ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И

ИНФОРМАТИКИ»

Кафедра вычислительных систем

КУРСОВАЯ РАБОТА

по дисциплине «Технологии разработки программного обеспечения»

на тему "Игра Жизнь"

Выполнил: Хотылев Александр Сергеевич

Группа: ИВ-621

Проверил: Пудов С. Г.

Новосибирск, 2017

Содержание

[ВВЕДЕНИЕ И ПОСТАНОВКА ЗАДАЧИ 3](#_Toc482440476)

[ТЕХНИЧЕСКОЕ ЗАДАНИЕ 4](#_Toc482440477)

[ОПИСАНИЕ ВЫПОЛНЕННОГО ПРОЕКТА 5](#_Toc482440478)

[ЛИЧНЫЙ ВКЛАД В ПРОЕКТ 6](#_Toc482440479)

[ПРИЛОЖЕНИЕ. ТЕКСТ ПРОГРАММЫ 7](#_Toc482440480)

# ВВЕДЕНИЕ И ПОСТАНОВКА ЗАДАЧИ

В курсовой работе необходимо разработать многофайловое приложение, работая в команде и используя систему контроля версий Git. Язык написания программы — С. Среда разработки — Блокнот + командная строка linux. Основное требование к проекту: законченный проект, в виде консольного приложения, демонстрирующий клеточный автомат с возможностью поэтапного просмотра итераций и заданием собственных настроек поля (символ для живой и мертвой клеток, размер поля (x, y), состояния клеток (жива или мертва) в виде матрицы). Проект должен быть покрыт тестами и работать на Linux и Windows системах.

# ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Conway's Game of Life (Игра “жизнь”) - клеточный автомат, придуманный английским математиком Джоном Конвеем в 1970 году. Данная игра.

Поле ограничено и может изменяться только при помощи команда “read” и названия файла, в котором будут записаны размеры поля. Клетка может быть только в двух состояниях – жива(1) или мертва(0). Алгоритм был выбран “простейший” - со сменой поколений. Просматриваются все клетки поля и для каждой подсчитывается количество соседей, определяя ее судьбу (выживет, умрет или не изменится).

Данный алгоритм использует два массива структур типа Cell.

В программе используются команды : step (один игровой шаг), loop [num] (циклический step с опициональным количеством итераций), help (список и пояснение всех команд), calc (вычисление одного игрового шага), print (вывод матрицы в консоль), clear (очистка консоли), write (запись матрицы в файл с ее данными параметрами), read (чтение файла настроек), exit или quit (завершение программы) и next (исполнение предыдущей команды).

# ОПИСАНИЕ ВЫПОЛНЕННОГО ПРОЕКТА

1. **Функционал**.

Запуск программы: make && bin/main

Каждое действие отдается в руки пользователя.

1. **Команды**.

help : displays help.

next : repeats the previous command.

write [file\_name] : writes the matrix to a file (you must specify a file name)

read [file\_name] : reads the file containing the matrix and its settings (you must specify a file name)

loop [repeat\_num] [delay] : repeats "calc" and "print", "repeat\_num" times and with a "delay" in sec, delay is maybe float, repeat\_num can be zero then the loop will be infinite

calc : makes one game step.

print : print matrix in console.

step : makes one game step and print matrix in console.

clear : clear console.

exit : completes the game.

1. **Рекомендуемая последовательность команд для просмотра заготовленной матрицы в игре**:

*в системе Linux:*

@read settings

@loop 0

*в системе Windows*:

@read settings

@loop

1. Структура проекта

*.*

├── bin

│└── main

├── build

│└── src

│├── cell.o

│├── main.o

│└── matrix.o

├── Makefile

├── README.md

├── settings

├── src

│├── cell.c

│├── cell.h

│├── main.c

│├── matrix.c

│└── matrix.h

├── tests

│├── cell\_test.c

│├── main.c

│└── matrix\_test.c

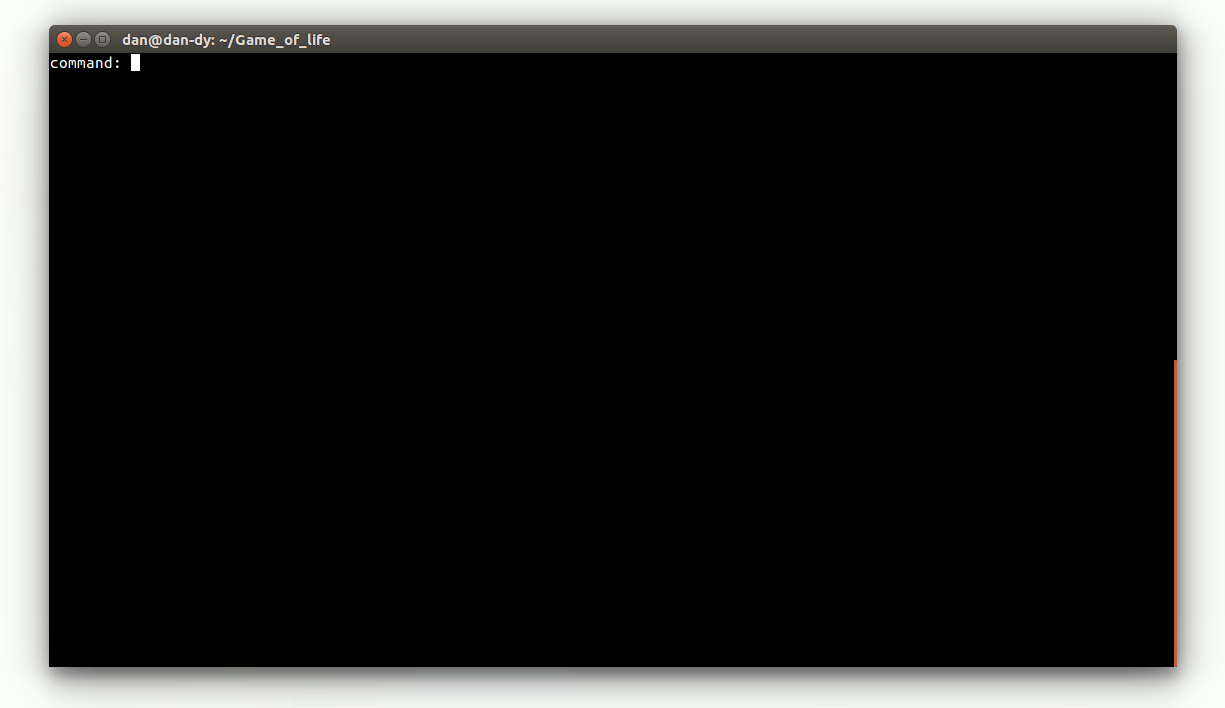
└── thirdparty

└── ctest.h

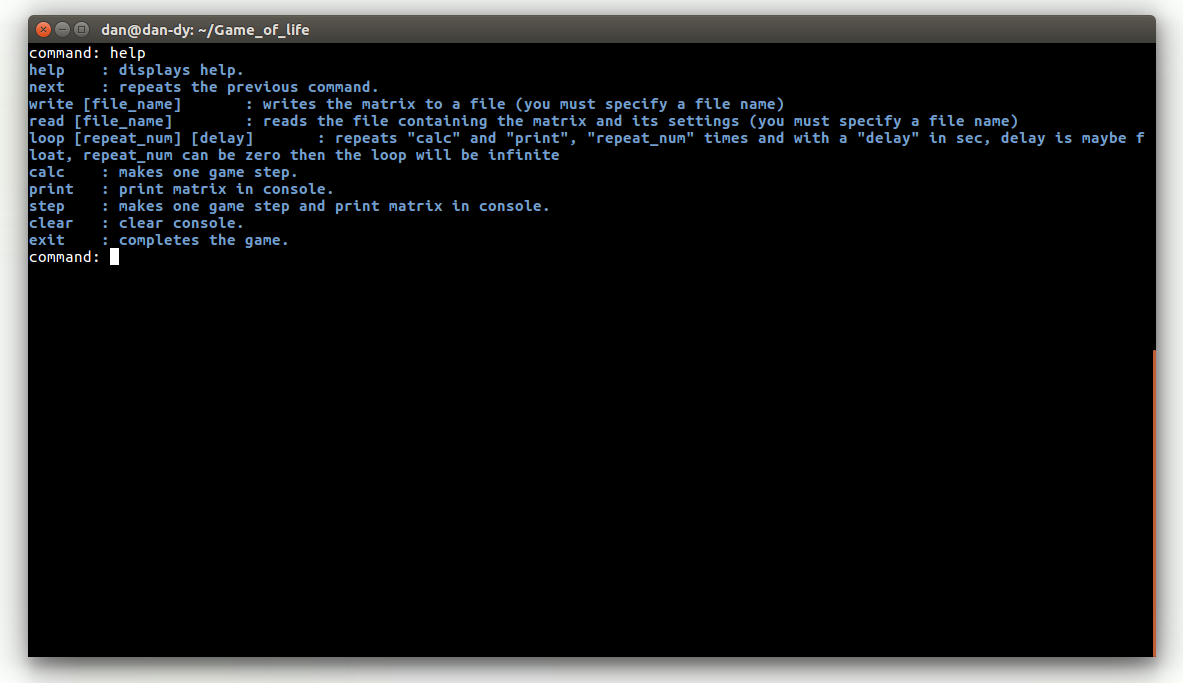
7 directories, 18 files

1. Визуальная составляющая проекта:

