Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Кафедра прикладной математики и программирования

Разработка игры «Тетрис»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ
по дисциплине «Языки программирования»
ЮУрГУ–010304.2023.113.ПЗ КР

Автор работы,
студент группы ЕТ-114
_____ Струков Л.О.
«____» _____ 2023 г.

Руководитель работы,
старший преподаватель
_____ Шелудько А.С.
«____» _____ 2023 г.

Работа защищена с оценкой
_____
«____» _____ 2023 г.

Челябинск 2023

АННОТАЦИЯ

Струков Л.О. Разработка игры «Тетрис». –Челябинск: ЮУрГУ, ЕТ-114, 2023. – 45 с., 19 ил., библиогр. список – 5 наим., 3 прил.

Целью работы является разработка компьютерной игры «Тетрис». В разделе 1 приведены требования к интерфейсу и функционалу программы, а также схемы программного меню, игрового поля и вспомогательных окон. В разделе 2 рассмотрена формализация задачи и описаны структуры данных, которые используются в программной реализации. В разделе 3 представлены схемы алгоритмов, в том числе алгоритм работы программного меню, алгоритм генерации блоков. В разделе 4 приведено описание программных модулей, структур данных, функций, констант и глобальных переменных. Текст программы, руководство пользователя и примеры выполнения программы приведены в приложениях.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Целью работы является разработка компьютерной игры «Тетрис». Для разработки используется язык программирования C++, графическая библиотека WinBGIm и среда разработки MinIDE. Для достижения поставленной цели необходимо выполнить следующие этапы разработки:

– определить требования к интерфейсу и функционалу программы;

– провести формализацию задачи, определить как состояние игры будет представляться в цифровом виде;

– определить структуры данных, которые будут использоваться в программной реализации;

– разработать алгоритм генерации фигур тетриса, алгоритм вращения фигур тетриса, механизм перемещения фигур тетриса;

– написать, отладить и протестировать программу, которая реализует игровой процесс;

– реализовать программное меню.

1 ПОСТАНОВКА ЗАДАЧИ

Рассмотрим основные требования к интерфейсу программы, а также функциональные возможности, которые должны быть реализованы при разработке компьютерной игры «Тетрис».

После запуска программы появляется заставка игры. После нажатия любой клавиши на клавиатуре открывается главное меню программы, которое содержит пункты «Начать игру», «Настройки», «О программе», «Статистика» и «Выход» (рисунок 1.1). Выбор пункта меню осуществляется нажатием левой кнопки мыши.

При выборе пункта «Настройки» открывается окно с настройками, в котором пользователь может задать скорость падения блоков и сменить имя игрока. Для возврата в главное меню необходимо нажать клавишу ESC. Схема окна с настройками показана на рисунке 1.2.

При выборе пункта «Статистика» открывается окно с информацией о игроках и их счетом. Схема окна с рекордами показана на рисунке 1.3.

При выборе пункта «О программе» появляется окно с информацией о правилах игры, а также сведения о разработчике программы. При выборе пункта «Выход» программа завершает работу.

При выборе пользователем пункта «Начать игру» появляется игровое окно (рисунок 1.4). В левой части окна расположено игровое поле тетриса, в котором отмечено текущее положение блоков.

Для перемещения и вращения фигурок тетриса используются стрелки на клавиатуре.

Справа от игрового поля расположены текстовые поля с указанием количества собранных линий, очков, следующией фигуры тетриса, скорость падения блоков.

Случайные генерируемые фигурки тетрамино падают сверху в прямоугольныое игровое поле шириной 10 и высотой 20 клеток. В полете игрок может поворачивать фигурку на 90° и двигать её по горизонтали. Также можно «сбрасывать» фигурку, то есть ускорять ее падение, когда уже решено, куда фигурка должна упасть. Фигурка летит до тех пор, пока не наткнётся на другую фигурку либо не достигнет нижней части игрово-

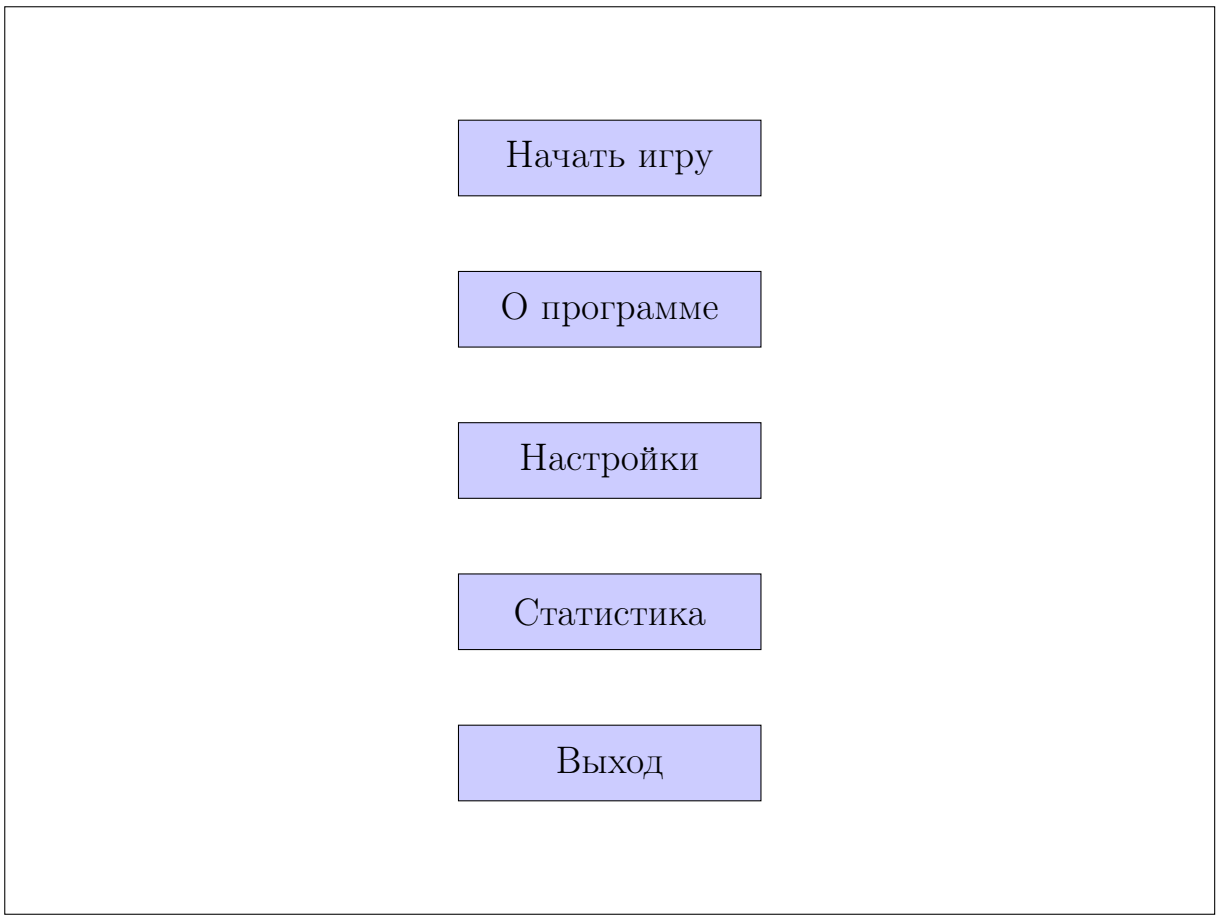

Рисунок 1.1 – Схема главного меню

го поля. Если при этом заполнился горизонтальный ряд из 10 клеток, он пропадает и все, что выше него, опускается на одну клетку. Дополнительно показывается фигурка, которая будет следовать после текущей. Темп игры постепенно ускоряется. Игра заканчивается, когда новая фигурка не может поместиться в игровое поле. Игрок получает очки за каждый заполненный ряд.

