);
00724         // create for every living cell a white rectangle
00725         for(int y = 0; y < _gameOfLife->get_grid()->get_dimension().y; y++){
00726             for(int x = 0; x < _gameOfLife->get_grid()->get_dimension().x; x++){
00727                 if(_gameOfLife->get_grid()->get_cell((Vec2){x,y})->get_state()){
00728                     ImGui::GetWindowDrawList()->AddRectFilled(ImVec2(p.x+((float)x*zoom)+1,
      p.y+((float)y*zoom)+1), ImVec2(p.x+((float)x*zoom)+zoom, p.y+((float)y*zoom)+zoom),
      _gameOfLife->get_grid()->get_cell((Vec2){x,y})->get_color());
00729                     //ImGui::GetWindowDrawList()->AddRectFilled(ImVec2(p.x+(x*zoom), p.y+(y*zoom)),
      ImVec2(p.x+(x*zoom)+zoom, p.y+(y*zoom)+zoom),
      _gameOfLife->get_grid()->get_cell((Vec2){x,y})->get_color());
00730
00731                 }
00732             }
00733         }
00734
00735
00736         ImGui::EndChild();
00737     }
```

5.8.2.22 screenshot_UI()

```
void gol::UI::screenshot_UI (
        UI_Flag & uiFlag,
        GameOfLife & gameOfLife,
        std::string & message )  [static]
```

initialize Screenshot user interface

Parameters

uiFlag	: &UI_Flag -> includes save_UI flags to enable/disable window
gameOfLife	: gol::GameOfLife -> includes grid & cell information
message	: std::string -> required for status messages

Definition at line 1134 of file game_of_life.cpp.

```
01134                                                                                   {
01135         static char path[256] =
      "/Users/user404/Studium/CE27_Softwaretechnik/PROTO_GOL/test_Screenshot.lif";
01136
01137         ImGui::Begin("Save to File", nullptr, 0/*ImGuiWindowFlags_NoMove*/);
01138         ImGui::Text("enter full filepath");
01139         ImGui::InputText("filepath", path, IM_ARRAYSIZE(path)); //Text input path
01140
01141         try {
01142             button_save(uiFlag,gameOfLife, path);
01143         }catch(std::logic_error &err){
01144             message = (std::string)err.what();
01145             uiFlag |= UI_Flag_enable_MessageUI;
01146         }
01147         // abort button
01148         ImGui::SameLine();
01149         ImGui::PushID(0);
01150         //color red
```

```
01151         ImGui::PushStyleColor(ImGuiCol_Button, (ImVec4)ImColor::HSV(0 / 7.0f, 0.6f, 0.6f));
01152         ImGui::PushStyleColor(ImGuiCol_ButtonHovered, (ImVec4)ImColor::HSV(0 / 7.0f, 0.7f, 0.7f));
01153         ImGui::PushStyleColor(ImGuiCol_ButtonActive, (ImVec4)ImColor::HSV(0 / 7.0f, 0.8f, 0.8f));
01154         if (ImGui::Button("Abort")) {
01155             uiFlag &= ~UI_Flag_enable_SaveUI;
01156         }
01157         ImGui::PopStyleColor(3);
01158         ImGui::PopID();
01159 
01160 
01161         ImGui::End();
01162     }
```

5.8.2.23 setting_UI() void gol::UI::setting_UI (UI_Flag & *uiFlag*, UI_Options_Flag & *uiOptionsFlag*, GameOfLife & *gameOfLife*, Figure & *reset_figure*, Figure_Stack & *figureStack*, float & *zoom*, int & *speed*) [static]

initialize Setting user interface

Parameters

uiFlag	: &UI_Flag -> includes UI flags to enable/disable window
uiOptionsFlag	: &UI_Options_Flag -> used to enable auto scroll, auto zoom and run
gameOfLife	: gol::GameOfLife -> includes grid & cell information
reset_figure	: gol::Figure -> reference required for resetting grid
zoom	: float -> zoom factor
speed	: int -> speed factor
run	: bool -> run Game of Life trigger
load_UI	: bool -> load user interface trigger
save_UI	: bool -> save user interface trigger

Definition at line 959 of file game_of_life.cpp.

```
00959 
      {
00960 
00961         ImGui::Begin("Settings", nullptr, /*ImGuiWindowFlags_NoMove |*/
      ImGuiWindowFlags_NoDecoration);
00962 
00963         //Autoscroll Checkbox Begin
00964         bool autoScroll;
00965         if(uiOptionsFlag & UI_Options_Flag_enable_AutoScroll){ autoScroll = true; }else{ autoScroll =
      false; }
00966         ImGui::Checkbox("Auto Scroll", &autoScroll);
00967         if(autoScroll){ uiOptionsFlag |= UI_Options_Flag_enable_AutoScroll;}else{uiOptionsFlag &=
      ~UI_Options_Flag_enable_AutoScroll;}
00968         //Autoscroll Checkbox End
00969 
00970         UI::button_setting_zoom(uiOptionsFlag, zoom);      // create zoom button
00971 
00972         UI::button_setting_speed(speed);    // create speed button
00973         UI::button_setting_step(gameOfLife, figureStack); // create step button
00974         ImGui::Spacing();
00975         UI::button_setting_run(uiOptionsFlag);        // run game of life button
00976         //ImGui::SameLine();
00977         //UI::button_setting_stop(uiOptionsFlag);       // stop game of life button
00978 
00979         ImGui::SameLine();
00980         UI::button_setting_reset(gameOfLife, reset_figure, figureStack); // reset game of life button
00981 
00982         ImGui::SameLine();
```

```
00983         UI::button_setting_clear(gameOfLife, reset_figure, figureStack); // clear game of life button
00984
00985         ImGui::Spacing();
00986         UI::button_setting_random(gameOfLife, reset_figure, figureStack); // random button
00987
00988         UI::button_setting_load(uiFlag);   // load from file button
00989         UI::button_setting_load_exp(uiFlag);
00990         UI::button_setting_screenshot(uiFlag);
00991         ImGui::Spacing();
00992
00993         UI::setup_rules(uiOptionsFlag, gameOfLife);    // rules setting checkboxes
00994
00995         ImGui::End();
00996
00997     }
```

5.8.2.24 setup_rules() `void gol::UI::setup_rules (`

`UI_Options_Flag & uiOptionsFlag,`

`GameOfLife & gameOfLife ) [static], [private]`

create checkboxes for rule and color modifications << used by gol::UI::setting_UI() >>

Parameters

uiOptionsFlag	: &UI_Options_Flag -> includes color all option flag
gameOfLife	: &GameOfLife -> includes grid & cell information

Definition at line 1209 of file game_of_life.cpp.

```
01209                                                                                   {
01210         ImGui::Text("Rules"); //ImGui::SameLine();
01211         for(int i=0; i < 9; i++) {
01212             std::string label_a = std::to_string(i);
01213             label_a += " Alive";
01214             std::string label_d = std::to_string(i);
01215             label_d += " Dead";
01216
01217             bool d = gameOfLife.get_rules()->death[i];
01218             bool a = gameOfLife.get_rules()->alive[i];
01219
01220             ImGui::Checkbox(label_a.c_str(), &gameOfLife.get_rules()->alive[i]);
01221             ImGui::SameLine();
01222             ImGui::Checkbox(label_d.c_str(), &gameOfLife.get_rules()->death[i]);
01223             if (((gameOfLife.get_rules()->alive[i])) && (gameOfLife.get_rules()->death[i])) {
01224                 if (d && !a) {
01225                     gameOfLife.get_rules()->death[i] = false;
01226                     gameOfLife.get_rules()->alive[i] = true;
01227                 } else {
01228                     gameOfLife.get_rules()->death[i] = true;
01229                     gameOfLife.get_rules()->alive[i] = false;
01230                 }
01231
01232             }
01233
01234
01235             // Color Picker
01236             if (!(uiOptionsFlag & UI_Options_Flag_enable_ColorizeAll)) {
01237                 ImGui::SameLine();
01238                 char index[2] = {char(i + 0x30), 0x00};
01239                 ColorPicker(gameOfLife, index, &gameOfLife.get_color_rules()->color[i]);
01240             }
01241         }
01242         button_setting_reset_colors(gameOfLife.get_color_rules());
01243         static bool color_all;
01244         if(uiOptionsFlag & UI_Options_Flag_enable_ColorizeAll){color_all = true; }else{color_all =
     false;}
01245         ImGui::Checkbox("color all: ", &color_all);
01246         if(color_all){uiOptionsFlag |= UI_Options_Flag_enable_ColorizeAll;}else{uiOptionsFlag &=
     ~UI_Options_Flag_enable_ColorizeAll; }
01247         if(uiOptionsFlag & UI_Options_Flag_enable_ColorizeAll) {
01248             ImGui::SameLine();
01249             ImU32 &color_a = gameOfLife.get_color_rules()->color[0];
01250
01251             ColorPicker(gameOfLife, " ", &color_a);
```

```
01252             for(unsigned int & i : gameOfLife.get_color_rules()->color){
01253
01254                 i = color_a;
01255             }
01256
01257         }
01258         button_setting_reset_rules(gameOfLife.get_rules());
01259     }
```

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

- game_of_life.h
- game_of_life.cpp

5.9 gol::Vec2 Struct Reference

structure defines a 2 dimensional Vector

`#include "game_of_life.h"`

Data Fields

- int x
- int y

5.9.1 Detailed Description

structure defines a 2 dimensional Vector

Definition at line 88 of file game_of_life.h.

5.9.2 Field Documentation

5.9.2.1 x `int gol::Vec2::x`

Definition at line 88 of file game_of_life.h.

5.9.2.2 y `int gol::Vec2::y`

Definition at line 88 of file game_of_life.h.

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

- game_of_life.h

6 File Documentation

6.1 game_of_life.cpp File Reference

defines a Cell (smallest element in grid)

```
#include "game_of_life.h"
#include <cmath>
#include <utility>
```

Namespaces

- namespace gol

6.1.1 Detailed Description

defines a Cell (smallest element in grid)

object which includes all Buttons an UI functions << used by gol::GameOfLife::run() >>

object which include a grid and is used to run Game Of Life with user interface

object which define a Figure with cell states

object define all neighbour Cells of owner cells

object define a Grid which includes x∗y cells

is used to load into grid

Definition in file game_of_life.cpp.

6.2 game_of_life.cpp

Go to the documentation of this file.

```
00001 //
00002 // Created by
00003 //          Benjamin Grothe s0580413
00004 //          Juan Jose Arguello Guerra s0580592
00005 // on 06.05.23
00006 //
00007
00008 #include "game_of_life.h"
00009 #include <cmath>
00010 #include <utility>
00011
00012 namespace gol {
00013
      //=============================================================================================
      CLASS CELL ===//
00014     Cell::Cell(Grid *_grid, Vec2 _position) {
00015         grid = _grid;
00016         position = _position;
00017         state = false;              // initialize Cell object as death
00018         color = IM_COL32_WHITE;     // initialize Cell color as white
00019     }
00020
      //=============================================================================================
      CLASS GRID ===//
00021     Grid::Grid(GameOfLife *_gameOfLife, Vec2 _dimension) {
```