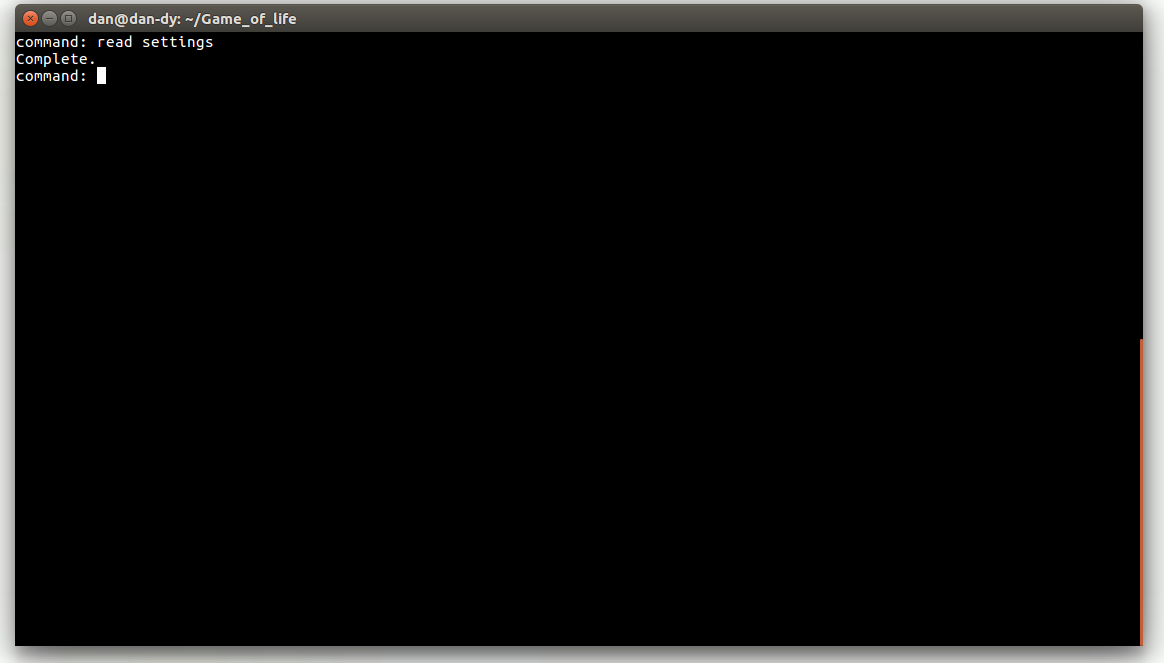
начало



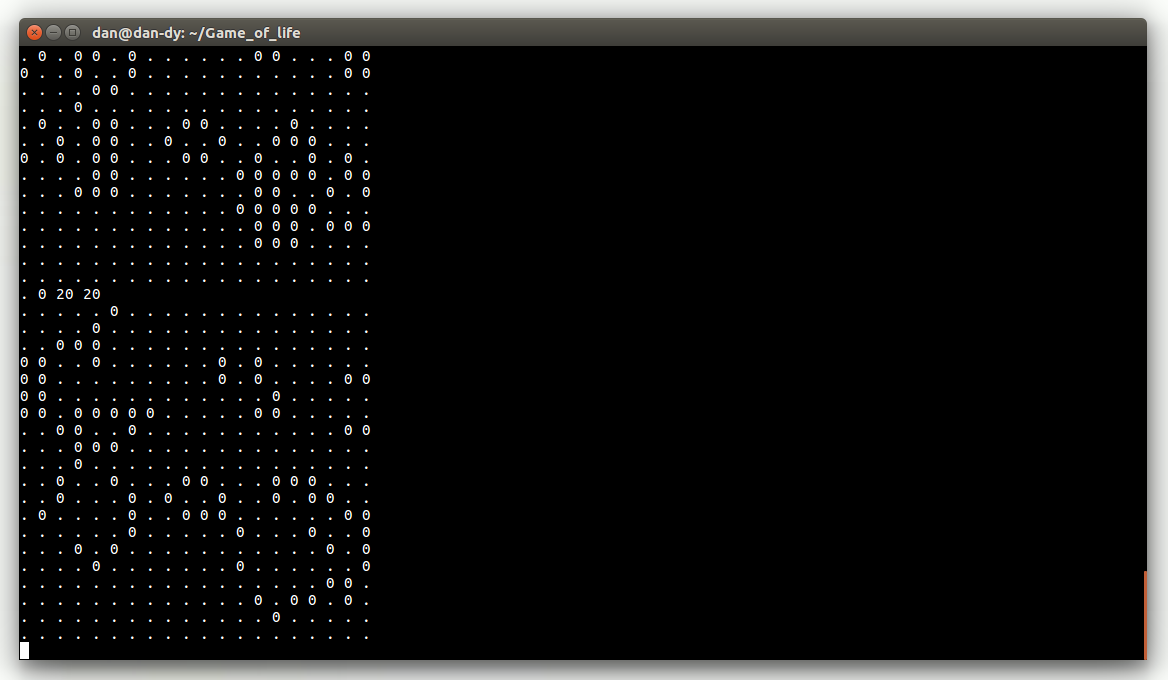
help



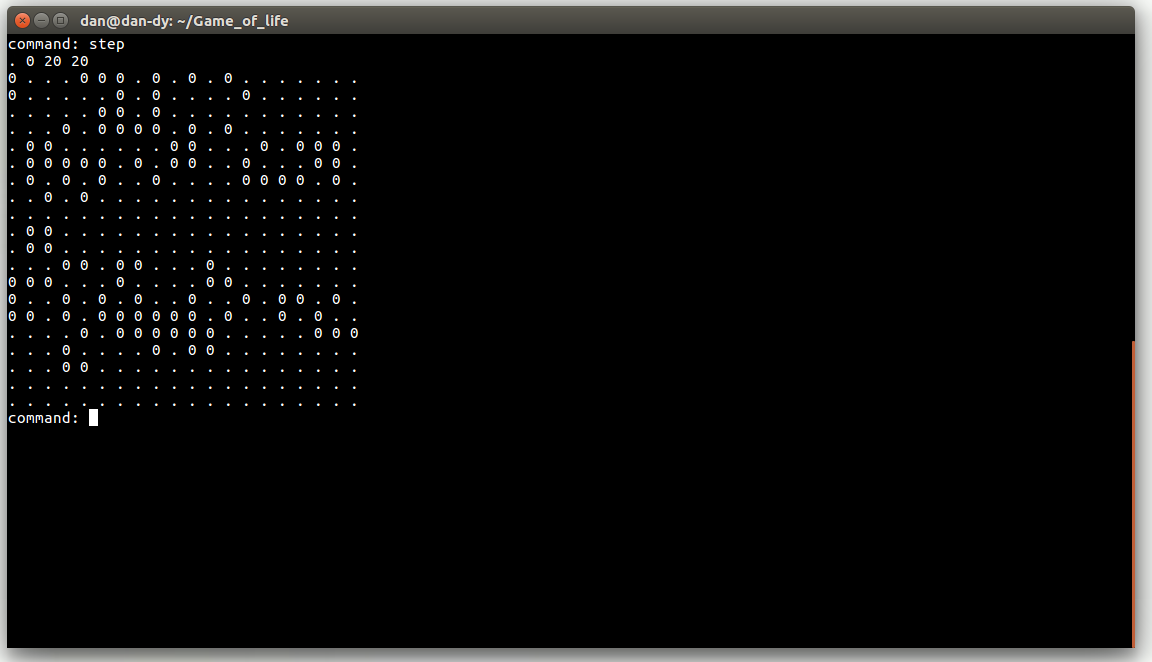
read



loop



step



# ЛИЧНЫЙ ВКЛАД В ПРОЕКТ

1. **Общее**

Наш проект — коллаборация двух друзей. Александр и я вели всю работу вместе: от создания фундамента до написания отчета. Создание структур для хранения матрицы и клеток и интерфейсы к ним - фундамент проекта. Далее была создана функция «menu» для работы с пользователем в консоли. Следующий пункт работы — тесты и отчет. Сделано.

