МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ
«Санкт-Петербургский национальный исследовательский
университет информационных технологий, механики и оптики»
Факультет информационных технологий и программирования
Кафедра информационных систем

Программирование
Лабораторная работа № 5

Выполнил студент:
Орлов Александр

Группа: M3107

САНКТ-ПЕТЕРБУРГ
2021

Целью лабораторной работы является реализация игры “Жизнь”, позволяющая
выводить поколение игры в монохромную картинку в формате BMP. Плоскость
“вселенной” игры ограничена положительными координатами.
Лабораторная работы должна быть выполнена в виде консольного приложения
принимающего в качестве аргументов следующие параметры:

1. --input input_file.bmp
Где input_file.bmp - монохромная картинка в формате bmp, хранящая начальную ситуация (первое поколение) игры
2. --output dir_name
Название директории для хранения поколений игры в виде монохромной картинки
3. --max_iter N
Максимальное число поколений, которое может эмулировать программа. Необязательный параметр, по-умолчанию бесконечность
4. --dump_freq N
Частота с которой программа должно сохранять поколения виде картинки. Необязательный параметр, по-умолчанию равен 1

main.c:

```
#include <stdio.h>
#include "bmp.h"
#include "game.h"
#include "glut.h"

unsigned char *Frame;
double Zoom = 1;
int isPause = 1, W, H;

void PutPixel(int X, int Y, int G, int B, int R)
{
    if (X < 0 || Y < 0 || X >= W || Y >= H)
        return;
    Frame[(Y * W + X) * 3 + 0] = B;
    Frame[(Y * W + X) * 3 + 1] = G;
    Frame[(Y * W + X) * 3 + 2] = R;
}

int GetCell(byte* F, int x, int y)
{
    x = (x + W) % W;
    y = (y + H) % H;
    return F[y * W + x];
}

void SetCell(byte* F, int x, int y, int v)
{
```

```
    x = (x + W) % W;
    y = (y + H) % H;
    F[y * W + x] = v;
}

void FieldDraw(byte* F)
{
    int x, y;

    for (y = 0; y < H; y++)
        for (x = 0; x < W; x++)
            if (GetCell(F, x, y))
                PutPixel(x, y, 255, 50, 100);
            else
                PutPixel(x, y, 60, 30, 100);
}

void FieldClear(byte* F)
{
    int x, y;

    for (y = 0; y < H; y++)
        for (x = 0; x < W; x++)
            SetCell(F, x, y, 0);
}

int GetNeighbours(byte* F, int x, int y)
{
    return GetCell(F, x + 1, y)
        + GetCell(F, x + 1, y + 1)
        + GetCell(F, x + 1, y - 1)
        + GetCell(F, x, y + 1)
        + GetCell(F, x, y - 1)
        + GetCell(F, x - 1, y - 1)
        + GetCell(F, x - 1, y)
        + GetCell(F, x - 1, y + 1);
}

void NewGeneration(byte* F1, byte* F2)
{
    int x, y, v, n;

    for (y = 0; y < H; y++)
        for (x = 0; x < W; x++)
        {
            n = GetNeighbours(F1, x, y);
            if (GetCell(F1, x, y))
                if (n < 2 || n > 3)
                    v = 0;
                else
                    v = 1;
            else
                if (n == 3)
                    v = 1;
                else
                    v = 0;
            SetCell(F2, x, y, v);
        }
}

void Keyboard(unsigned char key, int X, int Y)
{
    if (key == 27)
        exit(0);
    else
```

```
        PutPixel(X, Y, 0, 0, 500);
}

void Display(void)
{
    byte* tmp;

    glClearColor(0.5, 0.5, 0.5, 1);
    glClear(GL_COLOR_BUFFER_BIT);

    if (isPause)
    {
        FieldDraw(F1);
        NewGeneration(F1, F2);
        tmp = F1;
        F1 = F2;
        F2 = tmp;
    }

    glRasterPos2d(-1, 1);
    glPixelZoom(Zoom, -Zoom);
    glDrawPixels(W, H, GL_BGR_EXT, GL_UNSIGNED_BYTE, Frame);

    glFinish();
    glutPostRedisplay();
    glutSwapBuffers();
}

int main(int argc, char* argv[]) {
    char filename[MAX_STRING_SIZE], dirname[MAX_STRING_SIZE];
    char* offset;
    int iterations = 3, frequency = 1, opengl = 1;
    int **pixels, **pixels2;
    FILE *f;
    bitMapFile bmp;

    if (argc == 2 && !strcmp(argv[1], "--help")) {
        printf("This app supports two modes:\n1) Just generating several generations\n2)
OpenGl mode with saving to bmp\n");
        printf("If you want to activate OpenGl mode your .gmp picture should have NxN
size. Also n %% 100 == 0\n");
        printf("App supports 5 commands last three of them are optional:\n--input -
source image filename\n");
        printf("--output - output directory path\n--max_iter - max number of
iterations\n--dump_freq - frequency of creating generations\n");
        printf("--opengl - OpenGl status (0 to turn off, 1 to turn on)\n");
        printf("Example of command: Lab5.exe --input vitsan.bmp --output D:/ --max_iter 3
--dump_freq 1 --opengl 1\n");
        return 0;
    }

    if (argc < 5) {
        printf("Error: invalid parameters\n");
        return 1;
    }

    for (int i = 1; i < argc; i += 2) {
        if (!strcmp("--input", argv[i]))
            strcpy(filename, argv[i + 1]);
        else if (!strcmp("--output", argv[i]))
            strcpy(dirname, argv[i + 1]);
        else if (!strcmp("--max_iter", argv[i])) {
            iterations = atoi(argv[i + 1]);
            if (iterations < 0) {
                printf("Error: incorrect iterations\n");
```

```
            return 1;
        }
    }
    else if (!strcmp("--dump_freq", argv[i])) {
        frequency = atoi(argv[i + 1]);
        if (frequency < 0) {
            printf("Error: incorrect frequency\n");
            return 1;
        }
    }
    else if (!strcmp("--opengl", argv[i])) {
        opengl = atoi(argv[i + 1]);
        if (opengl != 0 && opengl != 1) {
            printf("Error: incorrect opengl status\n");
            return 1;
        }
    }
}

/* Trying to open the file */
f = fopen(filename, "rb");
if (f == NULL) {
    printf("Error: can't open the file\n");
    return 1;
}

/* Reading first 54 bytes of some information */
fread(&bmp, 1, sizeof(bitMapFile), f);

offset = (char *) calloc(sizeof(char), bmp.bfOffs);
fseek(f, 0, SEEK_SET);
fread(offset, 1, bmp.bfOffs, f); //after that pixels go

/* Initializing variables for OpenGl */
W = bmp.biWidth;
H = bmp.biHeight;

pixels = bmpToPixels(f, bmp);
pixels2 = pixels;
Frame = (unsigned char *)malloc(W * H * 3 * sizeof(unsigned char));

if (opengl) {
    if (W % 100 != 0 || H % 100 != 0 || W != H) {
        printf("Error: incorrect parameters for OpenGl mode\n");
        return 1;
    }

    for (int i = 0; i < iterations; i++)
        if (i % frequency == 0) {
            pixels = newGeneration(pixels, bmp);
            arrToBmp(pixels, bmp, i, dirname, offset);
        }


    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);

    F1 = malloc(W * H);
    F2 = malloc(W * H);
    fieldInit(F1, pixels2, bmp);
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(H, W);
    glutCreateWindow("Vitsan onelove");
    glutDisplayFunc(Display);
```

```
        glutKeyboardFunc(Keyboard);

        glutMainLoop();
    }
    else
        for (int i = 0; i < iterations; i++)
            if (i % frequency == 0) {
                pixels = newGeneration(pixels, bmp);
                arrToBmp(pixels, bmp, i, dirname, offset);
            }
    return 0;
}
```