```
00022         gameOfLife = _gameOfLife;
00023         dimension = _dimension;
00024 
00025         // initialize Grid object with x * y death Cell objects
00026         for(int y = 0; y < dimension.y; y++){
00027             for(int x = 0; x < dimension.x; x++){
00028                 cells.push_back(new Cell(this, (Vec2){x,y}));
00029             }
00030         }
00031 
00032         // create for each Cell object in Grid object a Neighbours object
00033         for(int y = 0; y < dimension.y; y++){
00034             for(int x = 0; x < dimension.x; x++){
00035                 new Neighbours(cells[(y*dimension.x)+x]);
00036             }
00037         }
00038     }
00039 
00040     Vec2 Grid::get_real_grid_dimension() {
00041         int first_row  = 0; bool first_row_flag = false;
00042         int last_row   = 0;
00043 
00044         // filter first and last row
00045         for(auto c: cells){
00046             if(c->get_state()){
00047                 if(!first_row_flag){
00048                     first_row_flag = true;
00049                     first_row = c->get_position().y;
00050                     last_row = c->get_position().y;
00051                 }else{
00052                     last_row = c->get_position().y;
00053                 }
00054             }
00055         }
00056 
00057         int first_column  = 0; bool first_column_flag = false;
00058         int last_column   = 0;
00059         // filter first and last column
00060         for(int x=0; x < dimension.x; x++){
00061             for(int y=0; y < dimension.y; y++){
00062                 if(cells[(y*dimension.x)+x]->get_state()){
00063                     if(!first_column_flag){
00064                         first_column_flag = true;
00065                         first_column = cells[(y*dimension.x)+x]->get_position().x;
00066                         last_column = cells[(y*dimension.x)+x]->get_position().x;
00067                     }else{
00068                         last_column = cells[(y*dimension.x)+x]->get_position().x;
00069                     }
00070                 }
00071             }
00072         }
00073 
00074         int y = last_row-first_row+1;
00075         int x = last_column-first_column+1;
00076 
00077 
00078         return Vec2{x,y};
00079     }
00080     float Grid::get_auto_zoom_factor() {
00081         float d = 1;
00082         if(get_real_grid_dimension().x > get_real_grid_dimension().y){ d = (float)dimension.x /
    (float)get_real_grid_dimension().x;}else{
00083             d = (float)dimension.y/(float)get_real_grid_dimension().y;
00084         }
00085         if(d <= 2.f){ return 2.f; }
00086         return d;
00087     }
00088     Vec2 Grid::get_real_grid_position() {
00089         //TODO get center of real grid dimension
00090 
00091         //TODO first living cell in row
00092         Vec2 position_A = Vec2{0,0};
00093         Vec2 position = Vec2{0,0};
00094         for(auto c: cells){
00095             if(c->get_state()) {
00096                 position_A.y = c->get_position().y;
00097                 break;
00098             }
00099 
00100         }
00101 
00102         // TODO first living cell in column
00103         bool b = false;
00104         for(int x=0; x < dimension.x; x++){
00105             for(int y=0; y < dimension.y; y++){
00106                 if(cells[(y*dimension.x)+x]->get_state()) {
00107                     position_A.x = cells[(y*dimension.x)+x]->get_position().x;
```

```
00108                     b = true;
00109                     break;
00110                 }
00111             }
00112             if(b){break;}
00113         }
00114         position.x = position_A.x;// + (get_real_grid_dimension().x/2);
00115         position.y = position_A.y;// + (get_real_grid_dimension().y/2);
00116 
00117         return position; // Center of figure
00118     }
00119     float Grid::get_scrollx_position() {
00120         float x = (float)get_real_grid_position().x;//+((float)get_real_grid_dimension().x/2);
00121         auto x_max = (float)dimension.x;
00122         float real_x = ((float)get_real_grid_dimension().x+2)*x/x_max;
00123         return (x+real_x)/x_max;
00124     }
00125     float Grid::get_scrolly_position() {
00126 
00127         float y = (float)get_real_grid_position().y;//+((float)get_real_grid_dimension().y/2);
00128         auto y_max = (float)dimension.y;
00129         float real_y = ((float)get_real_grid_dimension().y+1)*y/y_max;
00130         return (y+real_y)/y_max;
00131     }
00132 
00133     //=============================================================================================
      CLASS NEIGHBOURS ===//
00134     Neighbours::Neighbours(Cell *_owner) {
00135         owner = _owner;
00136 
00137         owner->set_neighbours(this);        // associate this neighbour to owner Cell
00138 
00139         Vec2 pos_o = owner->get_position(); // position owner
00140         Grid* grid = owner->get_grid();     // grid
00141 
00142         // add all cells next to this object
00143         for(int y= -1; y <= 1; y++){
00144             for(int x= -1; x <= 1; x++){
00145 
00146                 // position of next_to cell
00147                 Vec2 pos_n = (Vec2){pos_o.x+x,pos_o.y + y};
00148                 // check x position of next_to cell
00149                 if( (pos_n.x >= 0) && (pos_n.x < owner->get_grid()->get_dimension().x) ){
00150                     // check y position of next_to cell
00151                     if( (pos_n.y >= 0) && (pos_n.y < owner->get_grid()->get_dimension().y) ){
00152                         // filter owner cell object
00153                         if((pos_o.x != pos_n.x) || (pos_o.y != pos_n.y)){
00154                             cells.push_back(grid->get_cell(pos_n));
00155                         }
00156                     }
00157                 }
00158             }
00159         }
00160 
00161     }
00162     //=============================================================================================
      CLASS FIGURE ===//
00163     Figure::Figure(std::string &path) {
00164         std::fstream file;
00165 
00166         std::string code;   // whole code as string
00167         std::string dim;    // dimension as string
00168 
00169 
00170         file.open(path);
00171             // check if file exist else throw std::logic_error
00172         if(!file.is_open()){ throw std::logic_error("ERROR: could not load file"); }
00173 
00174         while(file){
00175 
00176             std::string row;
00177             // read line in file and write line into row
00178             std::getline(file, row);
00179             // if line starts with a number, 'b', 'o' , '$' or '!' write line into code
00180             // if line starts with '#' ignore line
00181             // else write line into dim 'dimension'
00182             if(((row[0] >= 0x30) && (row[0] <= 0x39)) || (row[0] == 'b') || (row[0] == 'o') || (row[0]
      == '$') || ((row[0] == '!'))){
00183                 for(auto c: row){
00184                     code.push_back(c);
00185                 }
00186             }else if(row[0] != '#'){
00187                 for(auto c : row){
00188                     dim.push_back(c);
00189                 }
00190             }
00191         }
```

```
00192
00193         // get dimension of figure from dim
00194         dimensions = convert_string_to_dimensions(dim);
00195         // fill figure_table with converted code
00196         states = convert_string_to_states(code);
00197
00198
00199     }
00200     Figure::Figure(GameOfLife *gameOfLife) {
00201         // get dimensions from GameOfLife object
00202         dimensions = gameOfLife->get_grid()->get_dimension();
00203         // get every cell state in grid
00204         for(auto c: gameOfLife->get_grid()->get_cells()){
00205             states.push_back(c->get_state());
00206         }
00207     }
00208
00209     Vec2 Figure::convert_string_to_dimensions(std::string str) {
00210         std::string x, y;
00211
00212         int comma = 0;
00213
00214         for(auto c: str){
00215
00216             // filter numbers
00217             if((c >= 0x30) && (c <= 0x39)){
00218                 if(comma == 0){ x.push_back(c); }
00219                 if(comma == 1){ y.push_back(c); }
00220             }
00221             // filter comma
00222             if(c == ','){ comma++; }
00223             if(comma > 1){ throw std::logic_error("ERROR: there are to much comma's in dimension
      line"); }
00224         }
00225
00226         return (Vec2){std::stoi(x), std::stoi(y)};
00227     }
00228     std::string Figure::convert_dimension_to_string() {
00229         std::string str_dim;
00230
00231         int x = 0;
00232         int y = 0;
00233
00234         // check real row of figure
00235         int t_y = 0;
00236         for(int _y = 0; _y < dimensions.y; _y++){
00237             int t_x = 0;
00238             for(int _x = 0; _x < dimensions.y; _x++){
00239                 if(states[(_y*dimensions.x)+_x]){t_x = _x;}
00240             }
00241
00242             if(x < t_x){ x = t_x; }
00243             if(t_x > 0){ y = _y; }
00244             if((t_y == 0) && (_y > t_y) && (t_x > 0)){
00245                 t_y = _y;
00246             }
00247         }
00248
00249         int column_f = 0;
00250         bool b = false;
00251         for(int _x=0; _x < dimensions.x; _x++){
00252             for(int _y=0; _y < dimensions.y; _y++){
00253                 if(states[(_y*dimensions.x+_x)]){ b = true; }
00254             }
00255             column_f = _x;
00256             if(b){break;}
00257
00258         }
00259         // convert real dimensions into a string
00260         str_dim += "x=" + std::to_string(x-column_f+1) + " , y=" + std::to_string(y-t_y+2);
00261
00262
00263         return str_dim;
00264     }
00265
00266     std::vector<bool> Figure::convert_string_to_states(std::string str) {
00267         std::vector<bool> _states;
00268         std::string _num;
00269         int x = 0;
00270         int y = 0;
00271
00272         // prepare _states for the right dimensions
00273         for(int i=0; i < (dimensions.x * dimensions.y); i++){
00274             _states.push_back(false);
00275         }
00276
00277         // check each character in string
```

```
00278         for(auto c: str){
00279             // if character is a number add to _num buffer
00280             if((c >=0x30) && (c <= 0x39)){ _num.push_back(c); }
00281             // if character is a 'b' set state on position [x;y](*_num) to false and increment x
00282             if(c == 'b'){
00283                 if(_num.empty()){ _num.push_back('1'); }
00284                 for(int b=0; b < std::stoi(_num); b++){
00285                     _states[(y *dimensions.x)+(x)]=false;
00286                     x++;
00287                 }
00288                 // reset _num buffer
00289                 _num.clear();
00290             }
00291             // if character is a 'o' set state on position [x;y](*_num) to true and increment x
00292             if(c == 'o'){
00293                 if(_num.empty()){ _num.push_back('1'); }
00294                 for(int o=0; o < std::stoi(_num); o++){
00295                     _states[(y*dimensions.x)+(x)]=true;
00296                     x++;
00297                 }
00298                 // reset _num buffer
00299                 _num.clear();
00300             }
00301             // if character is a '$' increment y; reset _num buffer; set x to 0
00302             if(c == '$'){
00303                 if(_num.empty()){ _num.push_back('1');}
00304                 for(int r=0; r < std::stoi(_num); r++){
00305                     y++;
00306                 }
00307                 _num.clear();
00308                 x=0;
00309             }
00310             // EOF "End Of File"
00311             if(c == '!'){ break; }
00312         }
00313         return _states;
00314     }
00315     std::string Figure::convert_states_to_string() {
00316         std::string str_states;
00317
00318         // convert states to string
00319         for(int y=0; y < dimensions.y; y++){
00320             for(int x=0; x < dimensions.x; x++){
00321                 if(states[(y*dimensions.x)+x]){ str_states += 'o'; }else{ str_states += 'b'; }
00322             }
00323             str_states += '$';
00324         }
00325         // count first empty columns
00326         int column_f = 0;
00327         bool b = false;
00328         for(int _x=0; _x < dimensions.x; _x++){
00329             for(int _y=0; _y < dimensions.y; _y++){
00330                 if(states[(_y*dimensions.x+_x)]){ b = true; }
00331             }
00332             column_f = _x;
00333             if(b){break;}
00334
00335         }
00336         // filter first columns
00337         b = true;
00338         int counter = 0;
00339         std::string _str_states;
00340         for(auto c: str_states){
00341             if(c == '$'){ _str_states.push_back(c); b = true;}else
00342             if(b == true){
00343                 counter++;
00344                 if(counter >= column_f){ b = false; counter = 0;}
00345             }else
00346             { _str_states.push_back(c);}
00347
00348         }
00349
00350         str_states = _str_states;
00351
00352         // filter last death columns
00353         counter = 0;
00354         char last = 0x00;
00355         _str_states.clear();
00356         for(auto c: str_states){
00357             if(last == 0x00){last = c;}
00358             if(c == last){ counter++;}
00359             if(c != last){
00360                 if((c == '$') && (last == 'b')){ counter = 1; last = 0x00; }
00361                 if(counter > 1){ _str_states += std::to_string(counter) + last; counter = 1; } else
    {_str_states += last;}
00362                 last = c;
00363             }
```

```
00364         }
00365
00366         int counter_f = 0; // counter first empty rows
00367         int counter_l = 0; // counter last empty rows
00368         b = false;    // first row marker
00369         for(auto c: _str_states){
00370             if((c=='$')&&(b == false)){ counter_f+=2; } //count first empty rows
00371             if((c=='$')&&(b == true)){ counter_l+=2; }  //count last empty rows
00372             if((c == 'o') || (c == 'b')){counter_l = 0; b = 1; }
00373         }
00374         // delete first empty rows
00375         _str_states.erase(0,counter_f);
00376
00377         // delete last empty rows
00378         while(counter_l > 0){
00379             _str_states.pop_back();
00380             counter_l--;
00381         }
00382         _str_states += '!';
00383
00384         str_states.clear();
00385         for(auto c: _str_states){
00386             if(c != 0x00){str_states.push_back(c);}
00387         }
00388         return str_states;
00389     }
00390
00391     int Figure::save_to_file(std::string &path) {
00392         std::fstream file;
00393
00394         // create file and open
00395         file.open(path , std::ios_base::out);
00396         // Error Handling
00397         if(!file.is_open()){ throw std::logic_error("ERROR: failed"); }
00398
00399         file « convert_dimension_to_string() « "\n";
00400         file « convert_states_to_string();
00401
00402         // close file
00403         file.close();
00404
00405         return 0;
00406     }
00407     //=========================================================================================== CLASS
      GAME OF LIFE ===//
00408     GameOfLife::GameOfLife(Vec2 _dimensions, Rule _rules, RuleColor _rulesColor) {
00409         rules = _rules;                                    // initialize rules
00410         grid = new Grid(this, _dimensions); // initialize new grid
00411         rules_color = _rulesColor;                         // initialize color Rules
00412     }
00413
00414     bool GameOfLife::check_rules_in_cell(Cell *_cell) {
00415         int counter = 0;
00416         // count every true cell in neighbours
00417         for(auto n: _cell->get_neighbours()->get_cells()){
00418             if(n->get_state()){ counter++; }
00419         }
00420         // check rules
00421         if (rules.alive[counter]){ return true; }
00422         if (rules.death[counter]){ return false; }
00423
00424
00425
00426         // returns new cell state
00427         return _cell->get_state();
00428     }
00429     void GameOfLife::check_rules_in_grid() {
00430         std::vector<bool> _states;  // buffer for cell states
00431
00432         // check rules for every cell in grid and push result into state buffer
00433         for(auto c: grid->get_cells()){
00434             _states.push_back(check_rules_in_cell(c));
00435         }
00436         // write new state from buffer to cells in grid
00437         for(auto c: grid->get_cells()){
00438             c->set_state(_states[(c->get_position().y*grid->get_dimension().x)+c->get_position().x]);
00439         }
00440
00441         // change color
00442         for(auto c: grid->get_cells()){
00443             c->get_neighbours()->set_n_alive(0);
00444             for(auto n: c->get_neighbours()->get_cells()){
00445                 if(n->get_state()){
      c->get_neighbours()->set_n_alive(c->get_neighbours()->get_n_alive()+1); }
00446             }
00447             c->set_color(rules_color.color[c->get_neighbours()->get_n_alive()]);
00448         }
```

