



## Game of Life

Lothar Gomoluch, Oliver Röckener und Niko Tepe

Version 1.02

Montag, 22.06.2020

# File Index

## File List

Here is a list of all documented files with brief descriptions:

<b>buffer.h</b>	.....
<b>game.h</b>	.....
<b>main.c</b>	.....
<b>menu.h</b>	.....

# Data Structure Documentation

## cell Struct Reference

### Data Fields

int **alive**  
int **livingNeighbors**  
struct cell \* **neighborCell** [8]

## menu\_button Struct Reference

### Data Fields

char **label** [20]  
COORD **pos**

## settings Struct Reference

### Data Fields

COORD **gridsize**  
char **symbolAlive**  
char **symbolDead**  
int **generationsToCalc**  
int **iterationsPerSecond**  
COORD **hud\_currentGeneration\_pos**  
COORD **hud\_gridSize\_pos**  
COORD **hud\_generationsToCalc\_pos**  
COORD **hud\_iterationsPerSecond\_pos**  
COORD **hud\_aliveCells\_pos**  
COORD **hud\_shortcutInfo\_pos**

# File Documentation

## main.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "include/menu.h"
#include "include/game.h"
#include "include/buffer.h"
```

## Functions

void **run** (int generationsToCalc, int ticksPerSecond)  
void **tick** (int \*end\_game, int \*pause\_game)  
*Diese Funktion berechnet die nächste Iteration des GOL.*

void **settings\_menu** ()  
*generiert ein Untermenü: Settingsmenu*

void **main\_menu** ()  
*generiert das Hauptmenü*

void **init\_settings** ()  
*initialisiert Grundwerte für die Programmausführung*

int **main** ()  
void **start\_game** (int is\_random)  
*diese Funktion startet ein Spiel*

void **start\_menu** ()  
*generiert ein Untermenü: startmenu*

void **rule\_menu** ()  
*generiert ein Untermenü: Rulemenu*

## Variables

```
struct settings gamesettings
struct rule gamerules
struct cell ** grid
struct cell ** gridcopy
char * buffer
struct menu_button mainMenu_Button [4]
struct menu_button startMenu_Button [2]
struct menu_button settingsMenu_Button [6]
struct menu_button ruleMenu_Button [3]
int aliveCells = 0
int aliveCellsPrevGen = 0
int currentGeneration = 0
```

```
int iterationsSinceLastChange = 0
int runtime_start = 0
```

## Function Documentation

### **void init\_settings ()**

initialisiert Grundwerte für die Programmausführung

### **void main\_menu ()**

generiert das Hauptmenü

### **void rule\_menu ()**

generiert ein Untermenü: Rulemenu

### **void run (int *generationsToCalc*, int *ticksPerSecond*)**

führt den Kern des Spiels aus

#### **Parameters**

<i>generationsToCalc</i>	
<i>ticksPerSecond</i>	

### **void settings\_menu ()**

generiert ein Untermenü: Settingsmenu

### **void start\_game (int *is\_random*)**

diese Funktion startet ein Spiel

#### **Parameters**

<i>is_random</i>	legt fest ob das Spiel zufällig generiert sein soll
------------------	---

### **void start\_menu ()**

generiert ein Untermenü: startmenu

### **void tick (int \* *end\_game*, int \* *pause\_game*)**

Diese Funktion berechnet die nächste Iteration des GOL.

#### **Parameters**

<i>end_game</i>	Verweis auf Int Value
<i>pause_game</i>	Verweis auf Int Value

## game.h File Reference

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <windows.h>
#include "menu.h"
```

## Data Structures

struct **cell**  
struct **settings**

## Functions

uint32\_t **generate\_random\_int\_msws** ()