bmp.c:

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "bmp.h"


int **bmpToPixels(FILE* f, bitMapFile bmp) {
    int width = bmp.biWidth, height = bmp.biHeight, lineLength;
    char* line;
    int** pixels;
    int byteWidth = (width / 8) + ((width % 8 > 0) ? 1 : 0); //should be integer number
of bytes

    /* Creating gaming pole */
    pixels = (int **)calloc(width, sizeof(int *));
    for (int i = 0; i < width; i++)
        pixels[i] = (int *)calloc(height, sizeof(int));

    fseek(f, bmp.bfOffs, SEEK_SET);

    if (byteWidth % 4 != 0)
        lineLength = byteWidth + (4 - (byteWidth % 4));
    else
        lineLength = byteWidth;

    line = (char *)calloc(lineLength, sizeof(char));

    for (int j = height - 1; j >= 0; j--) {
        fread(line, 1, lineLength, f);

        for (int i = 0; i < width; i++) {
            int currentByte = i / 8, currentBit = 1 << (7 - i % 8);

            if (line[currentByte] & currentBit)
                pixels[i][j] = 0;
            else
                pixels[i][j] = 1;
        }
    }
    free(line);
    return pixels;
}

int arrToBmp(int **pixels, bitMapFile bmp, int generation, char *dirname, char *offset) {
    char* filePath = (char *)calloc(MAX_STRING_SIZE, sizeof(char));
    char* gen = (char*)calloc(30, sizeof(char));
    FILE* f;
    int width = bmp.biWidth, height = bmp.biHeight, lineLength;