```
00449
00450     }
00451
00452     void GameOfLife::clear_grid() {
00453         // set every cell state in grit to false
00454         for(auto c: grid->get_cells()){
00455             c->set_state(false);
00456         }
00457     }
00458     void GameOfLife::populate_figure(Figure *_figure, Vec2 position, int angle) {
00459         int x_c = 0;
00460
00461         // Error Handler
00462         if((angle < 0)||(angle > 3)){throw std::logic_error("ERROR wrong angle in
      rotate_and_populate_figure()"); }
00463         // no roation
00464         if(angle == 0){
00465             // Error handling
00466             if(_figure->get_dimension().x+position.x > grid->get_dimension().x){throw
      std::logic_error("ERROR: figure dimension x is larger than grid dimension x");}
00467             if(_figure->get_dimension().y+position.y > grid->get_dimension().y){throw
      std::logic_error("ERROR: figure dimension y is larger than grid dimension y");}
00468             // insert figure states into grid
00469             for(int _y=0; _y < _figure->get_dimension().y; _y++){
00470                 for(int _x=0; _x < _figure->get_dimension().x; _x++){
00471                     grid->get_cell((Vec2){_x + position.x, _y +
      position.y})->set_state(_figure->get_state((Vec2){_x, _y}));
00472                 }
00473             }
00474         }
00475
00476         // rotate 180°
00477         if(angle == 2){
00478             // Error handling
00479             if(_figure->get_dimension().x > grid->get_dimension().x){throw std::logic_error("ERROR:
      figure dimension x is larger than grid dimension x");}
00480             if(_figure->get_dimension().y > grid->get_dimension().y){throw std::logic_error("ERROR:
      figure dimension y is larger than grid dimension y");}
00481             // insert figure states into grid
00482             for(int _y=_figure->get_dimension().y-1; _y >= 0; _y--){
00483                 int y_c = 0;
00484                 for(int _x=_figure->get_dimension().x-1; _x >= 0; _x--){
00485                     grid->get_cell((Vec2){y_c + position.x, x_c +
      position.y})->set_state(_figure->get_state((Vec2){_x, _y}));
00486                     y_c++;
00487                 }
00488                 x_c++;
00489             }
00490         }
00491
00492         // rotate 90°
00493         if(angle == 1){
00494             // Error Handling
00495             if(_figure->get_dimension().x > grid->get_dimension().y){throw std::logic_error("ERROR:
      figure dimension x is larger than grid dimension y");}
00496             if(_figure->get_dimension().y > grid->get_dimension().x){throw std::logic_error("ERROR:
      figure dimension y is larger than grid dimension x");}
00497             // insert figure states into grid
00498             for(int _x=0; _x < _figure->get_dimension().x; _x++){
00499                 int y_c = 0;
00500                 for(int _y=_figure->get_dimension().y-1; _y >= 0; _y--){
00501                     grid->get_cell((Vec2){y_c + position.x, x_c +
      position.y})->set_state(_figure->get_state((Vec2){_x,_y}));
00502                     y_c++;
00503                 }
00504                 x_c++;
00505             }
00506         }
00507
00508         // rotate 270°
00509         if(angle == 3){
00510             // Error Handling
00511             if(_figure->get_dimension().x > grid->get_dimension().y){throw std::logic_error("ERROR:
      figure dimension x is larger than grid dimension y");}
00512             if(_figure->get_dimension().y > grid->get_dimension().x){throw std::logic_error("ERROR:
      figure dimension y is larger than grid dimension x");}
00513             // insert figure states into grid
00514             for(int _x=_figure->get_dimension().x-1; _x >= 0; _x--){
00515                 int y_c = 0;
00516                 for(int _y=0; _y < _figure->get_dimension().y; _y++){
00517                     grid->get_cell((Vec2){y_c + position.x, x_c +
      position.y})->set_state(_figure->get_state((Vec2){_x, _y}));
00518                     y_c++;
00519                 }
00520                 x_c++;
00521             }
00522         }
```

```
00523     }
00524
00525     bool GameOfLife::random_bool() {
00526             if (rand() % 2 == 0){return true;}
00527             return false;
00528     }
00529     void GameOfLife::populate_random() {
00530         for(int _y=0; _y < grid->get_dimension().y; _y++){
00531             for(int _x=0; _x < grid->get_dimension().x; _x++){
00532                 grid->get_cell((Vec2){_x, _y})->set_state(random_bool());
00533             }
00534         }
00535         // Colorize Grid
00536         for(auto c: grid->get_cells()){
00537             c->get_neighbours()->set_n_alive(0);
00538             for(auto n: c->get_neighbours()->get_cells()){
00539                 if(n->get_state()){
      c->get_neighbours()->set_n_alive(c->get_neighbours()->get_n_alive()+1); }
00540             }
00541             c->set_color(rules_color.color[c->get_neighbours()->get_n_alive()]);
00542         }
00543     }
00544     int GameOfLife::run() {
00545
00546
00547         // Setup SDL // https://github.com/ocornut/imgui.git
00548         if (SDL_Init(SDL_INIT_VIDEO | SDL_INIT_TIMER | SDL_INIT_GAMECONTROLLER) != 0)
00549         {
00550             printf("Error: %s\n", SDL_GetError());
00551             return -1;
00552         }
00553
00554
00555         // From 2.0.18: Enable native IME. // https://github.com/ocornut/imgui.git
00556         #ifdef SDL_HINT_IME_SHOW_UI
00557         SDL_SetHint(SDL_HINT_IME_SHOW_UI, "1");
00558         #endif
00559
00560
00561         // Setup window // https://github.com/ocornut/imgui.git
00562         SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);
00563         SDL_GL_SetAttribute(SDL_GL_DEPTH_SIZE, 24);
00564         SDL_GL_SetAttribute(SDL_GL_STENCIL_SIZE, 8);
00565         SDL_GL_SetAttribute(SDL_GL_CONTEXT_MAJOR_VERSION, 2);
00566         SDL_GL_SetAttribute(SDL_GL_CONTEXT_MINOR_VERSION, 2);
00567         SDL_WindowFlags window_flags = (SDL_WindowFlags)(SDL_WINDOW_OPENGL | SDL_WINDOW_RESIZABLE |
      SDL_WINDOW_ALLOW_HIGHDPI);
00568         SDL_Window* window = SDL_CreateWindow("Cornway's Game Of Life", SDL_WINDOWPOS_CENTERED,
      SDL_WINDOWPOS_CENTERED, 1280, 720, window_flags);
00569         SDL_GLContext gl_context = SDL_GL_CreateContext(window);
00570
00571         SDL_GL_MakeCurrent(window, gl_context);
00572         SDL_GL_SetSwapInterval(1); // Enable vsync
00573
00574         // Setup Dear ImGui context // https://github.com/ocornut/imgui.git
00575         IMGUI_CHECKVERSION();
00576         ImGui::CreateContext();
00577
00578         ImGuiIO& io = ImGui::GetIO(); (void)io;
00579         io.ConfigFlags |= ImGuiConfigFlags_NavEnableKeyboard;     // Enable Keyboard Controls
00580         io.ConfigFlags |= ImGuiConfigFlags_NavEnableGamepad;      // Enable Gamepad Controls
00581         io.ConfigFlags |= ImGuiConfigFlags_DockingEnable;         // Enable Docking
00582         io.ConfigFlags |= ImGuiConfigFlags_ViewportsEnable;       // Enable Multi-Viewport / Platform
      Windows
00583         //io.ConfigViewportsNoAutoMerge = true;
00584         //io.ConfigViewportsNoTaskBarIcon = true;
00585
00586         // Setup Dear ImGui style
00587         ImGui::StyleColorsDark();
00588         //ImGui::StyleColorsLight();
00589
00590         // When viewports are enabled we tweak WindowRounding/WindowBg so platform windows can look
      identical to regular ones. // https://github.com/ocornut/imgui.git
00591         ImGuiStyle& style = ImGui::GetStyle();
00592         if (io.ConfigFlags & ImGuiConfigFlags_ViewportsEnable)
00593         {
00594             style.WindowRounding = 0.0f;
00595             style.Colors[ImGuiCol_WindowBg].w = 1.0f;
00596             style.ChildBorderSize = 0.0f;
00597         }
00598
00599         // Setup Platform/Renderer backends // https://github.com/ocornut/imgui.git
00600         ImGui_ImplSDL2_InitForOpenGL(window, gl_context);
00601         ImGui_ImplOpenGL2_Init();
00602
00603         ImVec4 clear_color = ImVec4(0.0f, 0.0f, 0.0f, 1.00f);
00604         // Main loop
```

```
00605         bool done = false;
00606         // -------------------------------------------------------------------------
00607         // USER CODE GLOBAL BEGIN -------------------------------------------------------
00608         UI_Flag uiFlag = UI_Flag_none;
00609         uiFlag |= UI_Flag_enable_GridUI | UI_Flag_enable_SetupUI;
00610 
00611         UI_Options_Flag uiOptFlag = UI_Options_Flag_none;
00612 
00613         std::string message;                                    // message for Message User Interface
00614 
00615         float zoom = 2.f;                                           // zoom variable
00616         int speed = 0, count=0;                                 // speed variables
00617 
00618         Figure* reset_figure = new Figure(this);     // Reset figure object
00619         Figure_Stack stack = {};                                // Stack needed for step back -> max
      size defined ba STACK_SIZE
00620 
00621         std::string p;                                          // filepath as string
00622 
00623         // USER CODE GLOBAL END ---------------------------------------------------------
00624         // -------------------------------------------------------------------------
00625         // main loop begin
00626         while (!done) {
00627 
00628             // Poll and handle events (inputs, window resize, etc.) //
      https://github.com/ocornut/imgui.git
00629             // You can read the io.WantCaptureMouse, io.WantCaptureKeyboard flags to tell if dear
      imgui wants to use your inputs.
00630             // - When io.WantCaptureMouse is true, do not dispatch mouse input data to your main
      application, or clear/overwrite your copy of the mouse data.
00631             // - When io.WantCaptureKeyboard is true, do not dispatch keyboard input data to your main
      application, or clear/overwrite your copy of the keyboard data.
00632             // Generally you may always pass all inputs to dear imgui, and hide them from your
      application based on those two flags.
00633             SDL_Event event;
00634             while (SDL_PollEvent(&event))
00635             {
00636                 ImGui_ImplSDL2_ProcessEvent(&event);
00637                 if (event.type == SDL_QUIT)
00638                     done = true;
00639                 if (event.type == SDL_WINDOWEVENT && event.window.event == SDL_WINDOWEVENT_CLOSE &&
      event.window.windowID == SDL_GetWindowID(window))
00640                     done = true;
00641             }
00642 
00643             ImGui_ImplOpenGL2_NewFrame();
00644             ImGui_ImplSDL2_NewFrame();
00645             ImGui::NewFrame();
00646 
      //-----------------------------------------------------------------------------------------------------------------
00647             // USER CODE FRAME BEGIN
      ------------------------------------------------------------------------------
00648             // initialize Grid User Interface if Grid_UI is true
00649             UI::Dockspace_UI();
00650 
00651             if(uiFlag & UI_Flag_enable_GridUI) {
00652 
00653                 UI::grid_UI(/*io,*/ uiFlag, uiOptFlag, *this, stack, zoom, speed, count);
00654             }
00655             // initialize Setting User Interface if Setting_UI is true
00656             if(uiFlag & UI_Flag_enable_SetupUI) {
00657                 UI::setting_UI(uiFlag, uiOptFlag, *this, *reset_figure, stack, zoom, speed);
00658             }
00659             // initialize Load User Interface if Load_UI is true
00660             if(uiFlag & UI_Flag_enable_LoadUI){
00661                 UI::load_UI(uiFlag, uiOptFlag, *this, *reset_figure, message, zoom); // TODO remove
      zoom input variable
00662             }
00663             // initialize Message User Interface if Message_UI is true
00664             if(uiFlag & UI_Flag_enable_MessageUI){
00665                 UI::messageBox_UI(uiFlag, message);
00666             }
00667             // initialize Screenshot User Interface if Save_UI is true
00668             if(uiFlag & UI_Flag_enable_SaveUI){
00669                 UI::screenshot_UI(uiFlag ,*this, message );
00670             }
00671             if(uiFlag & UI_Flag_enable_LoadExampleUI) {
00672                 UI::load_example_UI(uiFlag, uiOptFlag, *this, *reset_figure, zoom, speed);
00673             }
00674             // USER CODE FRAME END
      ------------------------------------------------------------------------------
00675 
      //-----------------------------------------------------------------------------------------------------------------
00676 
00677 
00678             // Rendering // https://github.com/ocornut/imgui.git
00679             ImGui::Render();
```