*Generiert eine Middle Square Weyl Sequence um daraus eine Randomzahl zu generieren.  
<https://pthree.org/2018/07/30/middle-square-weyl-sequence-prng/>  
Diese Funktion wird verwendet, da so eine bessere Zufallszahlgenerierung gewährleistet wird.*

void **alloc\_grid** (struct **cell** \*\*\*grid\_ptr, COORD gridsize)

*alloc\_grid reserviert den Speicher des Feldes struct cell \*\*\*grid\_ptr -> deklariere "grid" als pointer to pointer to pointer to struct cell*

void **dealloc\_grid** (struct **cell** \*\*\*grid\_ptr, const int x\_size)

*dealloc\_grid gibt den Speicher des Feldes wieder frei struct cell \*\*\*grid\_ptr -> deklariere "grid" als pointer to pointer to pointer to struct cell*

void **copy\_grid** (struct **cell** \*\*grid\_ptr\_dest, struct **cell** \*\*grid\_ptr\_src, COORD gridsize)

*copy\_grid kopiert alle Werte eines Feldes in ein neues Feld*

void **load\_preset\_to\_grid** (struct **cell** \*\*grid\_ptr, COORD gridsize)

*load\_preset\_to\_grid laed ein Preset aus einer Textdatei.*

void **save\_preset\_from\_grid** (struct **cell** \*\*grid\_ptr, COORD gridsize)

*save\_preset\_to\_grid speichert den Stand des Feldes als Preset in eine Textdatei.*

int **count\_living\_neighbors** (struct **cell** grid)

*count\_living\_neighbors zählt die lebenden umliegenden Nachbarn. diese Funktion ist möglichst effizient geschrieben, da sie beim initialisieren eines 1000x1000 Feldes bis zu 1.000.000x aufgerufen wird.*

void **define\_neighborhood** (struct **cell** \*\*grid\_ptr, COORD gridsize)

*define\_neighborhood definiert alle anliegenden Nachbarzellen für jede Zelle eines Feldes*

void **add\_neighborhood** (struct **cell** grid)

*add\_neighborhood informiert jede anliegende Nachbarzelle über den aktuellen Stand der Zelle das passiert durch das Hochzählen von livingNeighbors in jeder anliegenden Zelle diese Funktion ist möglichst effizient geschrieben, da sie bei einem 1000x1000 Feld bis zu 1.000.000x pro Tick aufgerufen werden kann.*

void **sub\_neighborhood** (struct **cell** grid)

*sub\_neighborhood* informiert jede anliegende Nachbarzelle über den aktuellen Stand der Zelle das passiert durch das Runterzählen von *livingNeighbors* in jeder anliegenden Zelle diese Funktion ist möglichst effizient geschrieben, da sie bei einem 1000x1000 Feld bis zu 1.000.000x pro Tick aufgerufen werden kann.

void **initialize\_empty\_grid** (struct **cell** \*\*grid\_ptr, COORD gridsize)  
*initialize\_empty\_grid* initialisiert ein leeres grid

void **calc\_all\_neighbors** (struct **cell** \*\*grid\_ptr, COORD gridsize)  
*calc\_all\_neighbors* initialisiert die *livingNeighbors* für jede Zelle.

void **generate\_random\_grid** (struct **cell** \*\*grid\_ptr, COORD gridsize)  
*generate\_random\_grid* initialisiert das übergebene Feld mit dem Modulo 2 (%2) aus einer zufälligen Zahl für jede Zelle eines Feldes

## Function Documentation

void **add\_neighborhood** (struct **cell** *grid*)

*add\_neighborhood* informiert jede anliegende Nachbarzelle über den aktuellen Stand der Zelle das passiert durch das Hochzählen von *livingNeighbors* in jeder anliegenden Zelle diese Funktion ist möglichst effizient geschrieben, da sie bei einem 1000x1000 Feld bis zu 1.000.000x pro Tick aufgerufen werden kann.