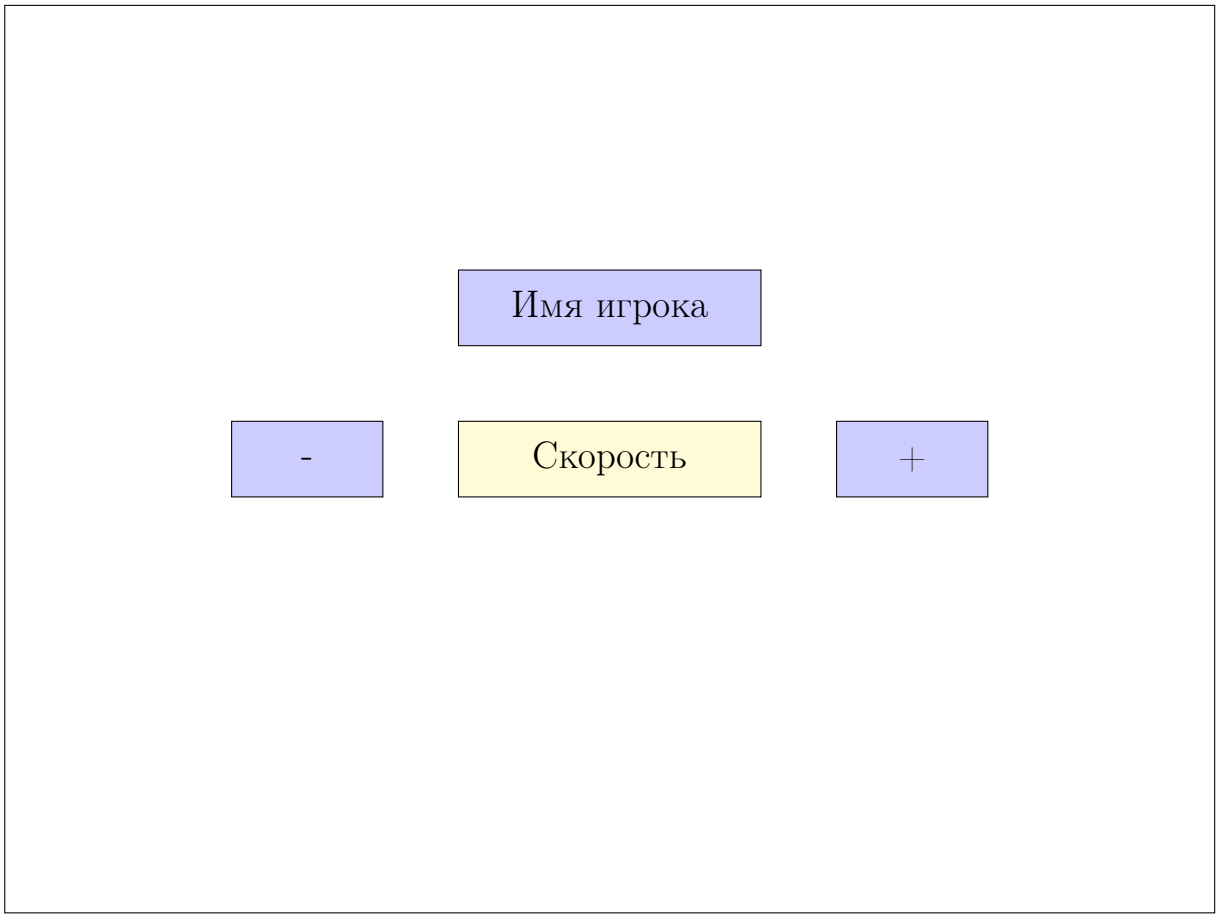

Рисунок 1.2 – Схема окна с настройками

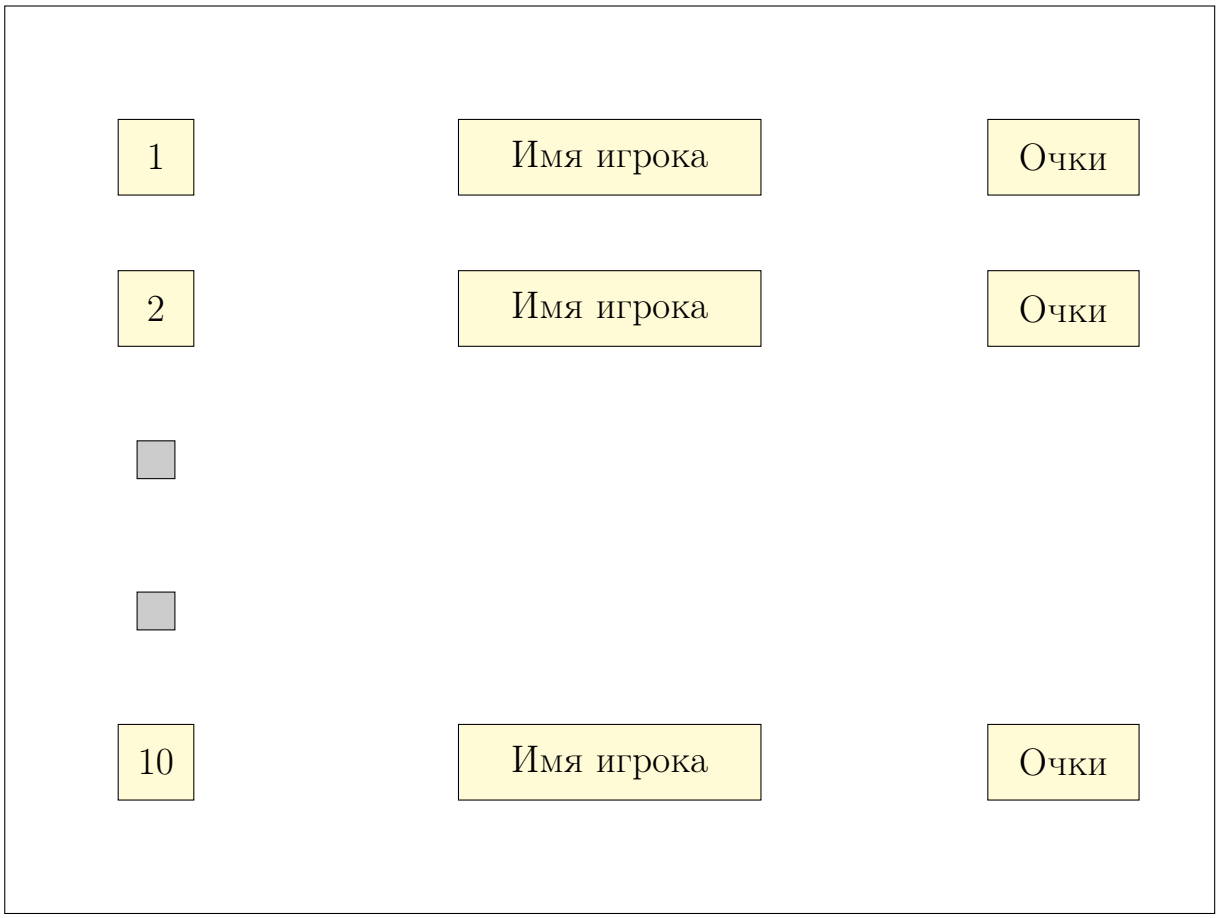

Рисунок 1.3 – Схема окна с Рекордами

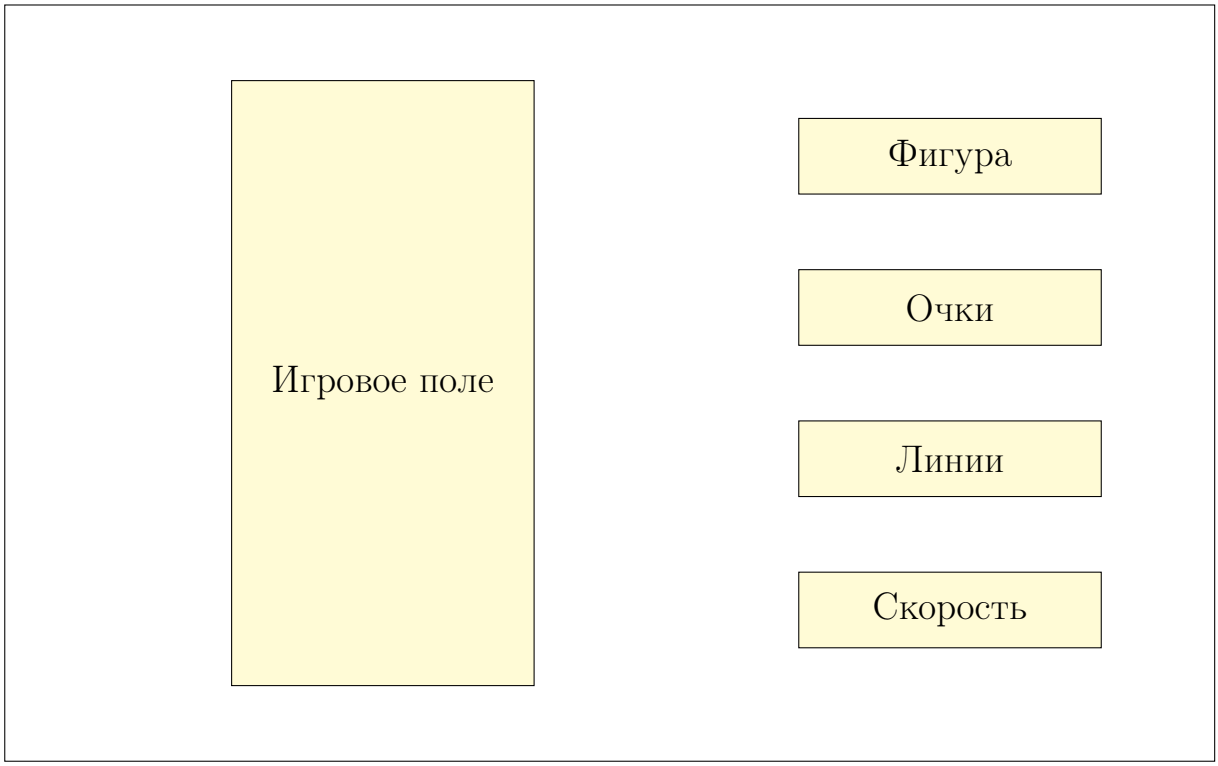

Рисунок 1.4 – Схема игрового окна

2 ФОРМАЛИЗАЦИЯ ЗАДАЧИ

Для отображения и хранения информации о игровом поле используется матрица `board` 10 на 20 ячеек.

В матрице board каждая ячейка содержит информацию о блоке на поле. Если ячейка равна 0 то значит блок пустой и закрашен черным цветом. Если ячека равна от 1 до 7, то значит блок заполнен и закрашен определенным цветом, который определяется индексом массива цветов от 0 до 7, где 0 – черный; 1 – синий; 2 – зеленый; 3 – голубой; 4 – красный; 5 – пурпурный; 6 – коричневый; 7 – желтый;

Для отображения фигуры используется матрица `tetromino` 4 на 4 ячейки.

Матрица `tetromino` задается с помощью изначально определеного массива матриц фигур тетриса. где индексы от 0 до 6 этого массива это матрица фигуры тетриса, где 0-ой индекс – I-образная фигруа; 1 – O-образная фигура; 2 – T-образная фиугра; 3 – S-образная фигура; 4 – Z-образная фигура; 5 – L-образная фигура; 6 – J-образная фигура;

Матрица tetromino хранит информацию о фигуре тетриса. Например матрицы:

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 2 & 0 \\ 0 & 2 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

будут хранить информацию о I-образной и O-образной фигурах тетриса соответствено.