```
00680             glViewport(0, 0, (int)io.DisplaySize.x, (int)io.DisplaySize.y);
00681             glClearColor(clear_color.x * clear_color.w, clear_color.y * clear_color.w, clear_color.z *
      clear_color.w, clear_color.w);
00682             glClear(GL_COLOR_BUFFER_BIT);
00683             //glUseProgram(0); // You may want this if using this code in an OpenGL 3+ context where
      shaders may be bound
00684             ImGui_ImplOpenGL2_RenderDrawData(ImGui::GetDrawData());
00685 
00686             // Update and Render additional Platform Windows // https://github.com/ocornut/imgui.git
00687             // (Platform functions may change the current OpenGL context, so we save/restore it to
      make it easier to paste this code elsewhere.
00688             //  For this specific demo app we could also call SDL_GL_MakeCurrent(window, gl_context)
      directly)
00689             if (io.ConfigFlags & ImGuiConfigFlags_ViewportsEnable)
00690             {
00691                 SDL_Window* backup_current_window = SDL_GL_GetCurrentWindow();
00692                 SDL_GLContext backup_current_context = SDL_GL_GetCurrentContext();
00693                 ImGui::UpdatePlatformWindows();
00694                 ImGui::RenderPlatformWindowsDefault();
00695                 SDL_GL_MakeCurrent(backup_current_window, backup_current_context);
00696             }
00697 
00698             SDL_GL_SwapWindow(window);
00699         }
00700 
00701 
00702 
00703         // Cleanup // https://github.com/ocornut/imgui.git
00704         ImGui_ImplOpenGL2_Shutdown();
00705         ImGui_ImplSDL2_Shutdown();
00706         ImGui::DestroyContext();
00707         //SDL_GLContext gl_context = SDL_GL_CreateContext(window);
00708         SDL_GL_DeleteContext(gl_context);
00709         SDL_DestroyWindow(window);
00710         SDL_Quit(); //
00711 
00712         return 0;
00713 
00714     }
00715 
00716 
      //=============================================================================================================
      CLASS UI ===//
00717     void UI::print(GameOfLife *_gameOfLife, float zoom) {
00718 
00719         ImVec2 p = ImGui::GetCursorScreenPos();
00720 
00721         // Create new child for scrolling
00722         ImVec2 scrolling_child_size = ImVec2(ImGui::GetFrameHeight() +
      (float)_gameOfLife->get_grid()->get_dimension().x*zoom,
      (float)_gameOfLife->get_grid()->get_dimension().y*zoom);
00723         ImGui::BeginChild("scrolling", scrolling_child_size, true,
      ImGuiWindowFlags_HorizontalScrollbar);
00724         // create for every living cell a white rectangle
00725         for(int y = 0; y < _gameOfLife->get_grid()->get_dimension().y; y++){
00726             for(int x = 0; x < _gameOfLife->get_grid()->get_dimension().x; x++){
00727                 if(_gameOfLife->get_grid()->get_cell((Vec2){x,y})->get_state()){
00728                     ImGui::GetWindowDrawList()->AddRectFilled(ImVec2(p.x+((float)x*zoom)+1,
      p.y+((float)y*zoom)+1), ImVec2(p.x+((float)x*zoom)+zoom, p.y+((float)y*zoom)+zoom),
      _gameOfLife->get_grid()->get_cell((Vec2){x,y})->get_color());
00729                     //ImGui::GetWindowDrawList()->AddRectFilled(ImVec2(p.x+(x*zoom), p.y+(y*zoom)),
      ImVec2(p.x+(x*zoom)+zoom, p.y+(y*zoom)+zoom),
      _gameOfLife->get_grid()->get_cell((Vec2){x,y})->get_color());
00730 
00731                 }
00732             }
00733         }
00734 
00735 
00736         ImGui::EndChild();
00737     }
00738 
00739     void UI::button_setting_zoom(UI_Options_Flag &uiOptionsFlag, float &factor) {
00740 
00741         static bool auto_zoom;
00742         if(uiOptionsFlag & UI_Options_Flag_enable_AutoZoom){ auto_zoom = true; }else{ auto_zoom =
      false; }
00743         ImGui::Checkbox("Auto Zoom", &auto_zoom);
00744         if(auto_zoom){uiOptionsFlag |= UI_Options_Flag_enable_AutoZoom; }else{uiOptionsFlag &=
      ~UI_Options_Flag_enable_AutoZoom; }
00745 
00746         // Zoom in button
00747         const ImVec2 size = ImVec2(96,20.);
00748         if(!(uiOptionsFlag & UI_Options_Flag_enable_AutoZoom)) {
00749             if (ImGui::Button("Zoom +", size)) {
00750 
00751                 factor +=2;
```

```
00752             }
00753             ImGui::SameLine();
00754             // Zoom out button
00755             if (ImGui::Button("Zoom -", size)) {
00756                 if (factor > 2) { factor -=2 ; }else{ factor = 2; }
00757             }
00758         }
00759     }
00760     void UI::button_setting_speed(int &factor) {
00761         // speed up button
00762         const ImVec2 size = ImVec2(96,20.);
00763         if (ImGui::Button("Speed +",size)) {
00764             if(factor > 0){factor--;}
00765         }
00766         ImGui::SameLine();
00767         // speed down button
00768         if (ImGui::Button("speed -",size)) {
00769             factor++;
00770         }
00771     }
00772     void UI::button_setting_run(UI_Options_Flag &uiOptionsFlag) {
00773         const ImVec2 size_run = ImVec2(96,20.);
00774         if(!(uiOptionsFlag & UI_Options_Flag_Run)) {
00775             ImGui::PushID(2);
00776             //color green
00777             ImGui::PushStyleColor(ImGuiCol_Button, (ImVec4) ImColor::HSV(2 / 7.0f, 0.6f, 0.6f));
00778             ImGui::PushStyleColor(ImGuiCol_ButtonHovered, (ImVec4) ImColor::HSV(2 / 7.0f, 0.7f,
    0.7f));
00779             ImGui::PushStyleColor(ImGuiCol_ButtonActive, (ImVec4) ImColor::HSV(2 / 7.0f, 0.8f, 0.8f));
00780             // create button
00781             if (ImGui::Button("RUN", size_run)) {
00782                 uiOptionsFlag |= UI_Options_Flag_Run;
00783             }
00784             ImGui::PopStyleColor(3);
00785             ImGui::PopID();
00786         }else{
00787             ImGui::PushID(0);
00788             //color red
00789             ImGui::PushStyleColor(ImGuiCol_Button, (ImVec4)ImColor::HSV(0 / 7.0f, 0.6f, 0.6f));
00790             ImGui::PushStyleColor(ImGuiCol_ButtonHovered, (ImVec4)ImColor::HSV(0 / 7.0f, 0.7f, 0.7f));
00791             ImGui::PushStyleColor(ImGuiCol_ButtonActive, (ImVec4)ImColor::HSV(0 / 7.0f, 0.8f, 0.8f));
00792             // create button
00793             if (ImGui::Button("STOP", size_run)) {
00794                 uiOptionsFlag &= ~UI_Options_Flag_Run;
00795             }
00796             ImGui::PopStyleColor(3);
00797             ImGui::PopID();
00798         }
00799     }
00800     /*void UI::button_setting_stop(UI_Options_Flag &uiOptionsFlag) {
00801         const ImVec2 size_stop = ImVec2(44,20.);
00802         ImGui::PushID(0);
00803         //color red
00804         ImGui::PushStyleColor(ImGuiCol_Button, (ImVec4)ImColor::HSV(0 / 7.0f, 0.6f, 0.6f));
00805         ImGui::PushStyleColor(ImGuiCol_ButtonHovered, (ImVec4)ImColor::HSV(0 / 7.0f, 0.7f, 0.7f));
00806         ImGui::PushStyleColor(ImGuiCol_ButtonActive, (ImVec4)ImColor::HSV(0 / 7.0f, 0.8f, 0.8f));
00807         // create button
00808         if (ImGui::Button("STOP", size_stop)) {
00809             uiOptionsFlag &= ~UI_Options_Flag_Run;
00810         }
00811         ImGui::PopStyleColor(3);
00812         ImGui::PopID();
00813     }*/
00814     void UI::button_setting_reset(GameOfLife &gameOfLife, Figure &reset_figure, Figure_Stack
    &figureStack) {
00815         const ImVec2 size_res = ImVec2(44,20.);
00816         // create button
00817         if (ImGui::Button("reset", size_res)) {
00818             gameOfLife.clear_grid();
00819             gameOfLife.populate_figure(&reset_figure, (Vec2){0,0},0);
00820             //Colorize grid
00821             for(auto c: gameOfLife.get_grid()->get_cells()){
00822                 c->get_neighbours()->set_n_alive(0);
00823                 for(auto n: c->get_neighbours()->get_cells()){
00824                     if(n->get_state()){
    c->get_neighbours()->set_n_alive(c->get_neighbours()->get_n_alive()+1); }
00825                 }
00826                 c->set_color(gameOfLife.get_color_rules()->color[c->get_neighbours()->get_n_alive()]);
00827             }
00828
00829             figureStack.clear();
00830         }
00831     }
00832     void UI::button_setting_step(GameOfLife &gameOfLife, Figure_Stack &_stack) {
00833         const ImVec2 size = ImVec2(96,20.);
00834         // create back button
00835         if (ImGui::Button("« Step", size)) {
```