1. **Личные наработки**.

Функции для токенайзера (принятие нескольких комманд, разбиение и обработка их). Советы по улучшению кода. Некоторые функции в matrix и cell.

1. **Описание структур**.

***Cell*** содержит

int state : хранит состояние клетки (1 — жив, 0 — мертв)

char empty\_char\_cell : хранит символ, который будет выводиться в консоли, если клетка мертва

char life\_char\_cell : хранит символ, который будет выводиться в консоли, если клетка жива

char char\_cell — хранит empty\_char\_cell или life\_char\_cell.

***Matrix*** содержит

char empty\_cell : хранит символ мертвой клетки

char life\_cell : хранит символ живой клетки

int size\_x : размер поля по x

int size\_y : размер поля по y

Cell \*cell : массив структур типа Cell.

1. **Описание функций**

***init\_matrix*** : инициализация матрицы

***get\_cell\_from\_matrix*** : возвращает указатель на клетку с координатами x и y из матрицы

***free\_matrix*** : очистка памяти выделенной под матрицу

***copy\_matrix*** : копирует матрицу в матрицу буффер

***read\_file*** : считывает данные настроек матрицы с файла

***write\_file*** : записывает данные настроек матрицы в файл

***resize\_matx*** : изменяет размеры матрицы

***print\_matrix*** : выводит матрицу в консоль

***rules\_matx*** : выполняет правила игры для матрицы

***init\_cell*** : инициализация массива клеток

***mode\_cell*** : изменение состояния клетки (жива, мертва)

***check\_cell*** : возвращает состояние клетки (1 — жива, 0 мертва)

***char\_cell*** : возвращает символ клетки (мертвой или живой)

# ПРИЛОЖЕНИЕ. ТЕКСТ ПРОГРАММЫ

1. Заголовочные файлы:

***matrix.h***

**#ifndef MATRIX\_H**

**#define MATRIX\_H**

**#include "cell.h"**

**typedef struct {**

**char empty\_cell;**

**char life\_cell;**

**int size\_x;**

**int size\_y;**

**Cell \*cell;**

**} Matrix;**

**Matrix \*init\_matrix(int size\_x, int size\_y, char life\_cell, char empty\_cell);**

**Cell \*get\_cell\_from\_matrix(Matrix \*matx, int x, int y);**

**void free\_matrix(Matrix \*matx);**

**void copy\_matrix(Matrix \*matx\_des, Matrix \*matx\_src);**

**int read\_file(char \*name\_file, Matrix \*matx);**

**int write\_file(char \*name\_file, Matrix \*matx);**

**Matrix \*resize\_matx(Matrix \*matx, int x, int y);**

**void print\_matrix(Matrix \*matx);**

**Matrix \*rules\_matx(Matrix \*matx);**

**#endif**

***cell.h***

**#ifndef CELL\_H**

**#define CELL\_H**

**typedef struct {**

**int state;**

**char empty\_char\_cell;**

**char life\_char\_cell;**

**char char\_cell;**

**} Cell;**

**Cell init\_cell(int mode, char life\_char, char empty\_char);**

**void mode\_cell(Cell \*cell, int mode);**

**int check\_cell(Cell cell);**

**char char\_cell(Cell cell);**

**#endif**

1. Исходные коды:

***matrix.c***

**#include "matrix.h"**

**#include "cell.h"**

**#include <stdlib.h>**

**#include <stdio.h>**

**#define LIFE 1**

**#define DEAD 0**

**Matrix \*init\_matrix(int size\_x, int size\_y, char life\_cell, char empty\_cell)**

**{**

**Matrix \*matx = calloc(1, sizeof(Matrix));**

**if (matx == NULL) {**

**return NULL;**

**}**

**matx->size\_x = size\_x;**

**matx->size\_y = size\_y;**

**matx->empty\_cell = empty\_cell;**

**matx->life\_cell = life\_cell;**

**matx->cell = calloc(size\_x \* size\_y, sizeof(Cell));**

**for (int i = 0; i < size\_x \* size\_y; i++) {**

**matx->cell[i] = init\_cell(0, life\_cell, empty\_cell);**

**}**

**return matx;**

**}**

**Cell \*get\_cell\_from\_matrix(Matrix \*matx, int x, int y)**

**{**

**return matx->cell + matx->size\_y \* y + x;**

**}**

**void free\_matrix(Matrix \*matx)**

**{**

**free(matx->cell);**

**free(matx);**

**}**

**void copy\_matrix(Matrix \*matx\_des, Matrix \*matx\_src)**

**{**

**for (int y = 0; y < matx\_src->size\_y; y++) {**

**for (int x = 0; x < matx\_src->size\_x; x++) {**

**matx\_des->cell[x + matx\_src->size\_y \* y] = \*get\_cell\_from\_matrix(matx\_src, x, y);**

**}**

**}**

**}**

**int read\_file(char \*name\_file, Matrix \*matx)**

**{**

**FILE \*in = fopen(name\_file, "r");**

**if (in == NULL) {**

**return 1;**

**}**

**int new\_x, new\_y;**

**fscanf(in, "%c %c %d %d", &matx->empty\_cell, &matx->life\_cell, &new\_x, &new\_y);**

**if (resize\_matx(matx, new\_x, new\_y) == NULL) {**

**return 1;**

**}**

**for (int y = 0; y < matx->size\_y; y++) {**

**for (int x = 0; x < matx->size\_x; x++) {**

**int tmp;**

**fscanf(in, "%d", &tmp);**

**if (tmp == '\n') {**

**x--;**

**}**

**matx->cell[x + matx->size\_y \* y] = init\_cell(tmp, matx->life\_cell, matx->empty\_cell);**

**}**

**}**

**fclose(in);**

**return 0;**

**}**

**int write\_file(char \*name\_file, Matrix \*matx)**

**{**

**FILE \*out = fopen(name\_file, "w");**

**if (out == NULL) {**

**return 1;**

**}**

**fprintf(out, "%c %c %d %d\n", matx->empty\_cell, matx->life\_cell, matx->size\_x, matx->size\_y);**

**for (int y = 0; y < matx->size\_y; y++) {**

**for (int x = 0; x < matx->size\_x; x++) {**

**fprintf(out, "%d ", check\_cell(\*get\_cell\_from\_matrix(matx, x, y)));**

**}**

**fprintf(out, "\n");**

**}**

**fclose(out);**

**return 0;**

**}**

**Matrix \*resize\_matx(Matrix \*matx, int x, int y)**

**{**

**matx->cell = realloc(matx->cell, x \* y \* sizeof(Cell));**

**if (!matx->cell) {**

**return NULL;**

**}**

**for (int i = matx->size\_y \* matx->size\_x; i < y \* x; i++) {**

**matx->cell[i] = init\_cell(0, matx->life\_cell, matx->empty\_cell);**

**}**

**matx->size\_x = x;**

**matx->size\_y = y;**

**return matx;**

**}**

**void print\_matrix(Matrix \*matx)**

**{**

**printf("%c %c %d %d\n", matx->empty\_cell, matx->life\_cell, matx->size\_x, matx->size\_y);**

**for (int y = 0; y < matx->size\_y; y++) {**

**for (int x = 0; x < matx->size\_x; x++) {**

**printf("%c ", char\_cell(\*get\_cell\_from\_matrix(matx, x, y)));**

**}**

**printf("\n");**

**}**

**}**

**Matrix \*rules\_matx(Matrix \*matx)**

**{**

**Matrix \*tmp\_matx = init\_matrix(matx->size\_x, matx->size\_y, matx->life\_cell, matx->empty\_cell);**