Чтобы отобразить фигуру на игровом поле, матрица tetromino соединяеться с матрицей `board` с помощью координат `[x + j, y + i]`, где `x, y` - координаты фигуры на игровом поле, где `x` отвечает на каком блоке фигура начинается по горизонтале, а `y` отвечает на каком блоке фигура начинается по вертикале. `i, j` - это индексы ячеек матрицы `tetromino` не равные нулю.

Реализация падения фигуры представляет собой увеличение коорди-

наты y на 1 с проверкой может ли переместиться фигура вниз по игровому полю и не выходит ли за границы поля. Чтобы проверить может ли фигура переместиться вниз по игровому полю, надо посмотреть индексы `i, j`матрицы `tetromino` не равные нулю и если индексы `[x + j, y + i + 1]` в матрице board не равны нулю, то значит переместить фигуру вниз нельзя, иначе можно.

Реализация вращения фигуры представляет собой транспонирование матрицы tetromino и проверка может ли транспонированная матрица `tetromino` поместиться на игровое поле, то есть проверка индексов i, j транспонированной матрицы `tetromino` не равных нулю и если индексы `[x + j, y + i]` в матрице `board` не равны нулю, то вращение нельзя выполнить, иначе можно.

Реализация управления фигурой тетриса представляет собой увеличение или уменьшение координа `x` или `y` на 1 и проверкой выхода за границу поля.