```
00836             if(!_stack.empty()){
00837
00838                 gameOfLife.populate_figure(_stack.back(), Vec2{0, 0}, 0);
00839                 _stack.pop_back();
00840                 // COLORIZE
00841                 for(auto c: gameOfLife.get_grid()->get_cells()){
00842                     c->get_neighbours()->set_n_alive(0);
00843                     for(auto n: c->get_neighbours()->get_cells()){
00844                         if(n->get_state()){
      c->get_neighbours()->set_n_alive(c->get_neighbours()->get_n_alive()+1);}
00845                     }
00846
      c->set_color(gameOfLife.get_color_rules()->color[c->get_neighbours()->get_n_alive()]);
00847                 }
00848             }
00849         }
00850         ImGui::SameLine();
00851         // step for button
00852         if (ImGui::Button("Step »", size)) {
00853             _stack.push_back(new Figure(&gameOfLife));
00854             if(_stack.size() > STACK_SIZE){ _stack.pop_front(); }
00855             gameOfLife.refresh_grid();
00856
00857         }
00858     }
00859     void UI::button_setting_clear(GameOfLife &gameOfLife, Figure &reset_figure, Figure_Stack
      &figureStack) {
00860         const ImVec2 size_clr = ImVec2(44,20.);
00861         if (ImGui::Button("clear",size_clr)) {
00862             gameOfLife.clear_grid();
00863             reset_figure = Figure(&gameOfLife);
00864             figureStack.clear();
00865         }
00866     }
00867     void UI::button_setting_random(GameOfLife &gameOfLife, Figure &reset_figure,Figure_Stack
      &figureStack) {
00868         const ImVec2 size_rand = ImVec2(200.,20.);
00869         if (ImGui::Button("random", size_rand)) {
00870             gameOfLife.clear_grid();
00871             gameOfLife.populate_random();
00872             reset_figure = Figure(&gameOfLife);
00873             figureStack.clear();
00874         }
00875     }
00876     void UI::button_setting_load(UI_Flag &uiFlag) {
00877         const ImVec2 size_load = ImVec2(200.,20.);
00878         if (ImGui::Button("load from file", size_load)) {
00879             uiFlag |= UI_Flag_enable_LoadUI;
00880         }
00881     }
00882     void UI::button_setting_load_exp(UI_Flag &uiFlag) {
00883         const ImVec2 size_load = ImVec2(200.,20.);
00884         if (ImGui::Button("load example", size_load)) {
00885             uiFlag |= UI_Flag_enable_LoadExampleUI;
00886         }
00887     }
00888     void UI::button_setting_screenshot(UI_Flag &uiFlag) {
00889         const ImVec2 size = ImVec2(200.,20.);
00890         if (ImGui::Button("screenshot to .lif file", size)) {
00891             uiFlag |= UI_Flag_enable_SaveUI;
00892         }
00893     }
00894     void UI::button_load(UI_Flag &uiFlag, UI_Options_Flag &uiOptionsFlag, GameOfLife &gameOfLife,
      Figure &reset_figure, std::string path, Vec2 position, int &angle, std::string &message, float &zoom)
      {
00895         ImGui::PushID(2);
00896         //color green
00897         ImGui::PushStyleColor(ImGuiCol_Button, (ImVec4)ImColor::HSV(2 / 7.0f, 0.6f, 0.6f));
00898         ImGui::PushStyleColor(ImGuiCol_ButtonHovered, (ImVec4)ImColor::HSV(2 / 7.0f, 0.7f, 0.7f));
00899         ImGui::PushStyleColor(ImGuiCol_ButtonActive, (ImVec4)ImColor::HSV(2 / 7.0f, 0.8f, 0.8f));
00900         if (ImGui::Button("Load")) {
00901
00902             try {
00903                 std::string _path = (std::string) std::move(path);
00904                 auto *_f = new gol::Figure(_path);
00905
00906                 // ERROR HANDLING
00907                 if(_f->get_dimension().x+position.x > gameOfLife.get_grid()->get_dimension().x){throw
      std::logic_error("ERROR: Figure Dimension x in comination with position x are to large");}
00908                 if(_f->get_dimension().y+position.y > gameOfLife.get_grid()->get_dimension().y){throw
      std::logic_error("ERROR: Figure Dimension y in comination with position y are to large");}
00909
00910
00911                 gameOfLife.populate_figure(_f, position, (angle / 90));
00912                 reset_figure = Figure(&gameOfLife);
00913                 uiFlag &= ~UI_Flag_enable_LoadUI;
00914                 delete _f;
```

```
00915
00916                 //Colorize grid
00917                 for(auto c: gameOfLife.get_grid()->get_cells()){
00918                     c->get_neighbours()->set_n_alive(0);
00919                     for(auto n: c->get_neighbours()->get_cells()){
00920                         if(n->get_state()){
    c->get_neighbours()->set_n_alive(c->get_neighbours()->get_n_alive()+1); }
00921                     }
00922
    c->set_color(gameOfLife.get_color_rules()->color[c->get_neighbours()->get_n_alive()]);
00923                 }
00924                 //zoom = gameOfLife.get_grid()->get_auto_zoom_factor();
00925                 uiOptionsFlag |= UI_Options_Flag_enable_AutoZoom | UI_Options_Flag_enable_AutoScroll;
00926             }catch(std::logic_error &err){
00927                 message = err.what();
00928                 uiFlag |= UI_Flag_enable_MessageUI;
00929             }
00930
00931
00932
00933         }
00934         ImGui::PopStyleColor(3);
00935         ImGui::PopID();
00936     }
00937     void UI::button_save(UI_Flag &uiFlag, GameOfLife &gameOfLife, std::string path) {
00938         //color green
00939         ImGui::PushID(2);
00940         ImGui::PushStyleColor(ImGuiCol_Button, (ImVec4)ImColor::HSV(2 / 7.0f, 0.6f, 0.6f));
00941         ImGui::PushStyleColor(ImGuiCol_ButtonHovered, (ImVec4)ImColor::HSV(2 / 7.0f, 0.7f, 0.7f));
00942         ImGui::PushStyleColor(ImGuiCol_ButtonActive, (ImVec4)ImColor::HSV(2 / 7.0f, 0.8f, 0.8f));
00943         if (ImGui::Button("Save")) {
00944             std::string Filename(path);
00945
00946             //Grid* grid = gameOfLife.get_grid();
00947
00948             auto *_fig = new gol::Figure(&gameOfLife);
00949
00950             _fig->save_to_file(path);
00951
00952             uiFlag &= ~UI_Flag_enable_SaveUI;
00953
00954         }
00955         ImGui::PopStyleColor(3);
00956         ImGui::PopID();
00957     }
00958
00959     void UI::setting_UI(UI_Flag &uiFlag, UI_Options_Flag &uiOptionsFlag, GameOfLife &gameOfLife,
    Figure &reset_figure, Figure_Stack &figureStack,float &zoom, int &speed) {
00960
00961         ImGui::Begin("Settings", nullptr, /*ImGuiWindowFlags_NoMove |*/
    ImGuiWindowFlags_NoDecoration);
00962
00963         //Autoscroll Checkbox Begin
00964         bool autoScroll;
00965         if(uiOptionsFlag & UI_Options_Flag_enable_AutoScroll){ autoScroll = true; }else{ autoScroll =
    false; }
00966         ImGui::Checkbox("Auto Scroll", &autoScroll);
00967         if(autoScroll){ uiOptionsFlag |= UI_Options_Flag_enable_AutoScroll;}else{uiOptionsFlag &=
    ~UI_Options_Flag_enable_AutoScroll;}
00968         //Autoscroll Checkbox End
00969
00970         UI::button_setting_zoom(uiOptionsFlag, zoom);      // create zoom button
00971
00972         UI::button_setting_speed(speed);    // create speed button
00973         UI::button_setting_step(gameOfLife, figureStack); // create step button
00974         ImGui::Spacing();
00975         UI::button_setting_run(uiOptionsFlag);        // run game of life button
00976         //ImGui::SameLine();
00977         //UI::button_setting_stop(uiOptionsFlag);       // stop game of life button
00978
00979         ImGui::SameLine();
00980         UI::button_setting_reset(gameOfLife, reset_figure, figureStack); // reset game of life button
00981
00982         ImGui::SameLine();
00983         UI::button_setting_clear(gameOfLife, reset_figure, figureStack); // clear game of life button
00984
00985         ImGui::Spacing();
00986         UI::button_setting_random(gameOfLife, reset_figure, figureStack); // random button
00987
00988         UI::button_setting_load(uiFlag);   // load from file button
00989         UI::button_setting_load_exp(uiFlag);
00990         UI::button_setting_screenshot(uiFlag);
00991         ImGui::Spacing();
00992
00993         UI::setup_rules(uiOptionsFlag, gameOfLife);    // rules setting checkboxes
00994
00995         ImGui::End();
```

```
00996
00997     }
00998
00999     void UI::Dockspace_UI(){
01000         static ImGuiDockNodeFlags dockspace_flags = ImGuiDockNodeFlags_None |
      ImGuiDockNodeFlags_PassthruCentralNode;
01001
01002         ImGuiWindowFlags window_flags = ImGuiWindowFlags_NoDocking;
01003         window_flags |= ImGuiWindowFlags_NoTitleBar | ImGuiWindowFlags_NoCollapse |
      ImGuiWindowFlags_NoResize | ImGuiWindowFlags_NoMove;
01004         window_flags |= ImGuiWindowFlags_NoBringToFrontOnFocus | ImGuiWindowFlags_NoNavFocus;
01005         window_flags |= ImGuiWindowFlags_NoBackground ;
01006
01007         const ImGuiViewport* viewport = ImGui::GetMainViewport();
01008         ImGui::SetNextWindowPos(viewport->WorkPos);
01009         ImGui::SetNextWindowSize(viewport->WorkSize);
01010
01011         ImGui::Begin("DockSpace", nullptr, window_flags);
01012
01013         ImGuiID dockspace_id = ImGui::GetID("MyDockSpace");
01014         ImGui::DockSpace(dockspace_id, ImVec2(0.0f, 0.0f), dockspace_flags);
01015
01016         ImGui::End();
01017     }
01018     void UI::grid_UI(UI_Flag &uiFlag, UI_Options_Flag &uiOptionsFlag, GameOfLife &gameOfLife,
      Figure_Stack &figureStack, float &zoom, int &speed, int &count) {
01019
01020
01021
01022
01023         ImGui::Begin("GRID",NULL,ImGuiWindowFlags_HorizontalScrollbar /*| ImGuiWindowFlags_NoMove */|
      ImGuiWindowFlags_MenuBar | ImGuiWindowFlags_NoTitleBar);
01024
01025         //Auto Scroll Begin
01026         if(uiOptionsFlag & UI_Options_Flag_enable_AutoScroll) {
01027
01028             ImGui::SetScrollX(gameOfLife.get_grid()->get_scrollx_position()*(ImGui::GetScrollMaxX()));
      //TODO
01029             ImGui::SetScrollY(gameOfLife.get_grid()->get_scrolly_position()*(ImGui::GetScrollMaxY()));
      //TODO
01030
01031             //std::cerr << ImGui::GetContentRegionMax().x << "x" << ImGui::GetContentRegionMax().y << " <-
      ";
01032             //std::cerr << ImGui::GetScrollMaxX() << "x" << ImGui::GetScrollMaxY() << "\n"; //TODO debug
01033         }
01034         //Auto Scroll End
01035         //Auto Zoom Begin
01036         if(uiOptionsFlag & UI_Options_Flag_enable_AutoZoom) {
01037             if ((gameOfLife.get_grid()->get_real_grid_dimension().x > 0) &&
      (gameOfLife.get_grid()->get_real_grid_dimension().y > 0)) {
01038                 zoom = gameOfLife.get_grid()->get_auto_zoom_factor();
01039             }
01040         }
01041         //Auto Zoom End
01042
01043
01044
01045         ImGuiIO& io = ImGui::GetIO(); (void)io;
01046         ImGui::Text("Application average %.3f ms/frame (%.2f FPS)", 1000.0f / io.Framerate,
      io.Framerate); //TODO: FOR DEBUG
01047
01048         // Menubar
01049         static bool setting = false;
01050         static bool run;
01051         if(uiFlag & UI_Flag_enable_SetupUI){ setting = true; }
01052         if(uiOptionsFlag & UI_Options_Flag_Run){ run = true; }else{run = false; }
01053         if (ImGui::BeginMenuBar()){
01054             if (ImGui::BeginMenu("Options")){
01055                 ImGui::MenuItem("Settings", NULL, &setting);
01056                 ImGui::MenuItem("run", NULL, &run); //TODO
01057                 ImGui::EndMenu();
01058             }
01059
01060         }
01061         ImGui::EndMenuBar();
01062         if(setting){ uiFlag |= UI_Flag_enable_SetupUI; }else{ uiFlag &= ~UI_Flag_enable_SetupUI; }
01063         if(run){ uiOptionsFlag |= UI_Options_Flag_Run; }else{ uiOptionsFlag &= ~UI_Options_Flag_Run; }
01064         // print grid to imGUI figure
01065         UI::print(&gameOfLife, zoom);
01066
01067         // run
01068         if (uiOptionsFlag & UI_Options_Flag_Run) {
01069             if (count > speed) { count = speed; }
01070             if (count == speed) {
01071                 figureStack.push_back(new Figure(&gameOfLife));
01072                 if(figureStack.size() > STACK_SIZE){ figureStack.pop_front(); }
01073                 gameOfLife.refresh_grid();
```