### Parameters

<i>cell</i>	Verweis auf eine Zelle im Spielfeld
-------------	-------------------------------------

void **alloc\_grid** (struct **cell** \*\*\* *grid\_ptr*, COORD *gridsize*)

*alloc\_grid* reserviert den Speicher des Feldes struct cell \*\*\**grid\_ptr* -> deklariere "grid" als pointer to pointer to pointer to struct cell

### Parameters

<i>grid_ptr</i>	Verweis auf das Grid
<i>gridsize</i>	Größe des Spielfeldes

void **calc\_all\_neighbors** (struct **cell** \*\* *grid\_ptr*, COORD *gridsize*)

*calc\_all\_neighbors* initialisiert die *livingNeighbors* für jede Zelle.

### Parameters

<i>grid_ptr</i>	Verweis auf das Grid
<i>gridsize</i>	Größe des Spielfeldes

**void copy\_grid (struct cell \*\* *grid\_ptr\_dest*, struct cell \*\* *grid\_ptr\_src*, COORD *gridsize*)**

copy\_grid kopiert alle Werte eines Feldes in ein neues Feld

#### Parameters

<i>grid_ptr_dest</i>	Verweis auf das Ziel Grid
<i>grid_ptr_src</i>	Verweis auf das Grid von dem kopiert werden soll
<i>gridsize</i>	Größe des Spielfeldes

**int count\_living\_neighbors (struct cell *grid*)**

count\_living\_neighbors zählt die lebenden umliegenden Nachbarn. diese Funktion ist möglichst effizient geschrieben, da sie beim initialisieren eines 1000x1000 Feldes bis zu 1.000.000x aufgerufen wird.

#### Parameters

<i>cell</i>	Verweis auf eine Zelle im Spielfeld
-------------	-------------------------------------

#### Returns

int Anzahl der lebenden Nachbarn

**void dealloc\_grid (struct cell \*\*\* *grid\_ptr*, const int *x\_size*)**

dealloc\_grid gibt den Speicher des Feldes wieder frei struct cell \*\*\*grid\_ptr -> deklariere "grid" als pointer to pointer to pointer to struct cell

#### Parameters

<i>grid_ptr</i>	Verweis auf das Grid
<i>x_size</i>	Größe der X-Achse des Spielfeldes

**void define\_neighborhood (struct cell \*\* *grid\_ptr*, COORD *gridsize*)**

define\_neighborhood definiert alle anliegenden Nachbarzellen für jede Zelle eines Feldes

#### Parameters

<i>grid_ptr</i>	Verweis auf das Grid
<i>gridsize</i>	Größe des Spielfeldes

**void generate\_random\_grid (struct cell \*\* *grid\_ptr*, COORD *gridsize*)**

generate\_random\_grid initialisiert das übergebene Feld mit dem Modulo 2 (%2) aus einer zufälligen Zahl für jede Zelle eines Feldes

#### Parameters

<i>grid_ptr</i>	Verweis auf das Grid
<i>gridsize</i>	Größe des Spielfeldes

**uint32\_t generate\_random\_int\_msws ()**

Generiert eine Middle Square Weyl Sequence um daraus eine Randomzahl zu generieren.

#### Returns

uint32\_t gibt einen unsigned 32bit Integer zurück

**void initialize\_empty\_grid (struct cell \*\* *grid\_ptr*, COORD *gridsize*)**

initialize\_empty\_grid initialisiert ein leeres grid

#### Parameters

<i>grid_ptr</i>	Verweis auf das Grid
<i>gridsize</i>	Größe des Spielfeldes

**void load\_preset\_to\_grid (struct cell \*\* *grid\_ptr*, COORD *gridsize*)**

load\_preset\_to\_grid laed ein Preset aus einer Textdatei.

#### Parameters

<i>grid_ptr</i>	Verweis auf das Grid
<i>gridsize</i>	Größe des Spielfeldes

**void save\_preset\_from\_grid (struct cell \*\* *grid\_ptr*, COORD *gridsize*)**

save\_preset\_to\_grid speichert den Stand des Feldes als Preset in eine Textdatei.