3 РАЗРАБОТКА АЛГОРИТМА

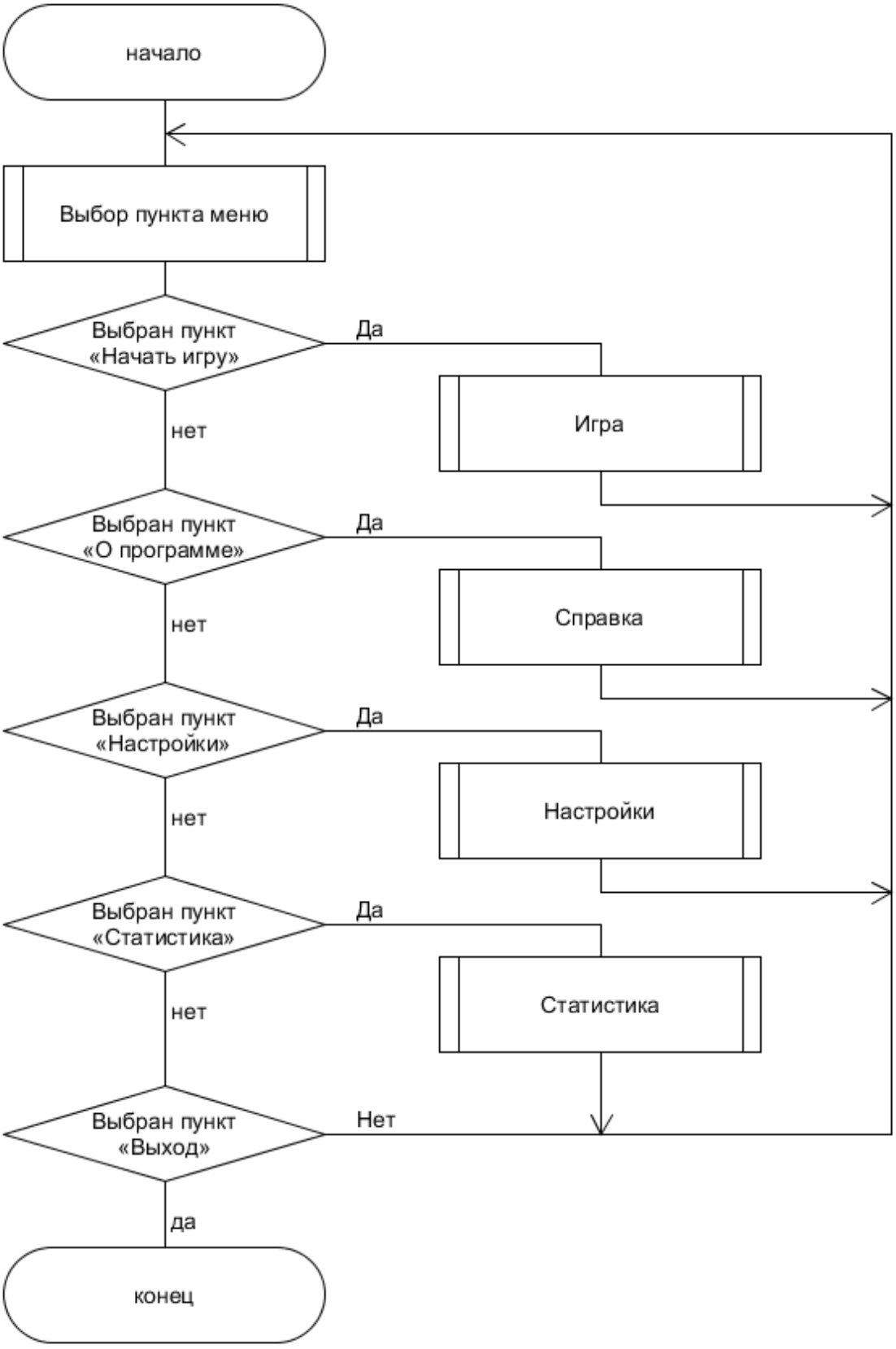

Рисунок 3.1 – Алгоритм работы программного меню

Рисунок 3.2 – Алгоритм генерации фигур тетриса

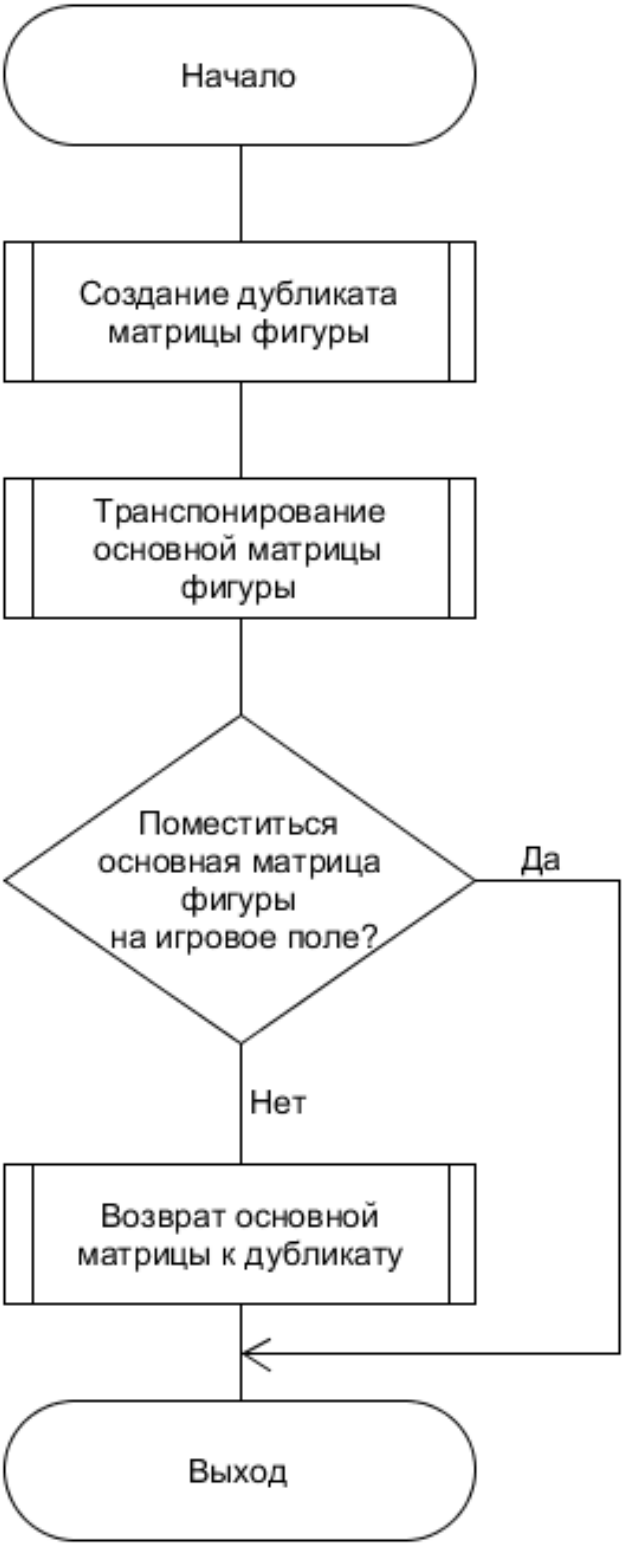

Рисунок 3.3 – Алгоритм вращения фигур тетриса

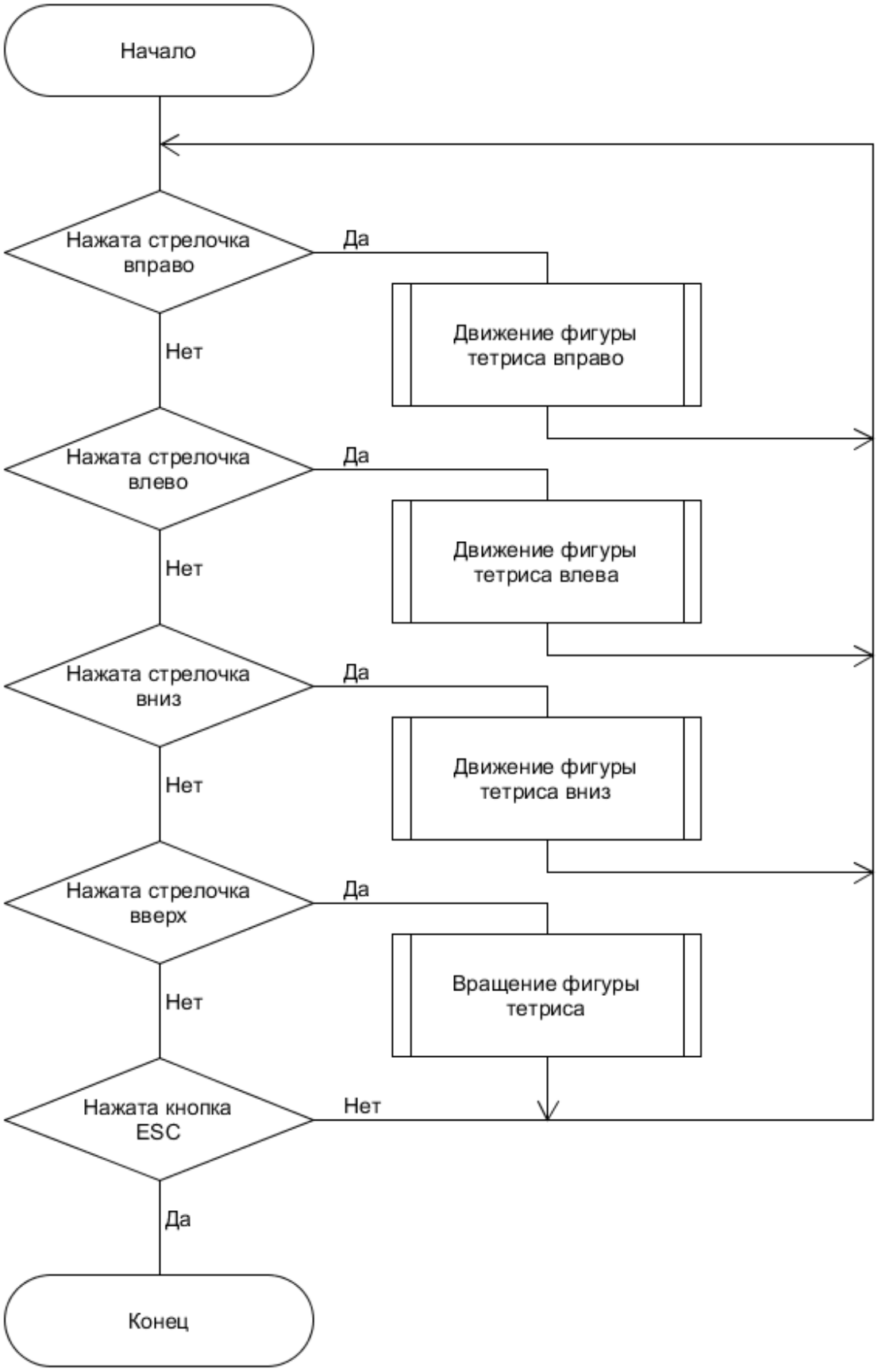

Рисунок 3.4 – Алгоритм работы управления

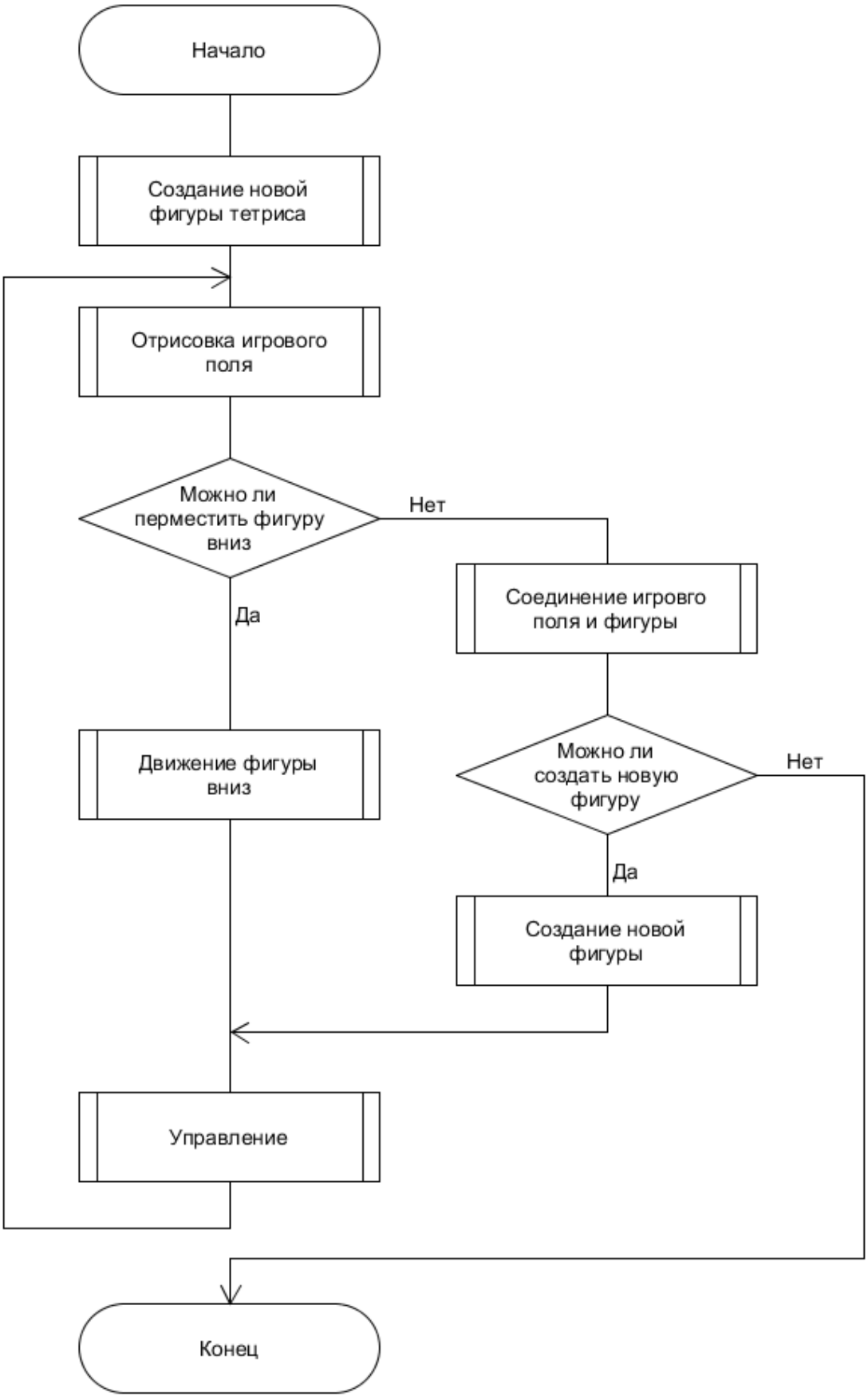

Рисунок 3.5 – Алгоритм работы игры

4 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Программа состоит из модулей `main.cpp`, `menu.cpp` и `game.cpp`.

Файл `main.cpp` является основным модулем программы, в котором создается графическое окно и вызывается функция запуска меню. Соответсвующий заголовочный файл `menu.h` сожержит глобальные переменные, значения параметров меню, прототипы функций и определения структур `Button` (кнопки меню) и `Group` (для работы с результатами игр).

Глобальная переменная `speeds` определяет скорость движения вниз фигур тетриса в игре. Глобальная переменная `name` определяет для списка рекордов имя игрока.

Константа `N_BUTTONS` определяет количество кнопок в главном меню.

Структура `Button` используется для хранения данных о кнопках меню:

```
struct Button
{
   int left;
   int top;
   int width;
   int height;
   IMAGE *image;
};
```

Поле `left` определяет координаты кнопки по оси абсцисс, поле `top` координаты кнопки по оси ординат, поле `width` ширину кнопки в пикселях, `height` высоту кнопки в пикселях. Изображения кнопки храниться в поле `image`.

Структура `Group` используется для хранения данных о результатах игр:

```
struct Group {
   std::string nickname;
   int score;
};
```

Поле nickname определяет имя пользователя, а поле score очки набранные им в игре.

Функция load инициализирует данные о координатах кнопок меню и загружает необходимые изображения для вывода на экран пользователя. Функция start выводит на экран пользователя изображение и завершается при нажатии клавиши клавиатуры. Функция menu отвечает за отображение меню и взаимодейсвие с кнопками. Функция about отвечает за реализацию кнопки «о программе». Функция print_setting отвечает за отображение настроек. Функция settings отвечает за реализацию кнопки «настройки». Функция sort_groups отвечает за сортировку по убыванию очков игроков. Функция load_groups отвечает за загрузку из файла «records.txt» имен и очков пользователей. Функция records отвечает за реализацию кнопки «статистика». Функция print_groups отвечает за отображение имен и очков 10 лучших игроков. Функция close отвечает за реализацию кнопки «выход». Функция create_button отвечает за создание кнопок меню. Функция add_group отвечает за загрузку результата игры в структуру «Group». Функция select_button позволяет определить какую кнопку главного меню выбрал пользователь с помощью компьютерной мыши.

Заголовочны файл game.h сожержит глобальную переменную, значения параметров игры и прототипы функций.

Глобальная переменая points определяет количество очков набранных пользователем в течении игры.

Константы BOARD_WIDTH и BOARD_HEIGHT определяют высоту и ширину игрового поля. Константа BLOCK_SIZE определяет размер клетки игрового поля в пикселях. Константа BLOCK_COLORS определяет цвет клетки на игровом поле. Константа TETROMINOS определяет матрицы основных фигур тетриса.