```
01074                 count = 0;
01075             } else { count++; }
01076         }
01077
01078         ImGui::End();
01079     }
01080     void UI::load_UI(UI_Flag &uiFlag, UI_Options_Flag &uiOptionsFlag, GameOfLife &gameOfLife, Figure
      &reset_figure, std::string &message, float &zoom) {
01081         static char path[256] =
      "/Users/user404/Studium/CE27_Softwaretechnik/Game_Of_Life2.0/figures/toggle.lif";
01082         static int x_pos = 0;
01083         static int y_pos = 0;
01084         static int angle = 0;
01085
01086         ImGui::Begin("Load from File", nullptr, 0/*ImGuiWindowFlags_NoMove*/  );
01087         ImGui::Text("enter full filepath");
01088         ImGui::InputText("filepath", path, IM_ARRAYSIZE(path)); //Text input path
01089
01090
01091         ImGui::Text("enter x and y position of figure in grid");
01092         ImGui::InputInt("x", &x_pos); // Textfield x position
01093         if(x_pos > gameOfLife.get_grid()->get_dimension().x){x_pos =
      gameOfLife.get_grid()->get_dimension().x; }
01094         if(x_pos < 0){x_pos = 0;}
01095
01096         ImGui::InputInt("y", &y_pos); // Textfield y position
01097         if(y_pos > gameOfLife.get_grid()->get_dimension().y){ y_pos =
      gameOfLife.get_grid()->get_dimension().y; }
01098         if(y_pos < 0){y_pos = 0;}
01099
01100
01101         ImGui::Text("enter angle of figure in grid");
01102         ImGui::InputInt("angle", &angle, 90, 90, 0); // Textfield x position
01103         if(angle > 270) {
01104             angle = 270;
01105         }
01106         if(angle < 0){
01107             angle = 0;
01108         }
01109
01110         // Button load from file
01111         try {
01112             button_load(uiFlag, uiOptionsFlag, gameOfLife, reset_figure, (std::string) path, (Vec2)
      {x_pos, y_pos}, angle, message, zoom);
01113         }catch(std::logic_error &err){
01114             message = (std::string)err.what();
01115             uiFlag |= UI_Flag_enable_MessageUI;
01116         }
01117         // abort button
01118         ImGui::SameLine();
01119         //color red
01120         ImGui::PushID(0);
01121         ImGui::PushStyleColor(ImGuiCol_Button, (ImVec4)ImColor::HSV(0 / 7.0f, 0.6f, 0.6f));
01122         ImGui::PushStyleColor(ImGuiCol_ButtonHovered, (ImVec4)ImColor::HSV(0 / 7.0f, 0.7f, 0.7f));
01123         ImGui::PushStyleColor(ImGuiCol_ButtonActive, (ImVec4)ImColor::HSV(0 / 7.0f, 0.8f, 0.8f));
01124         if (ImGui::Button("Abort")) {
01125             uiFlag &= ~UI_Flag_enable_LoadUI;
01126         }
01127         ImGui::PopStyleColor(3);
01128         ImGui::PopID();
01129
01130         ImGui::End();
01131
01132         //TODO add some standard figures
01133     }
01134     void UI::screenshot_UI(UI_Flag &uiFlag, GameOfLife &gameOfLife, std::string &message){
01135         static char path[256] =
      "/Users/user404/Studium/CE27_Softwaretechnik/PROTO_GOL/test_Screenshot.lif";
01136
01137         ImGui::Begin("Save to File", nullptr, 0/*ImGuiWindowFlags_NoMove*/);
01138         ImGui::Text("enter full filepath");
01139         ImGui::InputText("filepath", path, IM_ARRAYSIZE(path)); //Text input path
01140
01141         try {
01142             button_save(uiFlag,gameOfLife, path);
01143         }catch(std::logic_error &err){
01144             message = (std::string)err.what();
01145             uiFlag |= UI_Flag_enable_MessageUI;
01146         }
01147         // abort button
01148         ImGui::SameLine();
01149         ImGui::PushID(0);
01150         //color red
01151         ImGui::PushStyleColor(ImGuiCol_Button, (ImVec4)ImColor::HSV(0 / 7.0f, 0.6f, 0.6f));
01152         ImGui::PushStyleColor(ImGuiCol_ButtonHovered, (ImVec4)ImColor::HSV(0 / 7.0f, 0.7f, 0.7f));
01153         ImGui::PushStyleColor(ImGuiCol_ButtonActive, (ImVec4)ImColor::HSV(0 / 7.0f, 0.8f, 0.8f));
01154         if (ImGui::Button("Abort")) {
```

```
01155             uiFlag &= ~UI_Flag_enable_SaveUI;
01156         }
01157         ImGui::PopStyleColor(3);
01158         ImGui::PopID();
01159 
01160 
01161         ImGui::End();
01162     }
01163     void UI::messageBox_UI(UI_Flag &uiFlag, std::string &message) {
01164         ImGui::Begin("Message");
01165 
01166         ImGui::Text("%s", message.c_str());
01167 
01168         if (ImGui::Button("OK")) {
01169             uiFlag &= ~UI_Flag_enable_MessageUI;
01170             message.clear();
01171         }
01172 
01173         ImGui::End();
01174     }
01175     void UI::ColorPicker(GameOfLife &gameOfLife, const char* label, ImU32 *color){
01176         //ImGui::Begin("Color Picker");
01177             float col[4];
01178             col[0] = (float)((*color      ) & 0xFF) / 255.0f;
01179             col[1] = (float)((*color » 8 ) & 0xFF) / 255.0f;
01180             col[2] = (float)((*color » 16) & 0xFF) / 255.0f;
01181             col[3] = (float)((*color » 24) & 0xFF) / 255.0f;
01182 
01183             static bool drag_and_drop = false;
01184             static bool hdr = true;
01185             static bool alpha_preview = true;
01186             static bool alpha_half_preview = true;
01187             static bool options_menu = true;
01188             ImGuiColorEditFlags misc_flags = (hdr ? ImGuiColorEditFlags_HDR : 0) | (drag_and_drop ? 0
      : ImGuiColorEditFlags_NoDragDrop) | (alpha_half_preview ? ImGuiColorEditFlags_AlphaPreviewHalf :
      (alpha_preview ? ImGuiColorEditFlags_AlphaPreview : 0)) | (options_menu ? 0 :
      ImGuiColorEditFlags_NoOptions);
01189             //ImGui::ColorPicker4("##picker", &col[0], misc_flags | ImGuiColorEditFlags_NoSidePreview
      | ImGuiColorEditFlags_NoSmallPreview);
01190             //ImGui::ColorButton("##current", (ImVec4){col[0],col[1],col[2],col[3]},
      ImGuiColorEditFlags_NoPicker | ImGuiColorEditFlags_AlphaPreviewHalf, ImVec2(60, 40));
01191             //ImGuiColorEditFlags palette_button_flags = ImGuiColorEditFlags_NoAlpha |
      ImGuiColorEditFlags_NoPicker | ImGuiColorEditFlags_NoTooltip;
01192             //ImGui::ColorButton("##palette", (ImVec4){col[0],col[1],col[2],col[3]},
      palette_button_flags, ImVec2(20, 20));
01193 
01194             ImGui::ColorEdit4(label, &col[0], ImGuiColorEditFlags_NoInputs | misc_flags);
01195 
01196 
01197             ImU32 _col = ((ImU32)(col[0] * 255.0f)      ) |
01198                          ((ImU32)(col[1] * 255.0f) «  8) |
01199                          ((ImU32)(col[2] * 255.0f) « 16) |
01200                          ((ImU32)(col[3] * 255.0f) « 24);
01201 
01202             *color = _col;
01203 
01204 
01205 
01206 
01207         //ImGui::End();
01208     }
01209     void UI::setup_rules(UI_Options_Flag &uiOptionsFlag, GameOfLife &gameOfLife) {
01210         ImGui::Text("Rules"); //ImGui::SameLine();
01211         for(int i=0; i < 9; i++) {
01212             std::string label_a = std::to_string(i);
01213             label_a += " Alive";
01214             std::string label_d = std::to_string(i);
01215             label_d += " Dead";
01216 
01217             bool d = gameOfLife.get_rules()->death[i];
01218             bool a = gameOfLife.get_rules()->alive[i];
01219 
01220             ImGui::Checkbox(label_a.c_str(), &gameOfLife.get_rules()->alive[i]);
01221             ImGui::SameLine();
01222             ImGui::Checkbox(label_d.c_str(), &gameOfLife.get_rules()->death[i]);
01223             if (((gameOfLife.get_rules()->alive[i])) && (gameOfLife.get_rules()->death[i])) {
01224                 if (d && !a) {
01225                     gameOfLife.get_rules()->death[i] = false;
01226                     gameOfLife.get_rules()->alive[i] = true;
01227                 } else {
01228                     gameOfLife.get_rules()->death[i] = true;
01229                     gameOfLife.get_rules()->alive[i] = false;
01230                 }
01231 
01232             }
01233 
01234 
```

```
01235             // Color Picker
01236             if (!(uiOptionsFlag & UI_Options_Flag_enable_ColorizeAll)) {
01237                 ImGui::SameLine();
01238                 char index[2] = {char(i + 0x30), 0x00};
01239                 ColorPicker(gameOfLife, index, &gameOfLife.get_color_rules()->color[i]);
01240             }
01241         }
01242         button_setting_reset_colors(gameOfLife.get_color_rules());
01243         static bool color_all;
01244         if(uiOptionsFlag & UI_Options_Flag_enable_ColorizeAll){color_all = true; }else{color_all =
      false;}
01245         ImGui::Checkbox("color all: ", &color_all);
01246         if(color_all){uiOptionsFlag |= UI_Options_Flag_enable_ColorizeAll;}else{uiOptionsFlag &=
      ~UI_Options_Flag_enable_ColorizeAll; }
01247         if(uiOptionsFlag & UI_Options_Flag_enable_ColorizeAll) {
01248             ImGui::SameLine();
01249             ImU32 &color_a = gameOfLife.get_color_rules()->color[0];
01250 
01251             ColorPicker(gameOfLife, " ", &color_a);
01252             for(unsigned int & i : gameOfLife.get_color_rules()->color){
01253 
01254                 i = color_a;
01255             }
01256 
01257         }
01258         button_setting_reset_rules(gameOfLife.get_rules());
01259     }
01260 
01261     void UI::button_setting_reset_rules(Rule *_rule) {
01262         const ImVec2 size = ImVec2(200.,20.);
01263         if (ImGui::Button("Cornways", size)) {
01264             for(int r=0; r < 9; r++){
01265                 // reset game of life rules to cornways (23/3)
01266                 if(r == 3){_rule->alive[r] = true; }else{ _rule->alive[r] = false; }
01267                 if((r == 2) || (r == 3)){ _rule->death[r] = false; }else{ _rule->death[r] = true; }
01268             }
01269         }
01270         if (ImGui::Button("Anti Cornways", size)) {
01271             for(int r=0; r < 9; r++){
01272                 // reset game of life rules to cornways (56/5)
01273                 if(r == 5){_rule->death[r] = true; }else{_rule->death[r] = false; }
01274                 if((r == 5) || (r == 6)){_rule->alive[r] = false; }else{ _rule->alive[r] = true; }
01275 
01276             }
01277         }
01278         if (ImGui::Button("Kopiersystem", size)) {
01279             for(int r=0; r < 9; r++){
01280                 // reset game of life rules to (1357/1357)
01281                 if((r==1)||(r==3)||(r==5)||(r==7)){_rule->alive[r] = true; }else{_rule->alive[r] =
      false; }
01282                 if((r==1)||(r==3)||(r==5)||(r==7)){_rule->death[r] = false; }else{_rule->death[r] =
      true; }
01283             }
01284         }
01285         if (ImGui::Button("Anti Kopiersystem", size)) {
01286             for(int r=0; r < 9; r++){
01287                 // reset game of life rules to (1357/1357)
01288                 if((r==1)||(r==3)||(r==5)||(r==7)){_rule->death[r] = true; }else{_rule->death[r] =
      false; }
01289                 if((r==1)||(r==3)||(r==5)||(r==7)){_rule->alive[r] = false; }else{_rule->alive[r] =
      true; }
01290             }
01291         }
01292 
01293 
01294     }
01295     void UI::button_setting_reset_colors(RuleColor* _rule){
01296 
01297         const ImVec2 size = ImVec2(200.,20.);
01298         if (ImGui::Button("Reset colors", size)) {
01299             // all colors are white
01300             *_rule = gol::RuleColor{
01301                     IM_COL32(255,0,0,255),
01302                     IM_COL32(198,57,0,255),
01303                     IM_COL32(141,114,0,255),
01304                     IM_COL32(84,171,0,255),
01305                     IM_COL32(0,255,0,255),
01306                     IM_COL32(0,198,57,255),
01307                     IM_COL32(0,141,114,255),
01308                     IM_COL32(0,84,171,255),
01309                     IM_COL32(0,0,255,255)
01310             };
01311         }
01312 
01313     }
01314 
01315     void UI::load_example_UI(UI_Flag &uiFlag, UI_Options_Flag &uiOptionsFlag, GameOfLife &gameOfLife,
```