#### Parameters

<i>grid_ptr</i>	Verweis auf das Grid
<i>gridsize</i>	Größe des Spielfeldes

**void sub\_neighborhood (struct cell *grid*)**

sub\_neighborhood informiert jede anliegende Nachbarzelle über den aktuellen Stand der Zelle das passiert durch das Runterzählen von livingNeighbors in jeder anliegenden Zelle diese Funktion ist möglichst effizient geschrieben, da sie bei einem 1000x1000 Feld bis zu 1.000.000x pro Tick aufgerufen werden kann.

#### Parameters

<i>cell</i>	Verweis auf eine Zelle im Spielfeld
-------------	-------------------------------------



## menu.h File Reference

```
#include <windows.h>
#include <string.h>
#include "game.h"
```

### Data Structures

struct **menu\_button**

### Functions

void **print\_logo** (int x, int y)

*gibt das Spiel-Logo an Position x,y aus*

void **set\_cursor** (int x, int y)

*Setzt den Consolecursor an eine Position im Consolenbuffer.*

void **set\_fontsize** (int size)

*Setzt die Consolenfontsize.*

void **draw\_menu** (struct **menu\_button** Menu\_Button[3], int array\_length)

*Zeichnet ein Menu anhand von Buttons.*

void **draw\_settings\_menu\_values** (struct **menu\_button** Menu\_Button[3], int array\_length, struct **settings** gamesettings)

*Zeichnet die Werte im Settingsmenü*

void **draw\_rule\_menu\_values** (struct **menu\_button** Menu\_Button[3], int array\_length, struct rule gamerules)

*Zeichnet die Werte im Rulemenü*

void **edit\_setting\_value** (struct **settings** \*gamesettings, struct **menu\_button** Menu\_Button[5], int cursor\_pos)

*Initialisiert eine Scanf-Sequenz im Settingsmenü um einen Wert anzupassen.*

void **edit\_rule\_value** (struct rule \*gamerules, struct **menu\_button** Menu\_Button[5], int cursor\_pos)

*Initialisiert eine Scanf-Sequenz im Rulesmenü um einen Wert anzupassen.*

void **set\_value\_cursor** (struct **menu\_button** Menu\_Button[5], int cursor\_pos)

*Bewegt den Cursor im Settingsmenü an eine übergebene Stelle.*

void **set\_menucursor** (struct **menu\_button** Menu\_Button[3], int array\_length, int position)

*Bewegt den Cursor im Mainmenü an eine übergebene Stelle.*

void **draw\_cursor** (COORD cords)

*Zeichnet den Menücursor an einer bestimmte Stelle.*

void **erase\_cursor** (COORD cords)

*Löscht den Menücursor an einer bestimmte Stelle.*

COORD **get\_console\_window\_size** (HANDLE hConsoleOutput)  
*Get the console window size object.*

void **set\_console\_fullscreen** ()  
*Setzt die Console auf Vollbild (so wie alt+enter)*

## Function Documentation

void **draw\_cursor** (COORD **cords**)

Zeichnet den Menücursor an einer bestimmte Stelle.

### Parameters

<i>cords</i>	Coords der Position
--------------	---------------------

void **draw\_menu** (struct menu\_button **Menu\_Button**[3], int **array\_length**)

Zeichnet ein Menu anhand von Buttons.

