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 «2048»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ
по дисциплине «Языки программирования»
ЮУрГУ–020301.2023.042.ПЗ КР

Автор работы,
студент группы ЕТ-111
_____ Хисматуллин В.В.
«____» _____ 2023 г.

Руководитель работы,
старший преподаватель
_____ Сурин В.А.
«____» _____ 2023 г.

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

Челябинск 2023

АННОТАЦИЯ

Хисматуллин В.В. Разработка игры «2048». – Челябинск: ЮУрГУ, ЕТ-111, 2023. – 41 с., 8 ил., библиогр. список – 5 наим., 3 прил.

Целью работы является разработка компьютерной игры «2048». 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 чисел на игровом поле, алгоритм передвижения и соединения чисел. 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 «2048». 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 случайных чисел на игровом поле, алгоритм передвижения и соединения этих чисел, механизм управления игровым процессом;

– 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 «2048».

После запуска программы появляется заставка игры. После нажатия любой клавиши на клавиатуре открывается главное меню программы, которое содержит пункты «Начать игру», «Настройки», «Об игре» 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 размер игрового поля (от 4×4 до 7×7). После выбора игрового поля происходит возврат в главное меню. Схема окна с настройками показана на рисунке 1.2.

При выборе пункта «Об игре» появляется окно с информацией о правилах игры, а также сведения о разработчике программы.