```
    Figure &reset_figure, float &zoom, int &speed) {
01316         static int x_pos = 0;
01317         static int y_pos = 0;
01318         static int angle = 0;
01319 
01320         static bool example_flags[] = {true,false,false,false,false,false,false,false};
01321         bool last_flags[8];
01322 
01323 
01324 
01325 
01326         for(int i=0; i < 8; i++){
01327             last_flags[i] = example_flags[i];
01328         }
01329 
01330         ImGui::Begin("Load example figures", nullptr, 0/*ImGuiWindowFlags_NoMove*/);
01331 
01332         ImGui::Text("Cornway's example figures");
01333         ImGui::Checkbox("Glider", &example_flags[0]);
01334         ImGui::Checkbox("Light-Weight Spaceship", &example_flags[1]);
01335         ImGui::Checkbox("Middle-Weight Spaceship", &example_flags[2]);
01336         ImGui::Checkbox("Heavy-Weight Spaceship", &example_flags[3]);
01337         ImGui::Checkbox("Gosper Gun", &example_flags[4]);
01338         ImGui::Checkbox("Eater", &example_flags[5]);
01339 
01340         ImGui::Text("\nother example figures");
01341         ImGui::Checkbox("HTW Logo", &example_flags[6]);
01342         ImGui::Checkbox("Pacman", &example_flags[7]);
01343 
01344         bool equal = true;
01345         for(int i=0; i < 8; i++){
01346             if(example_flags[i] != last_flags[i]){
01347                 equal = false;
01348             }
01349         }
01350 
01351         if(!equal){
01352             for(int i=0; i < 8; i++){
01353                 example_flags[i] =  (example_flags[i] + last_flags[i])%2; // XOR
01354             }
01355         }
01356 
01357         ImGui::Text("enter x and y position ");
01358         ImGui::InputInt("x", &x_pos); // Textfield x position
01359         if(x_pos > gameOfLife.get_grid()->get_dimension().x){x_pos =
    gameOfLife.get_grid()->get_dimension().x; }
01360         if(x_pos < 0){x_pos = 0;}
01361 
01362         ImGui::InputInt("y", &y_pos); // Textfield y position
01363         if(y_pos > gameOfLife.get_grid()->get_dimension().y){ y_pos =
    gameOfLife.get_grid()->get_dimension().y; }
01364         if(y_pos < 0){y_pos = 0;}
01365 
01366 
01367         ImGui::Text("enter angle of figure");
01368         ImGui::InputInt("angle", &angle, 90, 90, 0); // Textfield x position
01369         if(angle > 270) {
01370             angle = 270;
01371         }
01372         if(angle < 0){
01373             angle = 0;
01374         }
01375 
01376         ImGui::PushID(2);
01377         //color green
01378         ImGui::PushStyleColor(ImGuiCol_Button, (ImVec4)ImColor::HSV(2 / 7.0f, 0.6f, 0.6f));
01379         ImGui::PushStyleColor(ImGuiCol_ButtonHovered, (ImVec4)ImColor::HSV(2 / 7.0f, 0.7f, 0.7f));
01380         ImGui::PushStyleColor(ImGuiCol_ButtonActive, (ImVec4)ImColor::HSV(2 / 7.0f, 0.8f, 0.8f));
01381         if (ImGui::Button("Load")) {
01382             gol::Rule r_cornway = (gol::Rule){{false,false, false, true, false, false, false, false,
    false},
01383                                               {true, true, false, false, true, true, true, true,
    true}};
01384 
01385             gol::Rule r_copy = (gol::Rule){{false,true, false, true, false, true, false, true, false},
01386                                            {true, false, true, false, true, false, true, false,
    true}};
01387 
01388             if(example_flags[0]){
01389                 //Figure *f = new Figure(glider, glider_dim);
01390 
01391                 uiOptionsFlag |= UI_Options_Flag_enable_AutoScroll | UI_Options_Flag_enable_AutoZoom;
01392                 std::string glider_string = "bob$2bo$3o!";
01393                 Vec2 glider_dim = {3, 3};
01394                 auto *f = new Figure(glider_string, glider_dim);
01395                 gameOfLife.populate_figure(f,(Vec2){x_pos,y_pos}, angle/90);
01396                 gameOfLife.set_rules(r_cornway);
```

```
01397                 zoom = gameOfLife.get_grid()->get_auto_zoom_factor();
01398             }
01399             if(example_flags[1]){
01400                 uiOptionsFlag |= UI_Options_Flag_enable_AutoScroll | UI_Options_Flag_enable_AutoZoom;
01401                 //Figure *f = new Figure(lws, lws_dim);
01402                 std::string lws_string = "b4o$o3bo$4bo$o2b1o!";
01403                 Vec2 lws_dim = {5, 4};
01404                 auto *f = new Figure(lws_string, lws_dim);
01405                 gameOfLife.populate_figure(f,(Vec2){x_pos,y_pos}, angle/90);
01406                 gameOfLife.set_rules(r_cornway);
01407                 zoom = gameOfLife.get_grid()->get_auto_zoom_factor();
01408 
01409             }
01410             if(example_flags[2]){
01411                 uiOptionsFlag |= UI_Options_Flag_enable_AutoScroll | UI_Options_Flag_enable_AutoZoom;
01412                 //Figure *f = new Figure(mws, mws_dim);
01413                 std::string mws_string = "b5o$o4bo$5bo$o3bob$2bo3b!";
01414                 Vec2 mws_dim = {6, 5};
01415                 auto *f = new Figure(mws_string, mws_dim);
01416                 gameOfLife.populate_figure(f,(Vec2){x_pos,y_pos}, angle/90);
01417                 gameOfLife.set_rules(r_cornway);
01418                 zoom = gameOfLife.get_grid()->get_auto_zoom_factor();
01419 
01420             }
01421             if(example_flags[3]){
01422                 uiOptionsFlag |= UI_Options_Flag_enable_AutoScroll | UI_Options_Flag_enable_AutoZoom;
01423 
01424                 //Figure *f = new Figure(hws, hws_dim);
01425                 std::string hws_string = "b6o$o5bo$6bo$o4bob$2b2o2b!";
01426                 Vec2 hws_dim = {7, 5};
01427                 auto *f = new Figure(hws_string, hws_dim);
01428                 gameOfLife.populate_figure(f,(Vec2){x_pos,y_pos}, angle/90);
01429                 gameOfLife.set_rules(r_cornway);
01430                 zoom = gameOfLife.get_grid()->get_auto_zoom_factor();
01431             }
01432             if(example_flags[4]){
01433                 uiOptionsFlag |= UI_Options_Flag_enable_AutoScroll | UI_Options_Flag_enable_AutoZoom;
01434 
01435                 Vec2 gosper_dim = {36, 9};
01436                 std::string gosper_string =
    "24bo$22bobo$12b2o6b2o12b2o$11bo3bo4b2o12b2o$2o8bo5bo3b2o$2o8bo3bob2o4bobo$10bo5bo7bo$11bo3bo$12b2o!";
01437 
01438                 auto *f = new Figure(gosper_string, gosper_dim);
01439                 gameOfLife.populate_figure(f,(Vec2){x_pos,y_pos}, angle/90);
01440                 gameOfLife.set_rules(r_cornway);
01441                 zoom = gameOfLife.get_grid()->get_auto_zoom_factor();
01442             }
01443             if(example_flags[5]){
01444                 uiOptionsFlag |= UI_Options_Flag_enable_AutoScroll | UI_Options_Flag_enable_AutoZoom;
01445                 //Figure *f = new Figure(eater, eater_dim);
01446                 std::string eater_string = "2o2b$obob$2bob$2b2o!";
01447                 Vec2 eater_dim = {4, 4};
01448                 auto *f = new Figure(eater_string, eater_dim);
01449                 gameOfLife.populate_figure(f,(Vec2){x_pos,y_pos}, angle/90);
01450                 gameOfLife.set_rules(r_cornway);
01451                 zoom = gameOfLife.get_grid()->get_auto_zoom_factor();
01452             }
01453             if(example_flags[6]){
01454                 uiOptionsFlag |= UI_Options_Flag_enable_AutoScroll | UI_Options_Flag_enable_AutoZoom;
01455                 //Figure *f = new Figure(htw, htw_dim);
01456                 std::string htw_string =
    "10o30b10o$10o30b10o$10o30b10o$10o30b10o$10o30b10o$10o30b10o$10o30b10o$10o30b10o$10o30b10o$10o30b10o$10o30b10o$10o30b10o$10b11o19b20o10b10o1
01457                 Vec2 htw_dim = {130, 40};
01458                 auto *f = new Figure(htw_string, htw_dim);
01459                 gameOfLife.populate_figure(f,(Vec2){x_pos,y_pos}, angle/90);
01460                 gameOfLife.set_rules(r_copy);
01461                 zoom = gameOfLife.get_grid()->get_auto_zoom_factor();
01462             }
01463             if(example_flags[7]){
01464                 uiOptionsFlag |= UI_Options_Flag_enable_AutoScroll | UI_Options_Flag_enable_AutoZoom;
01465                 Vec2 pacman_dim = {13, 13};
01466                 //Figure *f = new Figure(pacman, pacman_dim);
01467                 std::string pacman_string =
    "5b3o$3b2o3b2o$2bo7bo$bo4b3o2bo$bo4b3o2bo$o11bo$o5b7o$o6bo$bo6bo$bo7bo$2bo7bo$3b2o3b2o$5b3o!";
01468                 auto *f = new Figure(pacman_string, pacman_dim);
01469                 gameOfLife.populate_figure(f,(Vec2){x_pos,y_pos}, angle/90);
01470                 gameOfLife.set_rules(r_copy);
01471                 zoom = gameOfLife.get_grid()->get_auto_zoom_factor();
01472             }
01473 
01474             for(auto c: gameOfLife.get_grid()->get_cells()){
01475                 c->get_neighbours()->set_n_alive(0);
01476                 for(auto n: c->get_neighbours()->get_cells()){
01477                     if(n->get_state()){
    c->get_neighbours()->set_n_alive(c->get_neighbours()->get_n_alive()+1); }
01478                 }
01479                 c->set_color(gameOfLife.get_color_rules()->color[c->get_neighbours()->get_n_alive()]);
```

```
01480             }
01481             reset_figure = Figure(&gameOfLife);
01482             uiFlag &= ~UI_Flag_enable_LoadExampleUI;
01483         }
01484         ImGui::PopStyleColor(3);
01485         ImGui::PopID();
01486         // abort button
01487         ImGui::SameLine();
01488         //color red
01489         ImGui::PushID(0);
01490         ImGui::PushStyleColor(ImGuiCol_Button, (ImVec4)ImColor::HSV(0 / 7.0f, 0.6f, 0.6f));
01491         ImGui::PushStyleColor(ImGuiCol_ButtonHovered, (ImVec4)ImColor::HSV(0 / 7.0f, 0.7f, 0.7f));
01492         ImGui::PushStyleColor(ImGuiCol_ButtonActive, (ImVec4)ImColor::HSV(0 / 7.0f, 0.8f, 0.8f));
01493         if (ImGui::Button("Abort")) {
01494             uiFlag &= ~UI_Flag_enable_LoadExampleUI;
01495         }
01496         ImGui::PopStyleColor(3);
01497         ImGui::PopID();
01498 
01499         ImGui::End();
01500     }
01501 
      //=============================================================================================================
      CLASS UI ===//
01502 
01503 } // gol
```

6.3 game_of_life.h File Reference

```
#include <utility>
#include <vector>
#include <list>
#include <iostream>
#include <fstream>
#include <string>
#include "../ImGUI/imgui.h"
#include "../ImGUI/backends/imgui_impl_sdl2.h"
#include "../ImGUI/backends/imgui_impl_opengl2.h"
#include <stdio.h>
#include <SDL.h>
#include <SDL_opengl.h>
#include <unistd.h>
```

Data Structures

- struct gol::Vec2

  structure defines a 2 dimensional Vector
- struct gol::Rule

  structure for Game of Life Rules.
- struct gol::RuleColor

  structure for Game of Life Color Rules
- class gol::Cell
- class gol::Grid
- class gol::Neighbours
- class gol::Figure
- class gol::GameOfLife
- class gol::UI

Namespaces

- namespace gol

Macros

- #define STACK_SIZE 100

Typedefs

- typedef std::list< Figure * > gol::Figure_Stack

  includes Figures and is used for step back
  Stack size is defined by STACK_SIZE
- typedef unsigned int gol::UI_Flag

  UI_Flags includes information if UI Windows are activated or not.
- typedef unsigned int gol::UI_Options_Flag

  UI_Options_Flag includes some informations about some UI options.

Enumerations

- enum gol::UI_Flag_ {
  gol::UI_Flag_none = 0 , gol::UI_Flag_enable_GridUI = 1<<0 , gol::UI_Flag_enable_SetupUI = 1<<1 ,
  gol::UI_Flag_enable_LoadUI = 1<<2 ,
  gol::UI_Flag_enable_LoadExampleUI = 1<<3 , gol::UI_Flag_enable_SaveUI = 1<<4 , gol::UI_Flag_enable_MessageUI = 1<<5 }

  UI_Flag_ defines Flags for UI_Flag typedef;.
- enum gol::UI_Options_Flag_ {
  gol::UI_Options_Flag_none = 0 , gol::UI_Options_Flag_Run = 1<<0 , gol::UI_Options_Flag_enable_AutoZoom = 1<<2 , gol::UI_Options_Flag_enable_AutoScroll = 1<<3 ,
  gol::UI_Options_Flag_enable_ColorizeAll = 1<<4 }

  UI_Options_Flag_ defines Flags for UI_Options_Flag typedef.

6.3.1 Macro Definition Documentation

6.3.1.1 STACK_SIZE `#define STACK_SIZE 100`

Definition at line 28 of file game_of_life.h.

6.4 game_of_life.h

Go to the documentation of this file.