Функция create_window отвечает за создание окна с игрой. Функция draw_block отвечает за отрисовку одной клетки на игровом поле. Функция create_tetromino отвечает за создание матрицы фигуры тетриса. Функция can_create_tetromino отвечает за проверу возможности поместить матрицу фигуры тетриса в матрицу игрового поля. Функция draw_-

`tetromino` отвечает за отрисовку матрицы фигуры тетриса на игровом поле. Функция `clear_tetromino` отвечает за стирание матрицы фигуры тетриса на игровом поле. Функция `put_tetromino` отвечает за соединение матрицы фигуры тетриса с матрицей игрового поля. Функции `can_move_down`, `can_move_right`, `can_move_left`: отвечают за проверку возможности переместить фигуру тетриса вниз, вправо, влево соответсвенно на игровом поле. Функция `can_rotate` отвечает за проверку возможности транспонировать матрицу фигуры тетриса на матрице игрового поля. Функции `move_right`, `move_left`, `move_down`: отвечают за перемещение фигуры тетриса вправо, влево, вниз соответственно на игровом поле. Функция `rotate_tetromino` отвечает за транспонирование матрицы фигуры тетриса. Функция `check_lines` отвечает за набирание очков игроком и удаление линий собранных из 10 клеток по горизонтале. Функция `infomation` отвечает за вывод информации о очках, скорости, собранных линий в игре. Функция `game` отвечает за реализацию игры тетрис

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы по программированию игры тетрис на C++, был разработан полностью функциональный вариант игры с интуитивно понятным пользовательским интерфейсом, игровой логикой и графикой. В рамках работы были реализованы следующие функциональности: отображение всех игровых элементов, основная логика игры, механизм перемещения и поворота фигур, обработка столкновений, алгоритм проверки завершения игры, настройка уровней сложности и другие функции, позволяющие игрокам наслаждаться игрой.

В процессе выполнения работы было проведено исследование стандартных правил игры тетрис, изучены основные принципы работы C++ и его средств разработки и написаны все необходимые алгоритмы и сценарии для корректного функционирования игры.

Результаты выполнения курсовой работы соответствуют поставленным целям и могут быть использованы как основа для дальнейших разработок и совершенствований игры.

ЛИТЕРАТУРА

1. Аммерал, Л. Принципы программирования в машинной графике / Л. Аммерал. – Мосвка : «Сол Систем», 1992. – 224 с.
2. Керниган, Б. Практика программирования / Б. Керниган, Р. Пайк. – Москва : Вильямс, 2004. – 288 с.
3. Левитин, А.В. Алгоритмы: введение в разработку и анализ / А.В. Левитин. – Москва : Вильямс, 2006. – 576 с.
4. Подбельский, В.В. Язык C++ : учебное пособие / В.В. Подбельский. – Москва : Финансы и статистика, 2003. – 560 с.
5. Шилдт, Г. Полный справочник по C++ / Г. Шилдт. – Москва : Вильямс, 2006. – 800 с.

ПРИЛОЖЕНИЕ 1

Текст программы

Файл main.cpp

```
#define WIDTH  540
#define HEIGHT 600

#include "menu.h"
#include "graphics.h"

int main()
{
   initwindow(WIDTH, HEIGHT);
   load();
   start();
   menu();
   closegraph();
   return 0;
}
```

Файл menu.h

```
#ifndef MENU_H
#define MENU_H

#include "graphics.h"

extern int speeds;
extern std::string name;
enum button_values { NONE = -1, GAME, ABOUT, SETTINGS, RECORDS,
                        EXIT, N_BUTTONS };

struct Button
{
   int left;
   int top;
   int width;
   int height;
   IMAGE *image;
};

struct Group {
   std::string nickname;
   int score;
};

void load();
void start();
void menu();
```

```
void about();
void print_setting();
void settings();
void sort_groups();
void load_groups();
void print_groups();
void records();
void close();

void create_button(int, int, int, const char*);
void add_group(std::string, int);
int  select_button();

#endif
```

Файл menu.cpp

```
#include <string>
#include <fstream>
#include "menu.h"
#include "game.h"
#include "graphics.h"

using namespace std;

Button buttons[N_BUTTONS];
IMAGE *menu_image, *about_image, *start_image;
int speeds = 500;
string name = "noname";
const int MAX_GROUPS = 100;
int num_groups = 0;
Group groups[MAX_GROUPS];

void add_group(string name1, int points1) {
   Group g;
   g.nickname = name1;
   g.score = points1;
   groups[num_groups] = g;
   num_groups++;
}

void load_groups() {
    ifstream infile("records.txt");
    while (true) {
        Group g;
        if (!(infile >> g.nickname >> g.score)) {
            if (infile.eof()) {
                break;
            }
        }
        groups[num_groups] = g;
```

```
        num_groups++;
    }

    infile.close();
}

void load()
{
   load_groups();
   menu_image  = loadBMP("images/menu.bmp");
   about_image = loadBMP("images/about.bmp");
   start_image = loadBMP("images/screensaver.bmp");

   create_button(GAME,  170, 100, "buttons/game.bmp");
   create_button(ABOUT, 170, 200, "buttons/about.bmp");
   create_button(SETTINGS, 170, 300, "buttons/settings.bmp");
   create_button(RECORDS, 170, 400, "buttons/records.bmp");
   create_button(EXIT,  170, 500, "buttons/exit.bmp");
}

void start()
{
   int r = 255;
   int g = 255;
   int b = 255;
   bool shift = true;

   while (true)
   {
      putimage(0, 0, start_image, COPY_PUT);
      setbkcolor(COLOR(238, 228, 203));

      if (shift)
      {
         r -= 15;
         g -= 15;
         b -= 15;
         if (r <= 0)
            shift = false;
      }
      else
      {
         r += 15;
         g += 15;
         b += 15;
         if (r >= 255)
            shift = true;
      }

      settextstyle(DEFAULT_FONT, HORIZ_DIR, 2);
      setcolor(COLOR(r,g,b));
      outtextxy(75, 550, "[Press any button!]");
```

```
        delay(50);

        if (kbhit())
        {
            break;
        }
    }
}


void menu()
{
    int state;

    while (true)
    {
        putimage(0, 0, menu_image, COPY_PUT);
        for (int i = 0; i < N_BUTTONS; i++)
        {
            putimage(buttons[i].left, buttons[i].top,
                     buttons[i].image, COPY_PUT);
        }

        state = NONE;
        while (state == NONE)
        {
            while (mousebuttons() != 1);
            state = select_button();
        }

        switch (state)
        {
            case GAME:  game();  break;
            case ABOUT: about(); break;
            case SETTINGS: settings(); break;
            case RECORDS: records(); break;
            case EXIT: return;
        }
    }
}

void about()
{
    putimage(0, 0, about_image, COPY_PUT);
    getch();
}

void print_setting()
{
    setcolor(BLACK);
    setbkcolor(COLOR(238, 228, 203));
```

```
    settextstyle(DEFAULT_FONT, HORIZ_DIR, 4);
    outtextxy(120 , 50, "Settings");
    settextstyle(DEFAULT_FONT, HORIZ_DIR, 3);
    outtextxy(220 , 170, "Name");
    settextstyle(DEFAULT_FONT, HORIZ_DIR, 3);
    outtextxy(170 , 270, "Speed");
    rectangle(50, 100, 490, 500);
    rectangle(170, 190, 360, 230);
    rectangle(200, 300, 330, 350);
    rectangle(350, 300, 400, 350);
    settextstyle(DEFAULT_FONT, HORIZ_DIR, 2);
    outtextxy(360 , 320, "->");
    rectangle(130, 300, 180, 350);
    settextstyle(DEFAULT_FONT, HORIZ_DIR, 2);
    outtextxy(140 , 320, "<-");
}

void settings()
{
    cleardevice();
    putimage(0, 0, menu_image, COPY_PUT);
    print_setting();
    stringstream speed;
    speed<< speeds;
    string numStr = speed.str();
    settextstyle(DEFAULT_FONT, HORIZ_DIR, 3);
    outtextxy(230 , 320, numStr.c_str());
    settextstyle(DEFAULT_FONT, HORIZ_DIR, 2);
    outtextxy(180, 200, name.c_str());


        while (true) {
            if (ismouseclick(WM_LBUTTONDOWN)) {
                int mx = mousex();
                int my = mousey();

                if (mx >= 170 && mx <= 360 && my >= 190 && my <= 230)
                {
                    name = "";
                    cleardevice();
                    print_setting();
                    settextstyle(DEFAULT_FONT, HORIZ_DIR, 3);
                    outtextxy(230 , 320, numStr.c_str());
                    while (name.length() < 10)
                    {
                        char c = getch();
                        if (c == KEY_BACKSPACE)
                        {
                            name.pop_back();
                            cleardevice();
                            print_setting();
                            settextstyle(DEFAULT_FONT, HORIZ_DIR, 3);
```

```
                    outtextxy(230 , 320, numStr.c_str());
                    settextstyle(DEFAULT_FONT, HORIZ_DIR, 2);
                    outtextxy(180, 200, name.c_str());
                }
                else if (c == KEY_ENTER || c == KEY_ESC)
                {
                    break;
                }
                else if (isalpha(c))
                {
                    name += c;
                    settextstyle(DEFAULT_FONT, HORIZ_DIR, 2);
                    outtextxy(180, 200, name.c_str());
                }
            }
        }

        if (mx >= 350 && mx <= 400 &&
            my >= 300 && my <= 350 && speeds > 100)
        {
            speeds -= 50;
            stringstream speed1;
            speed1<< speeds;
            numStr = speed1.str();
            settextstyle(DEFAULT_FONT, HORIZ_DIR, 3);
            outtextxy(230 , 320, numStr.c_str());
        }

        if (mx >= 130 && mx <= 180 &&
            my >= 300 && my <= 350 && speeds < 500)
        {
            speeds += 50;
            stringstream speed2;
            speed2<< speeds;
            numStr = speed2.str();
            settextstyle(DEFAULT_FONT, HORIZ_DIR, 3);
            outtextxy(230 , 320, numStr.c_str());
        }

        clearmouseclick(WM_LBUTTONDOWN);
    }
    if (kbhit())
    {
        int key = getch();
        if (key == KEY_ESC)
        {
            break;
        }
    }
}
}
```

```
void sort_groups() {
    bool swapped;
    do {
        swapped = false;
        for (int i = 0; i < num_groups - 1; i++) {
            if (groups[i].score< groups[i+1].score) {
                swap(groups[i], groups[i+1]);
                swapped = true;
            }
        }
    } while (swapped);
}

void print_groups() {
   setcolor(BLACK);
   setbkcolor(COLOR(238, 228, 203));
    for (int i = 0; i < num_groups; i++) {
      settextstyle(DEFAULT_FONT, HORIZ_DIR, 2);
      outtextxy(20, 100 + i * 50, groups[i].nickname.c_str());
      outtextxy(310, 100 + i * 50,
                    std::to_string(groups[i].score).c_str());
    }
}

void records()
{
   cleardevice();
   putimage(0, 0, menu_image, COPY_PUT);
   setcolor(BLACK);
   setbkcolor(COLOR(238, 228, 203));
   settextstyle(DEFAULT_FONT, HORIZ_DIR, 2);
   outtextxy(20, 25, "Name");
   outtextxy(310, 25, "Points");
   line (0, 50, 540, 50);
   line (300, 0, 300, 600);
   sort_groups();
   print_groups();
   getch();
}

void close()
{
   freeimage(menu_image);
   freeimage(about_image);
   freeimage(start_image);
   for (int i = 0; i < N_BUTTONS; i++)
   {
      freeimage(buttons[i].image);
   }
}

void create_button(int i, int left, int top,
```

```
                    const char *file_name)
{
   buttons[i].image  = loadBMP(file_name);
   buttons[i].left   = left;
   buttons[i].top    = top;
   buttons[i].width  = imagewidth(buttons[i].image);
   buttons[i].height = imageheight(buttons[i].image);
}

int select_button()
{
   int x, y;

   x = mousex();
   y = mousey();

   for (int i = 0; i < N_BUTTONS; i++)
   {
      if (x > buttons[i].left &&
          x < buttons[i].left + buttons[i].width &&
          y > buttons[i].top &&
          y < buttons[i].top + buttons[i].height)
      {
         return i;
      }
   }

   return NONE;
}
```