### Parameters

<i>Menu_Button</i>	Array mit Menubuttons
<i>array_length</i>	Arraylänge

void **draw\_rule\_menu\_values** (struct menu\_button **Menu\_Button**[3], int **array\_length**, struct rule **gamerules**)

Zeichnet die Werte im Rulemenü

### Parameters

<i>Menu_Button</i>	Array mit Menubuttons
<i>array_length</i>	Arraylänge
<i>gamerules</i>	Gamerule Struct

void **draw\_settings\_menu\_values** (struct menu\_button **Menu\_Button**[3], int **array\_length**, struct settings **gamesettings**)

Zeichnet die Werte im Settingsmenü

### Parameters

<i>Menu_Button</i>	Array mit Menubuttons
<i>array_length</i>	Arraylänge
<i>gamesettings</i>	Gamesettings Struct

void **edit\_rule\_value** (struct rule \* **gamerules**, struct menu\_button **Menu\_Button**[5], int **cursor\_pos**)

Initialisiert eine Scanf-Sequenz im Rulesmenü um einen Wert anzupassen.

### Parameters

<i>gamerules</i>	Gamesrules Struct
<i>Menu_Button</i>	Array mit Menubuttons
<i>cursor_pos</i>	Position des Cursors

**void edit\_setting\_value (struct settings \* *gamesettings*, struct menu\_button *Menu\_Button*[5], int *cursor\_pos*)**

Initialisiert eine Scanf-Sequenz im Settingsmenü um einen Wert anzupassen.

#### Parameters

<i>gamesettings</i>	Gamesettings Struct
<i>Menu_Button</i>	Array mit Menubuttons
<i>cursor_pos</i>	Position des Cursors

**void erase\_cursor (COORD *cords*)**

Löscht den Menücursor an einer bestimmte Stelle.

#### Parameters

<i>cords</i>	Coords der Position
--------------	---------------------

**COORD get\_console\_window\_size (HANDLE *hConsoleOutput*)**

Get the console window size object.

#### Parameters

<i>hConsoleOutput</i>	Console Handle
-----------------------	----------------

#### Returns

COORD Größe der Console als Coord-Objekt

**void print\_logo (int *x*, int *y*)**

gibt das Spiel-Logo an Position x,y aus

#### Parameters

<i>x</i>	X-Position im Consolenbuffer
<i>y</i>	Y-Position im Consolenbuffer

**void set\_console\_fullscreen ()**

Setzt die Console auf Vollbild (so wie alt+enter)

**void set\_cursor (int *x*, int *y*)**

Setzt den Consolecursor an eine Position im Consolenbuffer.

#### Parameters

<i>x</i>	X-Position im Consolenbuffer
<i>y</i>	Y-Position im Consolenbuffer

**void set\_fontsize (int *size*)**

Setzt die Consolenfontsize.

#### Parameters

<i>size</i>	Schriftgröße
-------------	--------------

**void set\_menucursor (struct menu\_button *Menu\_Button*[3], int *array\_length*, int *position*)**

Bewegt den Cursor im Mainmenü an eine übergebene Stelle.

#### Parameters

<i>Menu_Button</i>	Array mit Menubuttons
<i>array_length</i>	Arraylänge
<i>position</i>	Position des Cursors

**void set\_value\_cursor (struct menu\_button *Menu\_Button*[5], int *cursor\_pos*)**

Bewegt den Cursor im Settingsmenü an eine übergebene Stelle.

#### Parameters

<i>Menu_Button</i>	Array mit Menubuttons
<i>cursor_pos</i>	Position des Cursors

## buffer.h File Reference

```
#include <stdint.h>
#include <stdlib.h>
#include "menu.h"
#include "game.h"
```

## Functions

void **alloc\_buffer** (char \*\*buffer, COORD gridsize)

*aloziiert den Buffer*

void **dealloc\_buffer** (char \*buffer)

*dealoziiert den Buffer*

void **initialize\_buffer** (char \*buffer, COORD gridsize, char symbolAlive, char symbolDead)

*initialisiert den Buffer*

void **revive\_buffer\_at\_coord** (char \*buffer, struct **settings** gamesettings, int x\_pos, int y\_pos)

*Diese Funktion belebt eine Zelle an bestimmter Stelle im Buffer wieder.*

void **kill\_buffer\_at\_coord** (char \*buffer, struct **settings** gamesettings, int x\_pos, int y\_pos)