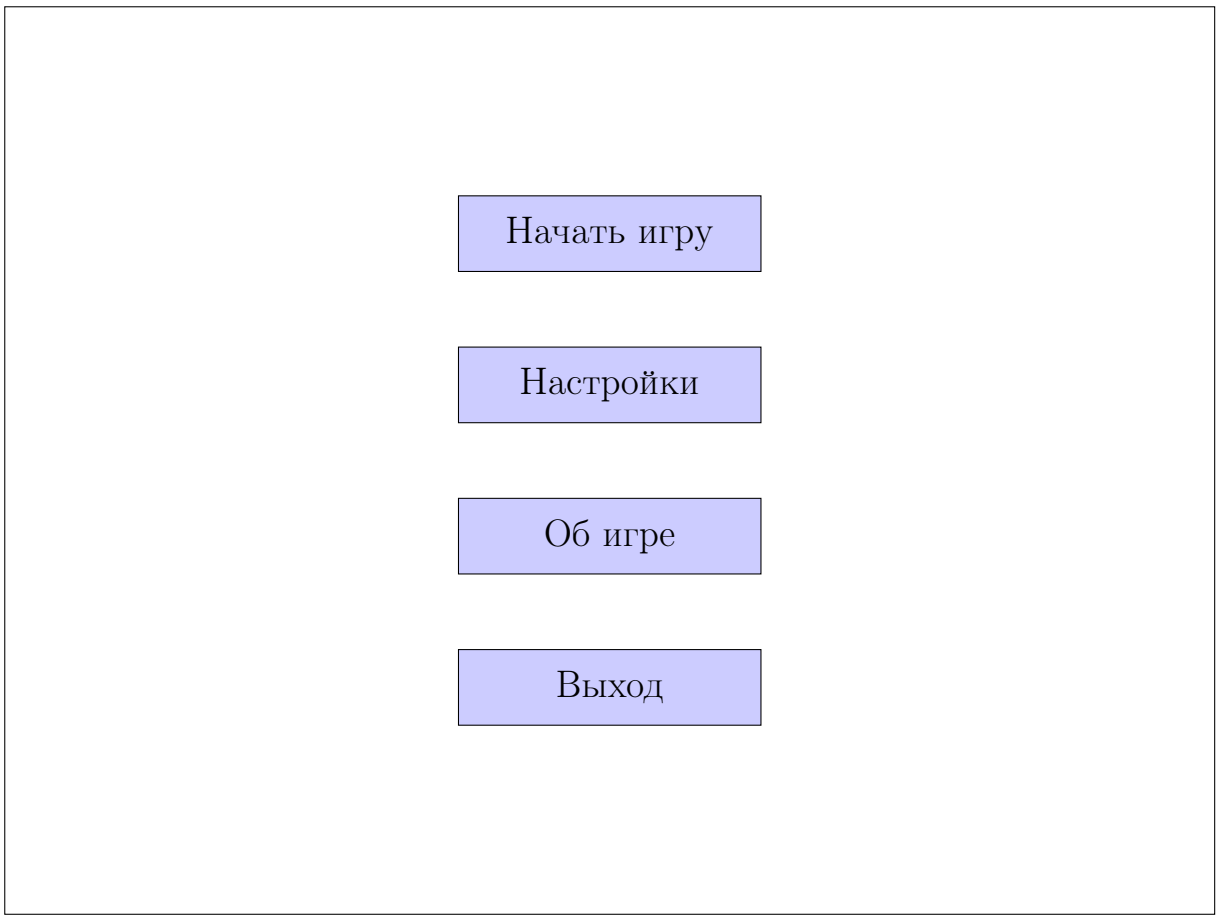

Рисунок 1.1 – Схема главного меню

При выборе пункта «Выход» программа завершает работу.

При выборе пользователем пункта «Начать игру» появляется игровое окно (рисунок 1.3). В центральной части окна расположено игровое поле.

Для перемещения цифр используются стрелки на клавиатуре.

Сверху от игрового поля расположены текстовые поля с указанием текущего количества очков и рекорда в выбранном режиме игры.

Цифры на игровом поле появляются в случайных ячейках поля. При соединении двух цифр, счетчик очков увеличивается на их сумму.

Если не остается возможных ходов появляется сообщение о проигрыше, при нажатии на определенную кнопку на клавиатуре, происходит возврат в главное меню игры, рекорд сохраняется.

При нажатии на клавишу Backspace поле возвращается на прошлый ход, счетчик очков так же возвращается в прошлое состояние. При нажатии на кнопку Esc происходит возврат в главное меню программы.