Файл game.h

```
#ifndef GAME_H
#define GAME_H

#include <stdbool.h>

extern int points;
void create_window();
void draw_block(int, int, int);
void draw_board();
void create_tetromino();
bool can_create_tetromino();
void draw_tetromino();
void clear_tetromino();
void put_tetromino();
bool can_move_down();
bool can_move_right();
bool can_move_left();
bool can_rotate();
void move_right();
```

```
void move_left();
void rotate_tetromino();
void check_lines();
void infomation();
void game();

#endif
```

Файл game.cpp

```
#include <ctime>
#include <chrono>
#include <fstream>
#include <string>
#include "graphics.h"
#include "game.h"
#include "menu.h"

using namespace std;

const int BOARD_WIDTH = 10;
const int BOARD_HEIGHT = 20;
const int BLOCK_SIZE = 30;

const int BLOCK_COLORS[] = {BLACK, BLUE, GREEN, CYAN, RED,
                                    MAGENTA, BROWN, LIGHTGRAY};

const int tetrominos[7][4][4] = {
       {
           {0, 0, 0, 0},
           {1, 1, 1, 1},
           {0, 0, 0, 0},
           {0, 0, 0, 0}
       },
       {
           {0, 0, 0, 0},
           {0, 2, 2, 0},
           {0, 2, 2, 0},
           {0, 0, 0, 0}
       },
       {
           {0, 3, 0, 0},
           {3, 3, 3, 0},
           {0, 0, 0, 0},
           {0, 0, 0, 0}
       },
       {
           {0, 4, 4, 0},
           {4, 4, 0, 0},
           {0, 0, 0, 0},
           {0, 0, 0, 0}
```

```
        },
        {
            {5, 5, 0, 0},
            {0, 5, 5, 0},
            {0, 0, 0, 0},
            {0, 0, 0, 0}
        },
        {
            {6, 6, 6, 0},
            {6, 0, 0, 0},
            {0, 0, 0, 0},
            {0, 0, 0, 0}
        },
        {
            {7, 7, 7, 0},
            {0, 0, 7, 0},
            {0, 0, 0, 0},
            {0, 0, 0, 0}
        },
    };

int board[BOARD_HEIGHT][BOARD_WIDTH];
bool exit_game = false;
int tetromino[4][4];
int tetromino_index;
int next_index;
int x = 3;
int y = 0;
int lines = 0;
int points = 0;


void create_window()
{
    setfillstyle(SOLID_FILL, COLOR(36, 38, 36));
    bar(BOARD_WIDTH * BLOCK_SIZE , 0,
       (BOARD_WIDTH + 10) * BLOCK_SIZE, BOARD_HEIGHT * BLOCK_SIZE);

    setfillstyle(SOLID_FILL, BLACK);
    bar(BOARD_WIDTH * BLOCK_SIZE + 15, 15,
          BOARD_WIDTH * BLOCK_SIZE + 195, 135);

    setcolor(WHITE);
    rectangle(BOARD_WIDTH * BLOCK_SIZE + 15, 15,
                  BOARD_WIDTH * BLOCK_SIZE + 195, 135);

    setfillstyle(SOLID_FILL, BLACK);
    setbkcolor(BLACK);
    bar(BOARD_WIDTH * BLOCK_SIZE + 15, 175,
          BOARD_WIDTH * BLOCK_SIZE + 195, 255);

    rectangle(BOARD_WIDTH * BLOCK_SIZE + 15, 175,
```

```
                BOARD_WIDTH * BLOCK_SIZE + 195, 255);

   settextstyle(DEFAULT_FONT, HORIZ_DIR, 2);
   setcolor(WHITE);
   outtextxy(BOARD_WIDTH * BLOCK_SIZE + 50, 190, "Points");

   setfillstyle(SOLID_FILL, BLACK);
   setbkcolor(BLACK);
    bar(BOARD_WIDTH * BLOCK_SIZE + 15, 290,
          BOARD_WIDTH * BLOCK_SIZE + 195, 370);

   rectangle(BOARD_WIDTH * BLOCK_SIZE + 15, 290,
                BOARD_WIDTH * BLOCK_SIZE + 195, 370);

   settextstyle(DEFAULT_FONT, HORIZ_DIR, 2);
   setcolor(WHITE);
   outtextxy(BOARD_WIDTH * BLOCK_SIZE + 50, 305, "Lines");

   setfillstyle(SOLID_FILL, BLACK);
   setbkcolor(BLACK);
   bar(BOARD_WIDTH * BLOCK_SIZE + 15, 405,
         BOARD_WIDTH * BLOCK_SIZE + 195, 485);

   rectangle(BOARD_WIDTH * BLOCK_SIZE + 15, 405,
                 BOARD_WIDTH * BLOCK_SIZE + 195, 485);

   settextstyle(DEFAULT_FONT, HORIZ_DIR, 2);
   setcolor(WHITE);
   outtextxy(BOARD_WIDTH * BLOCK_SIZE + 50, 420, "Speed");
}

void draw_block(int i, int j, int color)
{
   setfillstyle(SOLID_FILL, color);
   bar(i * BLOCK_SIZE, j * BLOCK_SIZE,
      (i + 1) * BLOCK_SIZE, (j + 1) * BLOCK_SIZE);

   if (color != 0){
      rectangle((i + 0.05) * BLOCK_SIZE, (j + 0.05) * BLOCK_SIZE,
      (i + 0.95) * BLOCK_SIZE, (j + 0.95) * BLOCK_SIZE);

   }
}

void draw_board()
{
   for (int j = 0; j < BOARD_HEIGHT; j++) {
      for (int i = 0; i < BOARD_WIDTH; i++) {
         draw_block(i, j, BLOCK_COLORS[board[j][i]]);
      }
   }
}
```

```
void create_tetromino()
{
   srand(time(NULL));
   tetromino_index = next_index;
   for (int i = 0; i < 4; i++)
   {
      for (int j = 0; j < 4; j++)
      {
         tetromino[i][j] = tetrominos[tetromino_index][i][j];
      }
   }

   for (int i = 0; i < 4; i++)
   {
      for (int j = 0; j < 4; j++)
      {
         if (tetromino[i][j] != 0)
         {
            draw_block(12 + j,1 + i, BLACK);
         }
      }
   }

   next_index =  rand() % 7;

   for (int i = 0; i < 4; i++)
   {
      for (int j = 0; j < 4; j++)
      {
         if (tetrominos[next_index][i][j] != 0)
            {
               draw_block(j + 12, i + 1,
                  BLOCK_COLORS[tetrominos[next_index][i][j]]);
            }
      }
   }
}

bool can_create_tetromino()
{
   for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (tetromino[i][j] != 0 && board[y + i][x + j] != 0)
            {
               return false;
            }
        }
    }
    return true;
```

```
}

void draw_tetromino()
{
   for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (tetromino[i][j] != 0)
            {
               draw_block(x + j,y + i,
                  BLOCK_COLORS[tetromino[i][j]]);
            }
        }
    }
}

void clear_tetromino()
{
   for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (tetromino[i][j] != 0)
            {
               draw_block(x + j,y + i, BLACK);
            }
        }
    }
}

void put_tetromino()
{
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (tetromino[i][j] != 0)
            {
               board[y + i][x + j] = tetromino[i][j];
            }
        }
    }

    draw_board();
}

bool can_move_down()
{
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            if (tetromino[i][j] != 0) {
```

```
                if (y + i >= BOARD_HEIGHT - 1) {
                    return false;
                }
                else if (board[y + i + 1][x + j] != 0) {
                    return false;
                }
            }
        }
    }
    return true;
}

bool can_move_right()
{
   for (int i = 0; i < 4; i++)
   {
      for (int j = 0; j < 4; j++)
      {
         if (tetromino[i][j] != 0)
         {
            if (x + j >= BOARD_WIDTH - 1 ||
               board[y + i][x + j + 1] != 0)
            {
               return false;
            }
         }
      }
   }
   return true;
}

bool can_move_left()
{
   for (int i = 0; i < 4; i++)
   {
      for (int j = 0; j < 4; j++)
      {
         if (tetromino[i][j] != 0)
         {
            if (x + j <= 0 || board[y + i][x + j - 1] != 0)
            {
               return false;
            }
         }
      }
   }
    return true;
}

bool can_rotate()
{
    for (int i = 0; i < 4; i++)
```

```
    {
         for (int j = 0; j < 4; j++)
        {
             if (tetromino[i][j] != 0)
             {
                 if (x + j > BOARD_WIDTH-1 ||
                    board[y + i][x + j] != 0 || x + j < 0)
                 {
                      return false;
                 }
             }
         }
     }
     return true;
}

void move_right()
{
    if (can_move_right())
    {
        clear_tetromino();
        x++;
        draw_tetromino();
    }
}

void move_left()
{
    if (can_move_left())
    {
        clear_tetromino();
        x--;
        draw_tetromino();
    }
}

void rotate_tetromino()
{
    int temp_tetromino[4][4];
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            temp_tetromino[i][j] = tetromino[i][j];
        }
    }

    if (tetromino_index != 0 && tetromino_index != 1)
    {
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
```

```
            {
                tetromino[i][j] = temp_tetromino[2-j][i];
            }
        }
    }
    else
    {
        for (int i = 0; i < 4; i++)
        {
            for (int j = 0; j < 4; j++)
            {
                tetromino[i][j] = temp_tetromino[3-j][i];
            }
        }
    }

     if (!can_rotate()) {
         for (int i = 0; i < 4; i++) {
             for (int j = 0; j < 4; j++) {
                 tetromino[i][j] = temp_tetromino[i][j];
             }
         }
     }
}

void check_lines()
{
    int c = 0;
    for (int i = BOARD_HEIGHT - 1; i >= 0; i--)
    {
        bool full_line = true;
        for (int j = 0; j < BOARD_WIDTH; j++)
        {
            if (board[i][j] == 0)
            {
                full_line = false;
                if (c != 0)
                {
                    if (c == 1)
                        points += 100;
                    else if (c == 2)
                        points += 300;
                    else if (c == 3)
                        points += 500;
                    else if (c == 4)
                        points += 1000;
                    lines += c;
                    c = 0;
                }
                break;
            }
        }
```

```
      if (full_line)
      {
         for (int k = i; k > 0; k--)
         {
            for (int j = 0; j < BOARD_WIDTH; j++)
            {
               board[k][j] = board[k - 1][j];
            }
         }
         for (int j = 0; j < BOARD_WIDTH; j++)
         {
            board[0][j] = 0;
         }
         i++;
         c++;
      }
   }
}

void infomation()
{
   stringstream point;
   point << points;
   string numStr = point.str();
   settextstyle(DEFAULT_FONT, HORIZ_DIR, 3);
   outtextxy((BOARD_WIDTH + 2.5) * BLOCK_SIZE ,
                  7 * BLOCK_SIZE, numStr.c_str());

   stringstream line;
   line << lines;
   string lineStr = line.str();
   settextstyle(DEFAULT_FONT, HORIZ_DIR, 3);
   outtextxy((BOARD_WIDTH + 2.5) * BLOCK_SIZE ,
                  11 * BLOCK_SIZE, lineStr.c_str());

   stringstream speed;
   speed << speeds;
   string speedStr = speed.str();
   settextstyle(DEFAULT_FONT, HORIZ_DIR, 3);
   outtextxy((BOARD_WIDTH + 2.5) * BLOCK_SIZE ,
                  15 * BLOCK_SIZE, speedStr.c_str());
}

void clear_board()
{
   exit_game = false;
   x = 3;
   y = 0;
   lines = 0;
   points = 0;
   for (int i = 0; i < BOARD_WIDTH; i++)
```

```
    {
        for (int j = 0; j < BOARD_HEIGHT; j++)
        {
            board[j][i] = 0 ;
        }
    }
}

void move_down()
{
    if (can_move_down())
    {
        clear_tetromino();
        y++;
        draw_tetromino();
    }
    else
    {
        put_tetromino();
        check_lines();
        if (lines >= 10 && speeds > 300)
        {
            speeds = 300;
        }
        if (lines >= 20 && speeds > 200)
        {
            speeds = 200;
        }
        if (lines >= 50 && speeds > 100)
        {
            speeds = 100;
        }
        infomation();
        x = 3;
        y = 0;
        if (can_create_tetromino())
        {
            create_tetromino();
        }
        else
        {
            exit_game = true;
        }
    }
}

void game()
{
    clear_board();
    create_window();
    next_index = rand() % 7;
    infomation();
```

```
    create_tetromino();
    draw_tetromino();
    while (!exit_game)
    {
        draw_board();
        move_down();

        auto start = chrono::steady_clock::now();
        while (chrono::duration_cast<chrono::milliseconds>
            (chrono::steady_clock::now() - start).count() < speeds)
        {
            if (kbhit())
            {
                int key = getch();
                switch(key)
                {
                    case KEY_RIGHT:
                        move_right();
                        break;
                    case KEY_LEFT:
                        move_left();
                        break;
                    case KEY_DOWN:
                        move_down();
                        break;
                    case KEY_UP:
                        rotate_tetromino();
                        break;
                    case KEY_ESC:
                        speeds = 500;
                        return;
                }
            }
        }
    }
    ofstream outfile("records.txt", ios::app);
    string data = name + " " + to_string(points);
    outfile << data << endl;
    outfile.close();
    add_group(name, points);
    clear_board();
    speeds = 500;
}
```