```
00001 //
00002 // Created by
00003 //          Benjamin Grothe s0580413
00004 //          Juan Jose Arguello Guerra s0580592
00005 // on 06.05.23
00006 //
00007
00008 #include <utility>
00009 #include <vector>
00010 #include <list>
00011 #include <iostream>
00012 #include <fstream>
00013 #include <string>
00014
00015 #include "../ImGUI/imgui.h"
00016 #include "../ImGUI/backends/imgui_impl_sdl2.h"
```

```
00017 #include "../ImGUI/backends/imgui_impl_opengl2.h"
00018 #include <stdio.h>
00019 #include <SDL.h>
00020 #include <SDL_opengl.h>
00021
00022 #ifdef _Win32
00023 #include <Windows.h>
00024 #else
00025 #include <unistd.h>
00026 #endif
00027
00028 #define STACK_SIZE      100
00029
00030 namespace gol {
00031     class Cell;
00032     class Neighbours;
00033     class Grid;
00034     class Figure;
00035     class GameOfLife;
00036     class UI;
00037
00038     // TODO: zoom grid to fit in Grid UI (for exp. Fullscreen)
00039
00040     // TODO: increase example speed ???
00041
00042     // TODO: ERROR HANDLING {get rule from .lif if exist}
00043     // TODO: optimize optical User Interface setup with ImGui
00044
00045
00046     // FIXME: irgendwas stimmt mit dem save_button nicht !!!! random -> screenshot -> speichert nur
      300x300 und nicht 475x300 <- problem liegt nur bei dimensions_to_string
00047
00048     // NOTE: actually only 16bit (max ca. 180x180 cells) !? should be 32bit but how -> uncomment «
      #define ImDrawIdx unsigned int » in imconfig.h
00049     // NOTE: 32Bit works for MacOS X & Linux --> windows not testet
00050
00054     typedef std::list<Figure*> Figure_Stack;
00055
00059     typedef unsigned int UI_Flag;
00063     enum UI_Flag_{
00064         UI_Flag_none = 0,                       // no UI_Flag is set
00065         UI_Flag_enable_GridUI = 1«0,           // UI_Flag which is used for enabling Grid_UI
00066         UI_Flag_enable_SetupUI = 1«1,          // UI_Flag which is used for enabling Setting_UI
00067         UI_Flag_enable_LoadUI = 1«2,           // UI_Flag which is used for enabling Load_UI
00068         UI_Flag_enable_LoadExampleUI = 1«3,    // UI_Flag which is used for enabling Load_Exp_UI
00069         UI_Flag_enable_SaveUI = 1«4,           // UI_Flag which is used for enabling Save_UI
00070         UI_Flag_enable_MessageUI = 1«5         // UI_Flag which is used for enabling Message_UI
00071     };
00072
00077     typedef unsigned int UI_Options_Flag;
00079     enum UI_Options_Flag_{
00080         UI_Options_Flag_none = 0,                   // no UI_Options_Flag is set
00081         UI_Options_Flag_Run = 1«0,                 // UI_Options_Flag which shows if Game of life is
      running
00082         UI_Options_Flag_enable_AutoZoom = 1«2,     // UI_Options_Flag which enabling auto zoom
00083         UI_Options_Flag_enable_AutoScroll = 1«3,   // UI_Options_Flag which enabling auto scroll
00084         UI_Options_Flag_enable_ColorizeAll = 1«4   // UI_Options_Flag which shows if all alive cells
      have the same color
00085     };
00086
00088     struct Vec2 { int x, y; };                      // 2 dimensional Vector
00093     struct Rule { bool alive[9], death[9]; };
00098     struct RuleColor { unsigned int color[9]; };
00099
00100
      //=============================================================================================
      CLASS CELL ===//
00105     class Cell{
00106     private:
00107         Vec2 position;                      // position in grid
00108         bool state;                         // shows if this cell is alive [true] or death [false]
00109         unsigned int color;                 // includes cell Color
00110         Grid* grid;                         // class grid which includes this cell
00111         Neighbours* neighbours;             // class neighbours includes all Cells next to this cell
00112
00113     public:
00119         Cell(Grid* _grid, Vec2 _position);
00120
00122         ~Cell() = default;
00123
00128         Vec2 get_position() const { return position; }
00133         bool get_state() const { return state; }
00138         unsigned int get_color() const {return color; };
00143         Grid* get_grid() const { return grid; }
00148         Neighbours* get_neighbours() const { return neighbours; }
00149
00150         //void set_position(Vec2 _position){ position = _position; }
```

```
00155         void set_state(bool _state){ state = _state; }
00160         void set_color(unsigned int _color){ color = _color; }
00165         void set_neighbours(Neighbours* _neighbours){ neighbours = _neighbours; }
00166
00167     };
00168
      //=============================================================================================
      CLASS GRID ===//
00173     class Grid{
00174     private:
00175         Vec2 dimension;             // dimension of this grid
00176         std::vector<Cell*> cells;   // includes all existing cells in this grid
00177         GameOfLife* gameOfLife;     // owner GameOfLife object [unused]
00178
00183         Vec2 get_real_grid_position();
00184
00185     public:
00191         Grid(GameOfLife* _gameOfLife, Vec2 _dimension);
00193         ~Grid() = default;
00194
00195         Vec2 get_dimension() const { return dimension; }
00200         std::vector<Cell*> get_cells() { return cells;}
00206         Cell* get_cell(Vec2 position) { return cells[((position.y* this->dimension.x)+position.x)]; }
00207         //GameOfLife* get_gameOfLife(){ return gameOfLife; }
00208
00214         Vec2 get_real_grid_dimension();
00219         float get_scrollx_position();
00224         float get_scrolly_position();
00229         float get_auto_zoom_factor();
00230
00231     };
00232     //=============================================================================================
      CLASS NEIGHBOURS ===//
00237     class Neighbours{
00238     private:
00239         Cell* owner;                // cell which owns this object
00240         std::vector<Cell*> cells;   // includes all cells next to owner
00241
00242         int n_alive;                // shows number of cells with state true
00243
00244     public:
00249         explicit Neighbours(Cell* _owner);
00251         ~Neighbours() = default;
00252
00257         std::vector<Cell*> get_cells() const { return cells; }
00258         //Cell* get_owner() { return owner; }
00263         int get_n_alive() const { return n_alive; }
00268         void set_n_alive(int _n) { n_alive = _n; }
00269     };
00270     //=============================================================================================
      CLASS FIGURE ===//
00276     class Figure{
00277     private:
00278         Vec2 dimensions;            // dimensions of this object
00279         std::vector<bool> states;   // all states of this object
00280
00286         static Vec2 convert_string_to_dimensions(std::string str);
00293         std::vector<bool> convert_string_to_states(std::string str);
00294
00296         std::string convert_dimension_to_string();
00298         std::string convert_states_to_string();
00299
00300
00301
00302     public:
00308         explicit Figure(std::vector<bool> _states, Vec2 &_dimensions){dimensions = _dimensions; states
      = std::move(_states);  }
00314         explicit Figure(std::string &str, Vec2 &_dimensions){ dimensions = _dimensions; states =
      convert_string_to_states(str); }
00319         explicit Figure(std::string &path);
00324         explicit Figure(GameOfLife* gameOfLife);
00326         ~Figure() = default;
00327
00332         Vec2 get_dimension() const { return dimensions; }
00333         //std::vector<bool> get_states() const { return states; } // unused
00339         bool get_state(Vec2 _position) const { return states[(_position.y*dimensions.x)+_position.x];
      }
00340
00346         int save_to_file(std::string &path);
00347
00348     };
00349     //============================================================================== CLASS
      GAME OF LIFE ===//
00354     class GameOfLife{
00355     private:
00356         Grid* grid;                 // grid object with cells
00357         Rule rules;                 // includes rules for game of life
```

```
00358         RuleColor rules_color;      // includes color rules for game of life
00359
00360
00366         bool check_rules_in_cell(Cell* _cell);
00367
00369         void check_rules_in_grid();
00370
00372         static bool random_bool();
00373
00374     public:
00380         GameOfLife(Vec2 _dimensions, Rule _rules, RuleColor _rulesColor);
00382         ~GameOfLife() = default;
00383
00384         Grid* get_grid(){ return grid; }
00385         Rule* get_rules(){ return &rules; }
00386         RuleColor* get_color_rules(){ return &rules_color; }
00387
00388         void set_rules(Rule rule){ rules = rule; }
00389
00394         int run();
00401         void populate_figure(Figure* _figure, Vec2 position, int angle);
00403         void populate_random();
00405         void refresh_grid(){ check_rules_in_grid(); } // FIXME: its to slow ( ca  « 9(x*y)++ » loops
      on 1 core ) <- Optimazion flag -O1 !! <- eventually save only alive cells in grid calculate neighbours
      and change them "Stack style"
00407         void clear_grid();
00408
00409
00410     };
00411
      //=============================================================================================================
      CLASS UI ===//
00415     class UI{
00416     private:
00417
00423         static void print(GameOfLife* _gameOfLife, float zoom);
00429         static void button_setting_zoom(UI_Options_Flag &uiOptionsFlag, float &factor);
00434         static void button_setting_speed(int &factor);
00440         static void button_setting_step(GameOfLife &gameOfLife, Figure_Stack &_stack);
00445         static void button_setting_run(UI_Options_Flag &uiOptionsFlag);
00450         //static void button_setting_stop(UI_Options_Flag &uiOptionsFlag);
00457         static void button_setting_reset(GameOfLife &gameOfLife, Figure &reset_figure, Figure_Stack
      &figureStack);
00464         static void button_setting_clear(GameOfLife &gameOfLife, Figure &reset_figure, Figure_Stack
      &figureStack);
00471         static void button_setting_random(GameOfLife &gameOfLife, Figure &reset_figure, Figure_Stack
      &figureStack);
00476         static void button_setting_load(UI_Flag &uiFlag);//TODO
00481         static void button_setting_load_exp(UI_Flag &uiFlag);
00486         static void button_setting_screenshot(UI_Flag &uiFlag);
00491         static void button_setting_reset_rules(Rule* _rule);
00496         static void button_setting_reset_colors(RuleColor* _rule);
00502         static void setup_rules(UI_Options_Flag &uiOptionsFlag, GameOfLife &gameOfLife);
00514         static void button_load(UI_Flag &uiFlag, UI_Options_Flag &uiOptionsFlag, GameOfLife
      &gameOfLife, Figure &reset_figure, std::string path, Vec2 position, int &angle, std::string &message,
      float &zoom);
00521         static void button_save(UI_Flag &uiFlag, GameOfLife &gameOfLife, std::string path);
00522
00523     public:
00533         static void grid_UI(UI_Flag &uiFlag, UI_Options_Flag &uiOptionsFlag, GameOfLife &gameOfLife,
      Figure_Stack &figureStack, float &zoom, int &speed, int &count);
00546         static void setting_UI(UI_Flag &uiFlag, UI_Options_Flag &uiOptionsFlag, GameOfLife
      &gameOfLife, Figure &reset_figure, Figure_Stack &figureStack,float &zoom, int &speed);
00557         static void load_UI(UI_Flag &uiFlag, UI_Options_Flag &uiOptionsFlag, GameOfLife &gameOfLife,
      Figure &reset_figure, std::string &message, float &zoom);
00564         static void screenshot_UI(UI_Flag &uiFlag, GameOfLife &gameOfLife, std::string &message);
00570         static void messageBox_UI(UI_Flag &uiFlag, std::string &message);
00577         static void ColorPicker(GameOfLife &gameOfLife, const char* label, ImU32 *color);
00578
00588         static void load_example_UI(UI_Flag &uiFlag, UI_Options_Flag &uiOptionsFlag, GameOfLife
      &gameOfLife, Figure &reset_figure, float &zoom, int &speed);
00589
00593         static void Dockspace_UI();
00594     };
00595 } // gol
00596
```