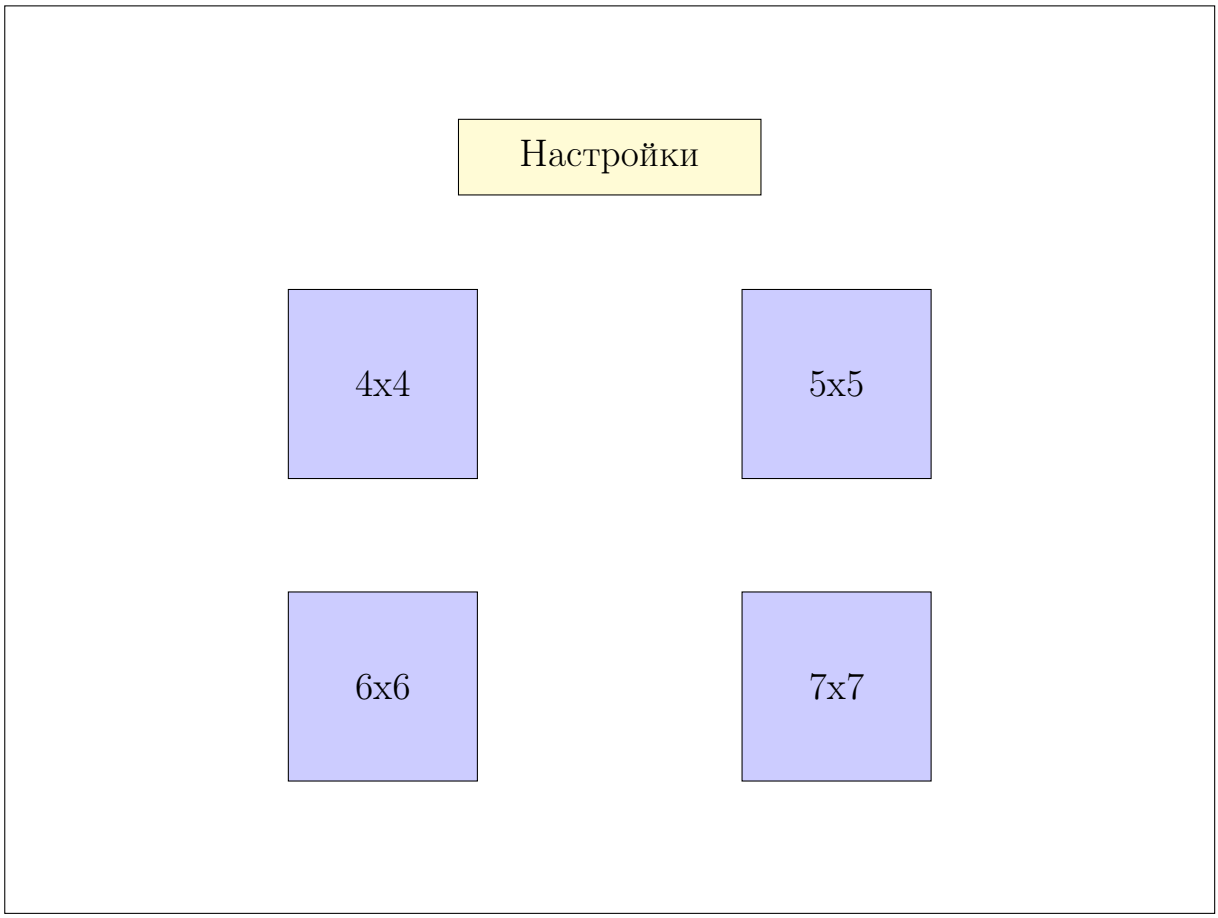

Рисунок 1.2 – Схема окна с настройками

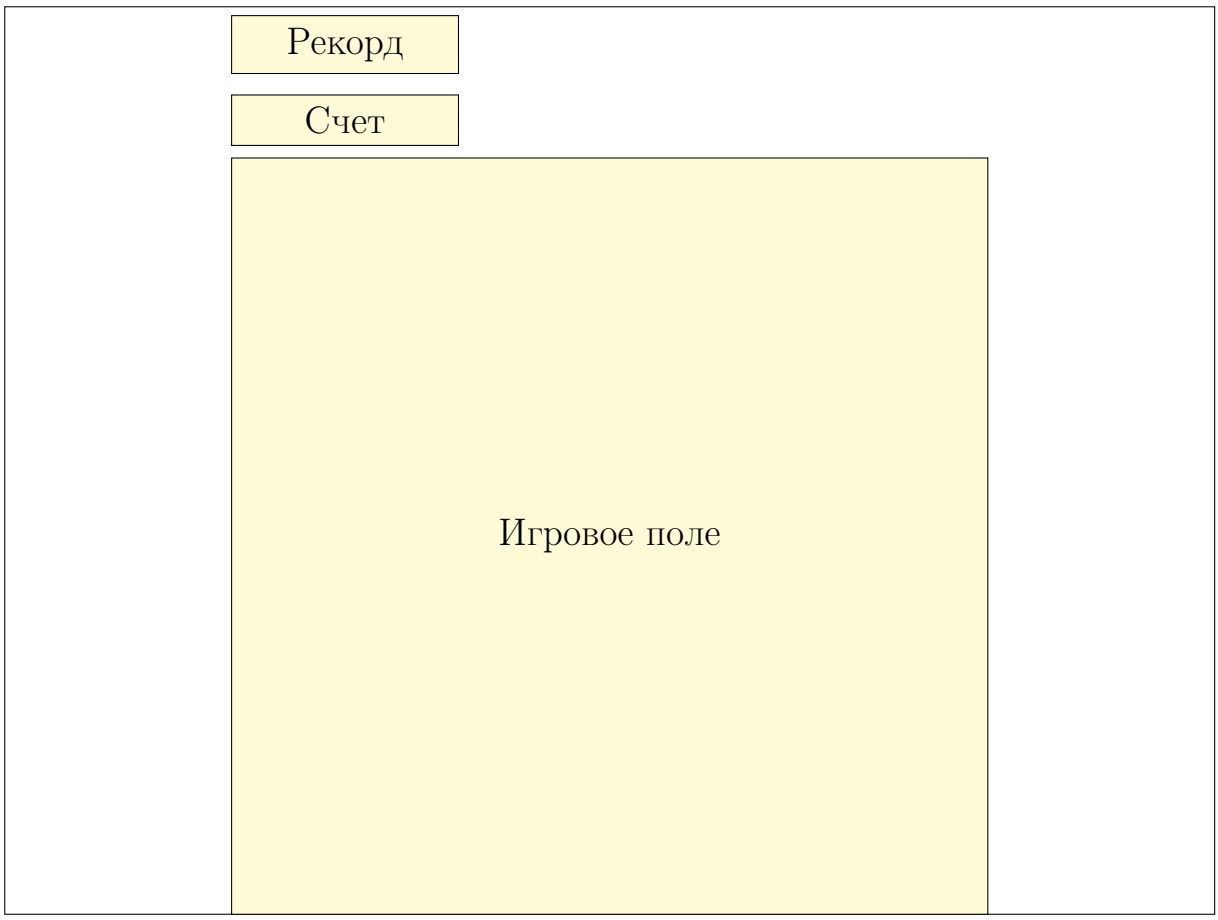

Рисунок 1.3 – Схема игрового окна

2 ФОРМАЛИЗАЦИЯ ЗАДАЧИ

Игровое поле будет представлять двумерную матрицу цифр A, размера $n \times n$ В каждой позиции матрицы будет храниться соответствующее значение игровой ячейки.

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$$

Игровое поле рисуется из примитив (рисунок 2.1). Ось X направлена вправо, ось Y - вниз. Отсчет ведется с точки с координатами (0;0) из верхнего левого угла.

Сначала рисуется серый прямоугольник - фон игровой сетки. Сетка игрового поля рисуется линиями, так получаются клетки игрового поля.

Размер каждой клетки a_s будет рассчитываться по формуле $(\frac{f_s}{f_n}) \times (\frac{f_s}{f_n})$, где f_s – размер игрового поля без стенок в пикселях, f_n – выбранный режим игры(количество клеток по горизонтали и вертикали). Толщина стенок игрового поля определяется переменной w_w. Таким образом размер игрового поля в пикселях находиться по формуле