ПРИЛОЖЕНИЕ 2

Руководство пользователя

При запуске программы пользователю представится заставка игры, которую можно пропустить, нажав любую клавишу на клавиатуре, чтобы перейти в главное меню. В меню будут представлены кнопки: «Начать игру», «О программе», «Настройки», «Статистика» и «Выход». Для взаимодействия с каждой кнопкой необходимо щелкнуть на ней левой кнопкой мыши.

При выборе «Начать игру» пользователь сможет сыграть в игру, которая будет соответствовать стандартным правилам тетриса. Основное правило игры заключается в перемещении кусочков фигур по игровому полю в таком порядке, чтобы собрать линии из десяти клеток по горизонтали. Игрок получает очки за каждую собранную линию, и количество очков зависит от количества одновременно убранных линий. При игре в правом верхнем углу отображается следующий кусочек, который поможет игроку сделать правильный ход. Игра окончится, если новый кусочек не может быть помещён на игровое поле. Результат каждой игры будет сохранён, чтобы пользователь мог просмотреть свои достижения в разделе «Статистика». Чтобы выйти из игры, достаточно нажать клавишу ESC на клавиатуре и вернуться в главное меню.

При выборе «О программе» пользователь сможет узнать о правилах игры и создателе игры. Чтобы закрыть это окно, нужно нажать клавишу ESC .

При выборе «Настройки» пользователь сможет изменить имя персонажа и скорость падения кусочков в игре. Имя пользователя отображается в разделе «Статистика». Чтобы закрыть это окно, нужно нажать клавишу ESC.

Выбрав «Статистика», пользователь сможет увидеть результаты 10 лучших игр. Чтобы закрыть это окно, нужно нажать клавишу ESC.

Выбирая «Выход», программа завершит свою работу.