**if (!tmp\_matx) {**

**return NULL;**

**}**

**copy\_matrix(tmp\_matx, matx);**

**for (int y = 0; y < matx->size\_y; y++) {**

**for (int x = 0; x < matx->size\_x; x++) {**

**int count = 0;**

**int local = x + matx->size\_y \* y;**

**for (int local\_y = y - 1; local\_y <= y + 1; local\_y++) {**

**if (local\_y < 0) {**

**continue;**

**}**

**if (local\_y >= matx->size\_y) {**

**continue;**

**}**

**for (int local\_x = x - 1; local\_x <= x + 1; local\_x++) {**

**if (local\_x < 0) {**

**continue;**

**}**

**if (local\_x >= matx->size\_x) {**

**continue;**

**}**

**if (local\_x == x && local\_y == y) {**

**continue;**

**}**

**if (check\_cell(\*get\_cell\_from\_matrix(matx, local\_x, local\_y)) == 1) {**

**count++;**

**}**

**}**

**}**

**if (check\_cell(matx->cell[local]) == 0 && count == 3) {**

**mode\_cell(tmp\_matx->cell + local, LIFE);**

**}**

**if (check\_cell(matx->cell[local]) == 1 && (count == 3 || count == 2)) {**

**continue;**

**}**

**if (count < 2 || count > 3) {**

**mode\_cell(tmp\_matx->cell + local, DEAD);**

**}**

**}**

**}**

**copy\_matrix(matx, tmp\_matx);**

**return matx;**

**}**

***cell.c***

**#include "cell.h"**

**#include <stdlib.h>**

**Cell init\_cell(int mode, char life\_char, char empty\_char)**

**{**

**Cell cell;**

**cell.life\_char\_cell = life\_char;**

**cell.empty\_char\_cell = empty\_char;**

**mode\_cell(&cell, mode);**

**return cell;**

**}**

**void mode\_cell(Cell \*cell, int mode)**

**{**

**cell->state = mode;**

**if (mode) {**

**cell->char\_cell = cell->life\_char\_cell;**

**} else {**

**cell->char\_cell = cell->empty\_char\_cell;**

**}**

**}**

**int check\_cell(Cell cell)**

**{**

**return cell.state;**

**}**

**char char\_cell(Cell cell)**

**{**

**return cell.char\_cell;**

**}**

***main.c***

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <time.h>**

**#include "matrix.h"**

**#include "cell.h"**

**#include <string.h>**

**#include <ctype.h>**

**#ifdef WIN32**

**#include <conio.h>**

**#include <windows.h>**

**#elif \_\_linux\_\_**

**#include <unistd.h>**

**#define CSI "\x1B\x5B"**

**char colors[][5] = {**

**"0;30", /\* Black \*/ "1;30", /\* Dark Gray \*/**

**"0;31", /\* Red \*/ "1;31", /\* Bold Red \*/**

**"0;32", /\* Green \*/ "1;32", /\* Bold Green \*/**

**"0;33", /\* Yellow \*/ "1;33", /\* Bold Yellow \*/**

**"0;34", /\* Blue \*/ "1;34", /\* Bold Blue \*/**

**"0;35", /\* Purple \*/ "1;35", /\* Bold Purple \*/**

**"0;36", /\* Cyan \*/ "1;36" /\*Bold Cyan \*/ };**

**int colors\_sz = sizeof(colors) / sizeof(colors[0]);**

**#endif**

**void print\_help()**

**{**

**#ifdef \_\_linux\_\_**

**printf("%s%sm", CSI, colors[9]);**

**#endif**

**printf("help\t: displays help.\n");**

**printf("next\t: repeats the previous command.\n");**

**printf("write [file\_name]\t: writes the matrix to a file (you must specify a file name)\n");**

**printf("read [file\_name]\t: reads the file containing the matrix and its settings (you must specify a file name)\n");**

**printf("loop [repeat\_num] [delay]\t: repeats \"calc\" and \"print\", \"repeat\_num\" times and with a \"delay\" in sec, delay is maybe float, repeat\_num can be zero then the loop will be infinite \n");**