$$(f_s + w_w \times (f_n + 1)) \times (f_s + w_w \times (f_n + 1))$$

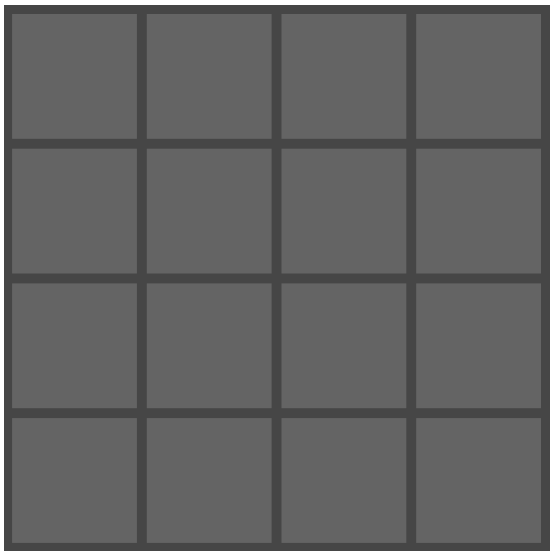

Рисунок 2.1 – Пример игрового поля 1

Координаты верхнего левого угла игровой сетки – $(x_0; y_t)$, верхнего правого угла – (w-x_0; y_t), нижнего левого угла – $(x_0; y_d)$, нижнего правого угла – (w-x_0; y_d) соответственно, где w - ширина игрового окна,, x_0 - смещение по x, y_t - координаты верхней стороны, y_d - координаты нижней стороны.

Если в матрице игрового поля есть цифра, то она рисуется на игровом поле. Сначала рисуется фон цифры, а затем сама цифра (рисунок 2.2). Каждая цифра рисуется по матрице структур. В этой структуре определены поля текущего значения и текущей позиции цифры, а так же позиции в которую перемещается цифра. Анимация движения цифр будет реализована с помощью этой структуры. Цифры будут перемещаться с определенным шагом и отрисовываться после каждого такого шага.