    _itoa(generation, gen, 30);
```

```
    strcat(filePath, dirname);
    strcat(filePath, "/");
    strcat(filePath, gen);
    strcat(filePath, ".bmp");

    f = fopen(filePath, "wb");

    if (f == NULL)
        return 0;

    fwrite(offset, 1, bmp.bfOffs, f);

    int byteWidth = (width / 8) + ((width % 8 > 0) ? 1 : 0);

    if (byteWidth % 4 != 0)
        lineLength = byteWidth + (4 - (byteWidth % 4));
    else
        lineLength = byteWidth;

    for (int j = height - 1; j >= 0; j--) {
        char* line = (char*)calloc(lineLength, sizeof(char));

        for (int i = 0; i < width; i++) {
            int currentByte = i / 8;

            if (pixels[i][j] == 0)
                line[currentByte] = (char)(1 << (7 - i % 8)) | line[currentByte];
        }

        fwrite(line, 1, lineLength, f);
        free(line);
    }
    fclose(f);
    free(filePath);
    free(gen);
    return 1;
}
```

game.c:

```
#include "game.h"
#include "bmp.h"
#include "glut.h"

int getCell(int **pixels, int x, int y, bitMapFile bmp) {
    if (x < 0 || x >= bmp.biWidth || y < 0 || y >= bmp.biHeight)
        return 0;
    return pixels[x][y];
}

int getNeighbours(int **pixels, int x, int y, bitMapFile bmp) {
    return getCell(pixels, x + 1, y, bmp)
        + getCell(pixels, x + 1, y + 1, bmp)
        + getCell(pixels, x + 1, y - 1, bmp)
        + getCell(pixels, x, y + 1, bmp)
        + getCell(pixels, x, y - 1, bmp)
        + getCell(pixels, x - 1, y - 1, bmp)
        + getCell(pixels, x - 1, y, bmp)
        + getCell(pixels, x - 1, y + 1, bmp);
}

int **newGeneration(int **pixels, bitMapFile bmp) {
    int x, y, v, n;
    int** pixels2;
```

```
    pixels2 = (int**)calloc(bmp.biWidth, sizeof(int*));
    for (int i = 0; i < bmp.biWidth; i++)
        pixels2[i] = (int*)calloc(bmp.biHeight, sizeof(int));

    for (y = 0; y < bmp.biHeight; y++)
        for (x = 0; x < bmp.biWidth; x++) {
            n = getNeighbours(pixels, x, y, bmp);
            if (pixels[x][y])
                if (n < 2 || n > 3)
                    v = 0;
                else
                    v = 1;
            else
                if (n == 3)
                    v = 1;
                else
                    v = 0;
            pixels2[x][y] = v;
        }
    return pixels2;
}

void setCell(byte* F, int x, int y, int v, bitMapFile bmp) {
    x = (x + bmp.biHeight) % bmp.biHeight;
    y = (y + bmp.biHeight) % bmp.biHeight;
    F[y * bmp.biHeight + x] = v;
}

void fieldInit(byte* F, int **pixels, bitMapFile bmp) {
    int x, y;

    for (y = 0; y < bmp.biHeight; y++)
        for (x = 0; x < bmp.biWidth; x++)
            setCell(F, x, y, pixels[x][y], bmp);
}
```

bmp.h:

```
#include <stdio.h>
#include <inttypes.h>
#include <stdlib.h>
#include <iso646.h>
#include <string.h>
#include "game.h"

#define MAX_STRING_SIZE 512

#pragma pack(push, 1)
typedef struct stBitMapFile {
    /* Header */
    uint16_t bfType;
    uint32_t bfSize;
    uint16_t bfRes1;
    uint16_t bfRes2;
    uint32_t bfOffs;

    /* InfoHeader */
    uint32_t biSize;
    int32_t biWidth;
    int32_t biHeight;
    uint16_t biPlanes;
    uint16_t biBitCnt;
    uint32_t biCompr;
    uint32_t biSizeIm;
    int32_t biXPels;
```

```
    int32_t biYPels;
    uint32_t biClrUsed;
    uint32_t biClrImp;
} bitMapFile;
#pragma pack(pop)

int **bmpToPixels(FILE* f, bitMapFile bmp);
int arrToBmp(int** pixels, bitMapFile bmp, int generation, char* dirname, char* offset);
int** newGeneration(int** pixels, bitMapFile bmp);
void fieldInit(byte* F, int** pixels, bitMapFile bmp);
```

game.h:

```
typedef unsigned char byte;
byte* F1, * F2;
```

установить необходимые библиотеки для запуска режима OpenGl можно по ссылкам:
https://www.opengl.org//resources/libraries/glut/glut_downloads.php
https://sourceforge.net/projects/freeglut/

Вывод:
В результате выполнения лабораторной работы удалось реализовать программу, которая позволяет запускать игру Жизнь на произвольных монохромных .bmp картинках, а также сохранять поколения игры с заданными параметрами. Кроме того, удалось реализовать графический режим работы игры с помощью OpenGL. Запуск режима осуществляется с помощью соответствующего параметра при запуске программы из консоли.