**printf("calc\t: makes one game step.\n");**

**printf("print\t: print matrix in console.\n");**

**printf("step\t: makes one game step and print matrix in console.\n");**

**printf("clear\t: clear console.\n");**

**printf("exit\t: completes the game.\n");**

**#ifdef \_\_linux\_\_**

**printf("%s0m", CSI);**

**#endif**

**}**

**void print\_get\_name\_file(char \*name\_file)**

**{**

**printf("Enter to name of file: ");**

**scanf("%s", name\_file);**

**}**

**int isdigit\_str(char \*str)**

**{**

**while (\*str != 0 && \*str != '\n') {**

**if (!isdigit(\*str)) {**

**return 0;**

**}**

**str++;**

**}**

**return 1;**

**}**

**int isfloat\_str(char \*str)**

**{**

**int check = 0;**

**while (\*str != 0 && \*str != '\n') {**

**if (!isdigit(\*str)) {**

**if (\*str == '.') {**

**check++;**

**} else {**

**return 0;**

**}**

**}**

**str++;**

**}**

**if (check == 1) {**

**return 1;**

**} else {**

**return 0;**

**}**

**}**

**char \*\*tokenize(char \*str, int \*token\_num)**

**{**

**char \*\*tokens = calloc(10, sizeof(char\*));**

**int tok\_p = 0;**

**tokens[tok\_p] = strtok(str, " ");**

**tok\_p++;**

**while (tokens[tok\_p - 1] != NULL) {**

**//printf("%s\n", tokens[tok\_p - 1]);**

**tokens[tok\_p] = strtok (NULL, " ");**

**tok\_p++;**

**}**

**\*token\_num = tok\_p - 1;**

**return tokens;**

**}**

**void menu(Matrix \*matx)**

**{**

**char answer[200];**

**char tmp\_answer[200];**

**while (1) {**

**printf("command: ");**

**//scanf("%s", answer);**

**fgets(answer, 200, stdin);**

**if (answer[strlen(answer) - 1] == '\n') {**

**answer[strlen(answer) - 1] = 0;**

**}**

**if (!strcmp(answer, "next") || answer[0] == '\0') {**

**strcpy(answer, tmp\_answer);**

**}**

**strcpy(tmp\_answer, answer);**

**int token\_num;**

**char \*\*tokens = tokenize(answer, &token\_num);**

**for (int i = 0; i < token\_num; i++) {**

**if (!strcmp(tokens[i], "help")) {**

**print\_help();**

**continue;**

**}**

**if (!strcmp(tokens[i], "loop")) {**

**float delay\_time = 0.5;**

**#ifdef WIN32**

**int iter\_num = 0;**

**#elif \_\_linux\_\_**

**int iter\_num = 5;**

**#endif**

**if (token\_num - i >= 2 && isdigit\_str(tokens[i + 1])) {**

**iter\_num = atof(tokens[i + 1]);**

**}**

**if (token\_num - i >= 3 && (isdigit\_str(tokens[i + 2]) || isfloat\_str(tokens[i + 2]))) {**

**delay\_time = atof(tokens[i + 2]);**

**}**

**for (int i = 0; (iter\_num == 0 || i < iter\_num); i++) {**

**rules\_matx(matx);**

**print\_matrix(matx);**

**#ifdef WIN32**

**Sleep(delay\_time \* 1000);**

**if (kbhit()) {**

**i = iter\_num + 1;**

**iter\_num = -1;**

**break;**

**}**

**#elif \_\_linux\_\_**

**usleep(delay\_time \* 1000000);**

**#endif**

**}**

**i = i + 2;**

**continue;**

**}**

**if (!strcmp(tokens[i], "calc")) {**

**rules\_matx(matx);**

**continue;**

**}**

**if (!strcmp(tokens[i], "print")) {**

**print\_matrix(matx);**

**continue;**

**}**

**if (!strcmp(tokens[i], "step")) {**

**rules\_matx(matx);**

**print\_matrix(matx);**

**continue;**

**}**

**if (!strcmp(tokens[i], "clear")) {**

**system("clear");**

**continue;**

**}**

**if (!strcmp(tokens[i], "write")) {**

**char name\_file[100];**

**if (token\_num - i == 1) {**

**printf("Enter to name of file: ");**

**fgets(name\_file, 100, stdin);**

**} else {**

**strcpy(name\_file, tokens[i + 1]);**

**i++;**

**}**

**if (name\_file[strlen(name\_file) - 1] == '\n') {**

**name\_file[strlen(name\_file) - 1] = 0;**

**}**

**write\_file(name\_file, matx);**

**printf("Complete.\n");**

**continue;**

**}**

**if (!strcmp(tokens[i], "read")) {**

**char name\_file[100];**

**if (token\_num - i == 1) {**

**printf("Enter to name of file: ");**

**fgets(name\_file, 100, stdin);**

**} else {**

**strcpy(name\_file, tokens[i + 1]);**

**i++;**

**}**

**if (name\_file[strlen(name\_file) - 1] == '\n') {**

**name\_file[strlen(name\_file) - 1] = 0;**

**}**

**read\_file(name\_file, matx);**

**printf("Complete.\n");**

**continue;**

**}**

**if (!strcmp(tokens[i], "exit") || !strcmp(tokens[i], "quit")) {**

**return;**

**}**

**print\_help();**

**}**

**}**

**}**

**int main()**

**{**

**char empty\_cell = '.';**

**char life\_cell = '0';**

**int size\_x = 10;**

**int size\_y = 10;**

**Matrix \*matx = init\_matrix(size\_x, size\_y, life\_cell, empty\_cell);**

**if (matx == NULL) {**

**printf("Error init matrix\n");**

**return 1;**

**}**

**menu(matx);**

**free\_matrix(matx);**

**return 0;**

**}**