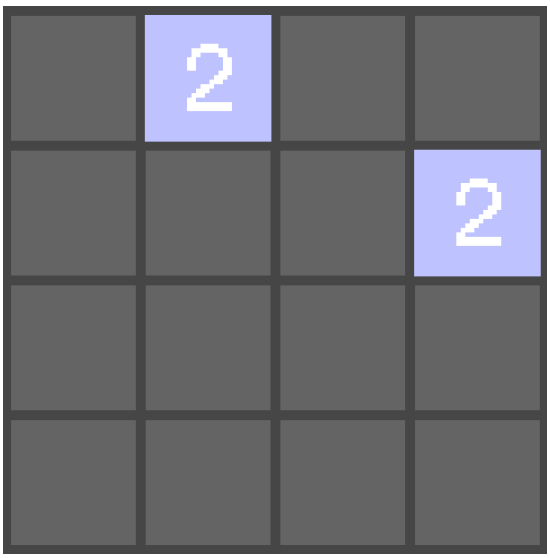

Рисунок 2.2 – Пример игрового поля 2

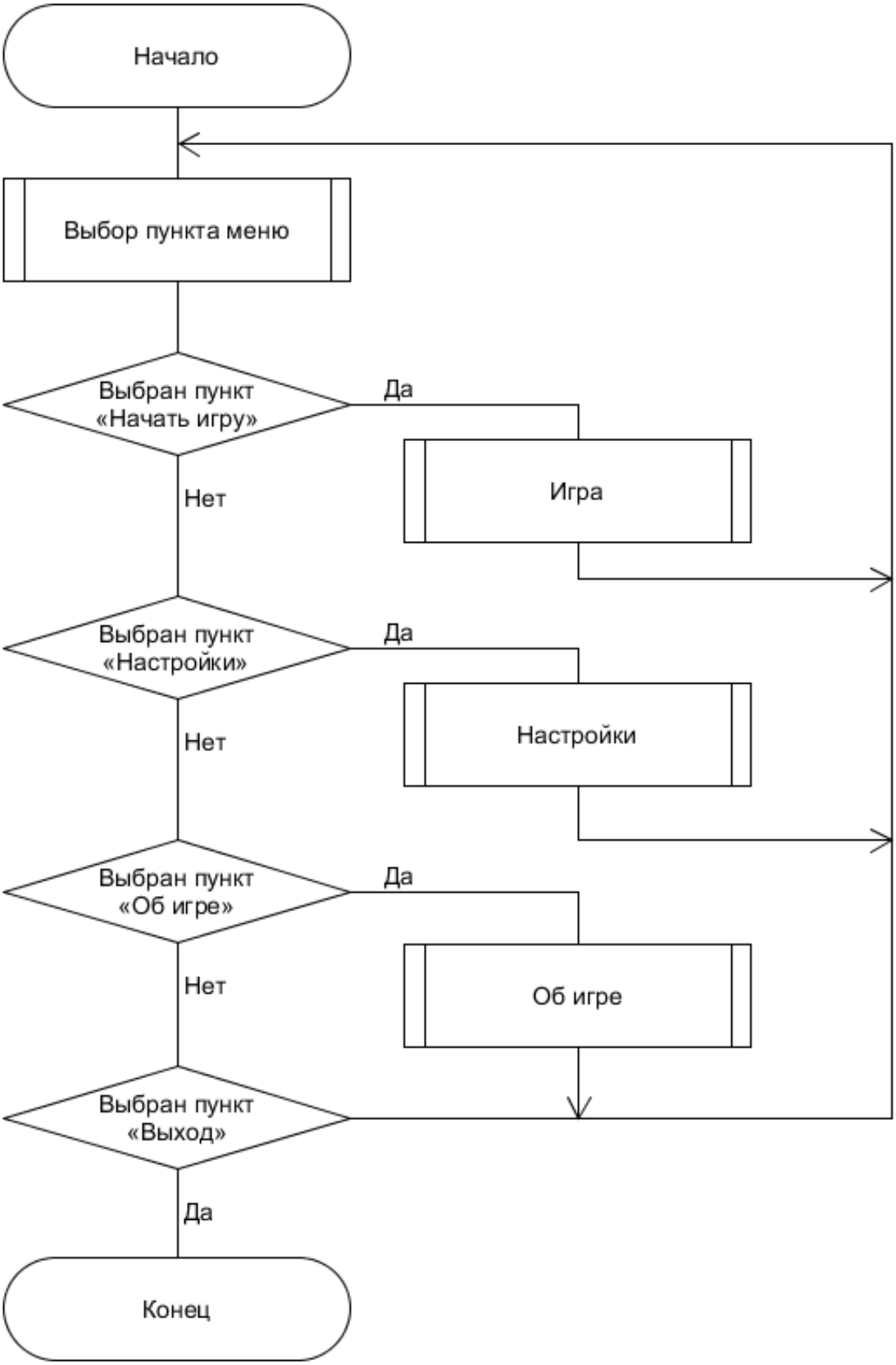

Рисунок 3.1 – Алгоритм работы программного меню

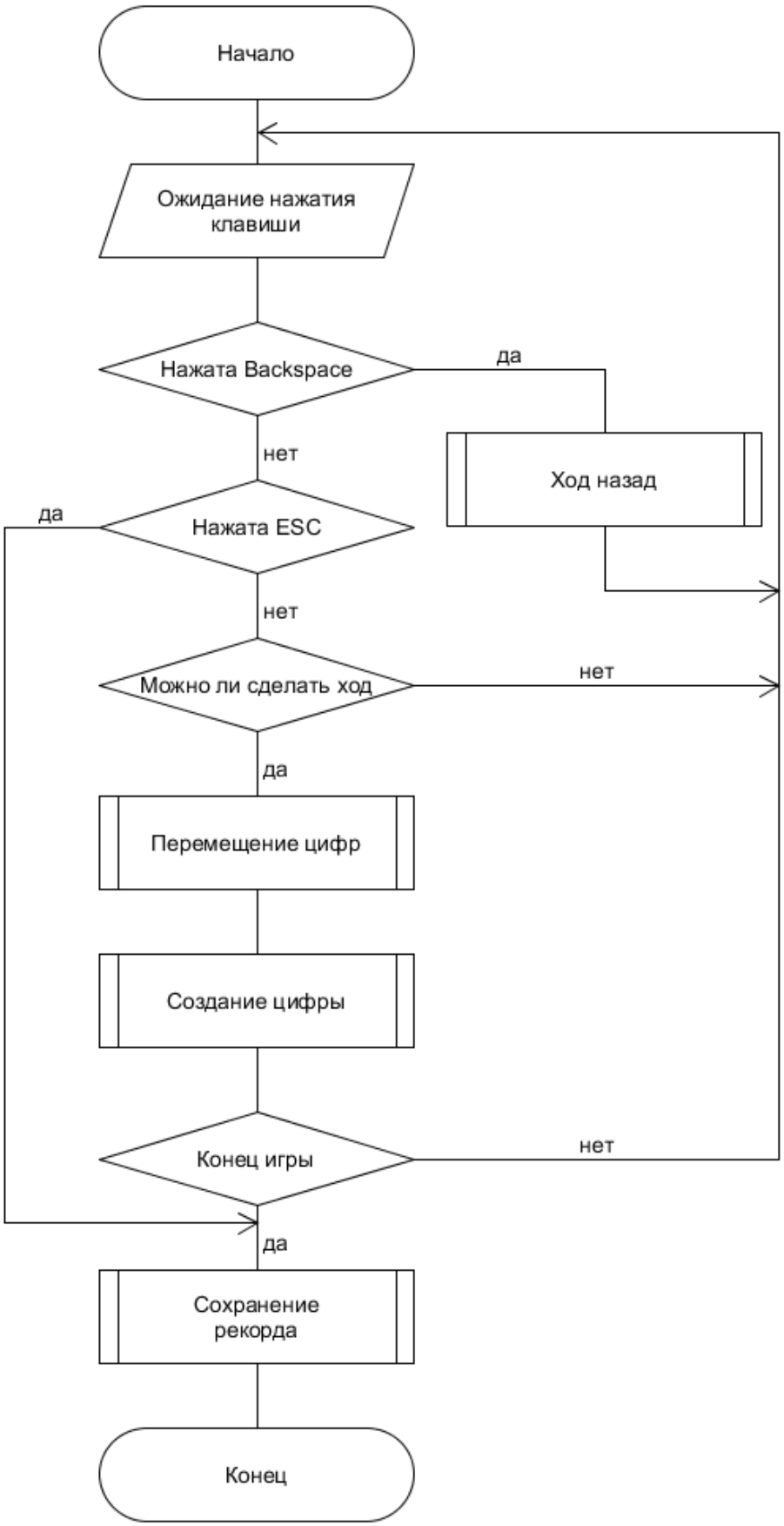

Рисунок 3.2 – Алгоритм игрового процесса

Рисунок 3.3 – Алгоритм генерации чисел

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. Соответствующий заголовочный файл `menu.h` содержит значения параметров меню, прототипы функций и определение структуры `Button` (кнопка меню).