*Diese Funktion tötet eine Zelle an bestimmter Stelle im Buffer.*

int **calc\_buffer\_size** (COORD gridsize)

*Berechnet die Speichergröße des Buffers.*

int **calc\_position\_in\_buffer** (COORD gridsize, const int x\_pos, const int y\_pos)

*Berechnet die Position einer Zelle im Buffer anhand der Position auf dem Spielfeld.*

void **draw\_hud** (struct **settings** gamesettings, int aliveCells, int currentGeneration, int generationsToCalc)

*Zeichnet das Head-Up-Display im Spiel.*

void **print\_buffer** (char \*buffer)

*Gibt den Buffer aus.*

## Function Documentation

**void alloc\_buffer (char \*\* *buffer*, COORD *gridsize*)**

aloziiert den Buffer

### Parameters

<i>buffer</i>	Verweis auf den Buffer
<i>gridsize</i>	Größe des Spielfeldes

**int calc\_buffer\_size (COORD *gridsize*)**

Berechnet die Speichergröße des Buffers.

### Parameters

<i>gridsize</i>	Größe des Spielfeldes
-----------------	-----------------------

### Returns

int Gibt die Speichergröße zurück

**int calc\_position\_in\_buffer (COORD *gridsize*, const int *x\_pos*, const int *y\_pos*)**

Berechnet die Position einer Zelle im Buffer anhand der Position auf dem Spielfeld.

### Parameters

<i>gridsize</i>	Größe des Spielfeldes
<i>x_pos</i>	Position X auf dem Spielfeld
<i>y_pos</i>	Position Y auf dem Spielfeld

### Returns

int Gibt die Position im Buffer zurück

**void dealloc\_buffer (char \* *buffer*)**

dealoziiert den Buffer

### Parameters

<i>buffer</i>	Verweis auf den Buffer
---------------	------------------------

**void draw\_hud (struct settings *gamesettings*, int *aliveCells*, int *currentGeneration*, int *generationsToCalc*)**

Zeichnet das Head-Up-Display im Spiel.

### Parameters

<i>gamesettings</i>	Größe des Spielfeldes
<i>aliveCells</i>	Anzahl lebender Zellen
<i>currentGeneration</i>	Nummer der aktuellen Generation
<i>generationsToCalc</i>	Noch zu berechnende Generationen

**void initialize\_buffer (char \* *buffer*, COORD *gridsize*, char *symbolAlive*, char *symbolDead*)**

initialisiert den Buffer

#### Parameters

<i>buffer</i>	Verweis auf den Buffer
<i>gridsize</i>	Größe des Spielfeldes
<i>symbolAlive</i>	Symbol für lebende Zellen
<i>symbolDead</i>	Symbol für tote Zellen

**void kill\_buffer\_at\_coord (char \* *buffer*, struct settings *gamesettings*, int *x\_pos*, int *y\_pos*)**

Diese Funktion tötet eine Zelle an bestimmter Stelle im Buffer.

#### Parameters

<i>buffer</i>	Verweis auf den Buffer
<i>gamesettings</i>	Gamesettings Struct
<i>x_pos</i>	Position X auf dem Spielfeld
<i>y_pos</i>	Position Y auf dem Spielfeld

**void print\_buffer (char \* *buffer*)**

Gibt den Buffer aus.

#### Parameters

<i>buffer</i>	Verweis auf den Buffer
---------------	------------------------

**void revive\_buffer\_at\_coord (char \* *buffer*, struct settings *gamesettings*, int *x\_pos*, int *y\_pos*)**

Diese Funktion belebt eine Zelle an bestimmter Stelle im Buffer wieder.

#### Parameters

<i>buffer</i>	Verweis auf den Buffer
<i>gamesettings</i>	Gamesettings Struct
<i>x_pos</i>	Position X auf dem Spielfeld
<i>y_pos</i>	Position Y auf dem Spielfeld