ПРИЛОЖЕНИЕ 3

Примеры выполнения программы

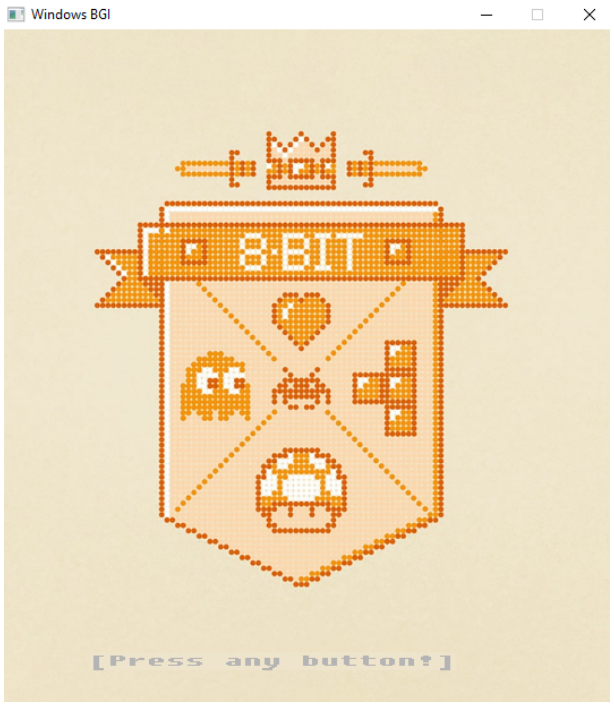

Рисунок П3.1 – Заставка

Рисунок П3.2 – меню

Рисунок П3.3 – о программе

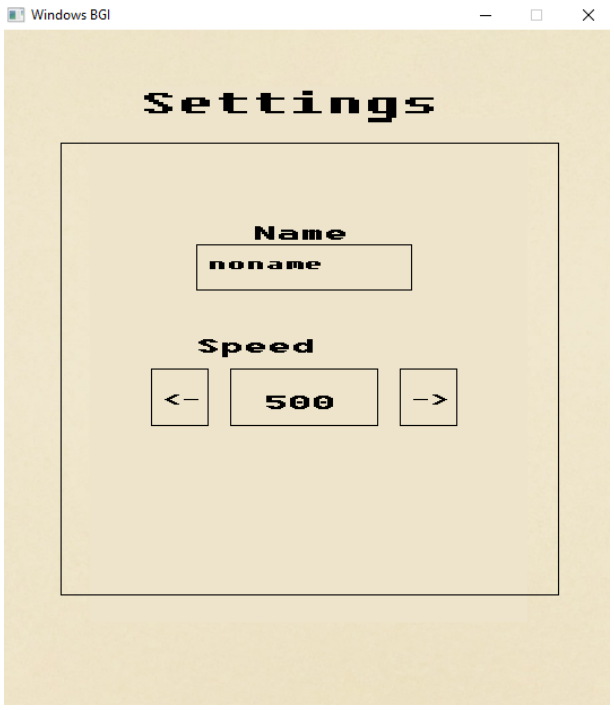

Рисунок П3.4 – настройки

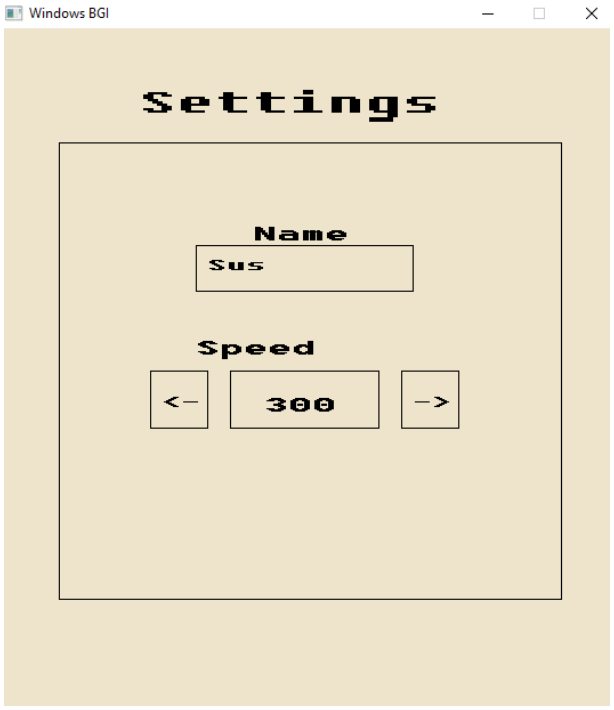

Рисунок П3.5 – настройки

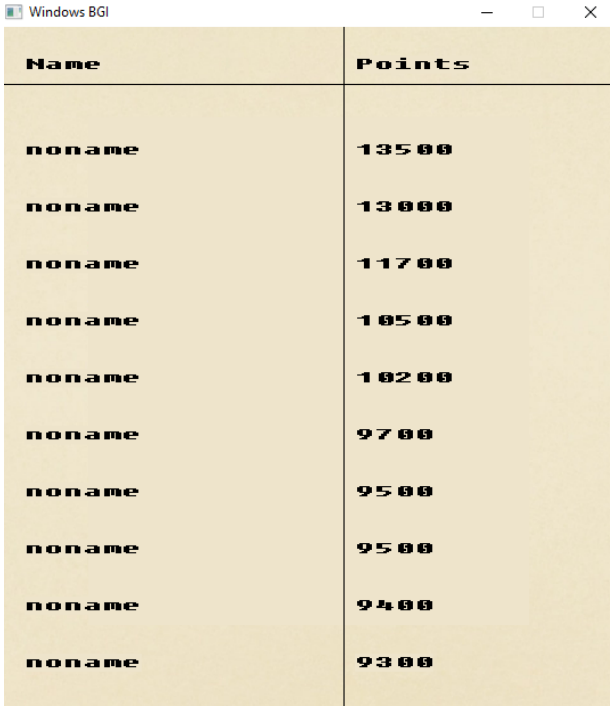

Рисунок П3.6 – Статистика

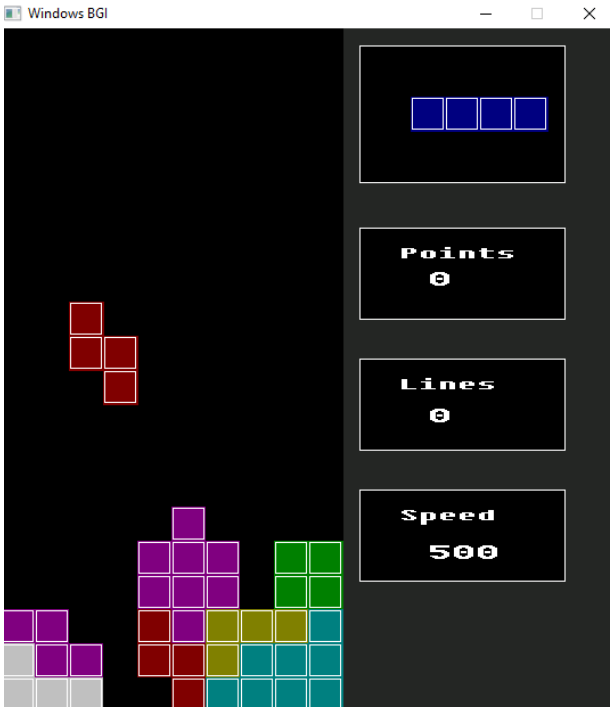

Рисунок П3.7 – игра

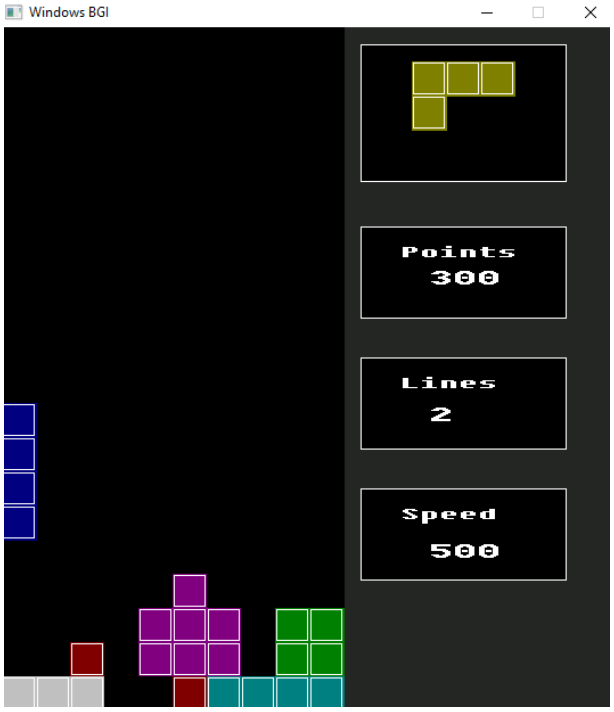

Рисунок П3.8 – игра

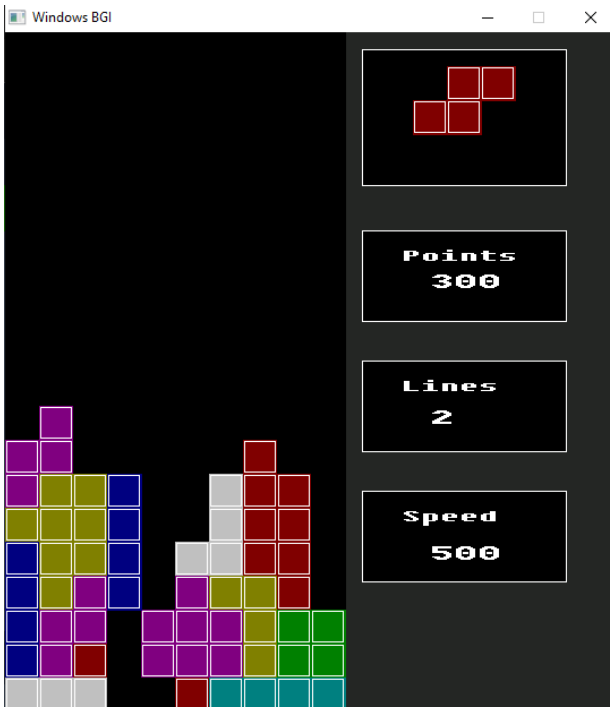

Рисунок П3.9 – игра

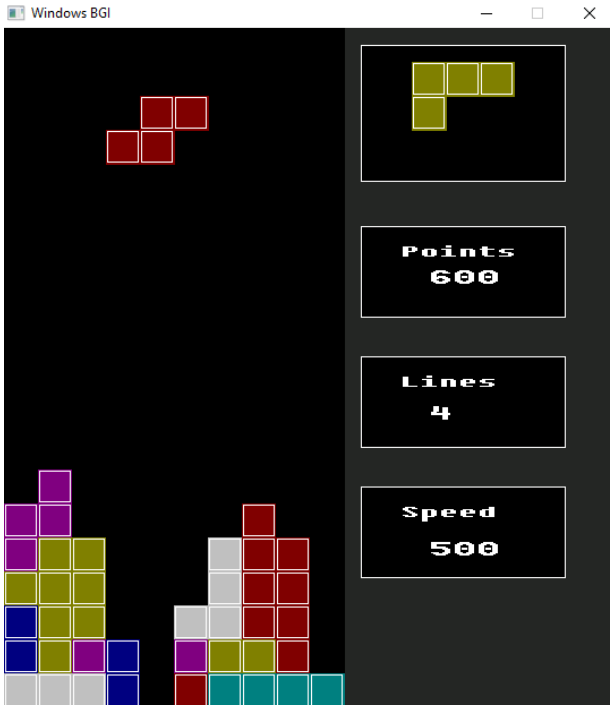

Рисунок П3.10 – игра