Константы `WIDTH`, `HEIGHT` определяют ширину и высоту игрового окна

Для идентификации кнопок введено перечисление `button_values`. Константа `N_BUTTONS` определяет количество кнопок в главном меню. Константы `GAME`, `SETTINGS`, `ABOUT`, `EXIT` определяют индексы кнопок в массиве структур кнопок . Константа `S_BUTTONS` определяет количество кнопок в настройках. Константы `FOUR`, `FIVE`, `SIX`, `SEVEN` определяют индексы кнопок настроек в массиве структур кнопок. При отрисовке кнопок настроек перебор ведется с `N_BUTTONS` + 1.

Структура `Button` используется для хранения данных о кнопке 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. Изображение кнопки хранится в поле `image`. Функция `load` инициализирует данные о координатах кнопок меню и загружает необходимые изображения для вывода на экран пользователя. Функция `select_button` позволяет определить какую кнопку глав-

ного меню выбрал пользователь с помощью компьютерной мыши. Функция принимает переменную типа `int`, если она ровна 0, то значит выбор кнопки происходит в главном меню, если же 1, то выбор кнопки происходит в настройках. Функция возвращает позицию в массиве выбранной кнопки. Функция `create_button` создает кнопку в i-ой позиции массива кнопок. В функцию передаются четыре переменные. Переменные типа `i`, `left`, `top`, определяют индекс массива который надо изменить, позицию левой стороны кнопки, и позицию верхней стороны кнопки соответственно. Так же переменная типа const char - `*file_name`, в которой указывается путь до изображения, которое необходимо загрузить в кнопку. Функция `start` выводит на экран заставку игры. Функция `menu` выводит на экран пользователя изображения меню и соответствующие изображения кнопок меню, и позволяет пользователю выбрать один из пунктов меню. Функция `settings` выводит на экран настройки игры, в котором можно выбрать размер игрового поля игры. Функция `about` выводит на экран информацию об игре. Функция `close` очищает память от всех загруженных изображений.

В модуле `menu.cpp` определены переменные типа `IMAGE` в которых храниться соответствующее изображение. Каждый элемент массива `buttons` определяет кнопку, главном меню и настроек игры. Переменная `field_size` определяет размерность поля, изменяется в настройках и передается в модуль `game.cpp`.

Заголовочный файл `game.h` содержит значения параметров игры, прототипы функций и определение структуры `Cell` (клетка игрового поля).

Структура `Cell` используется для хранения данных об игровой клетке.

```
struct Cell
{
    long long data;
    int pos_x;
    int pos_y;
    int size;
    int red;
    int green;
    int blue;
    int next_x;
    int next_y;
};
```

Поле `data` определяет значение игровой клетки. Поля `pos_x` и `pos_y` определяют позицию игровой клетки по координатам в игровом окне. Поле `size` определяет размер (ширину) клетки. Поля `red`, `green` и `blue` определяют цвет фона клетки в RGB. Поля `next_x` и `next_y` определяют позицию в координатах, в которую перемещается клетка.

Константа `FIELD_WALL_WIDTH` определяет ширину стенок игрового поля в пикселях. Константы `X_OFFSET` и `Y_OFFSET` определяют смещение игрового поля относительно начала координат.

В модуле `game.cpp` определены целочисленные массивы `playing_field` и `last_field` отвечающие за текущее и предыдущее состояние игрового поля. Массив `Cells` определяет данные игровых клеток. Переменные `free_fields` и `last_free_fields` определяют количество свободных клеток на текущем и предыдущем игровом поле. Переменные `win_image` и `lose_image` определяют изображения победы и проигрыша. Переменная `speed` определяет скорость движения цифр во время анимации. Переменная `field_width` определяет ширину игрового поля. Переменные `win` и `last_win` определяют была ли победа на текущем и предыдущем ходах. Переменные `score` и `last_score` определяют текущее и предыдущее количество очков. Переменная `record` определяет значение рекорда. Массив `move_pos` определяет позиции цифр которые находятся в движении в массиве `Cells`. Передаваемые переменные `field_size` и `background_image`

определяют размер игрового поля и фоновое изображения игрового окна соответственно.

Функция `game` запускает и заканчивает игровой процесс. Функция `load_game` загружает изображения и устанавливает все игровые параметры по умолчанию. Функция `create_field` создает в памяти массивы отвечающие за игровой процесс. Функиция `delete_field` очищает память от изображений и массивов.

Функция `play` запускает процесс игры в котором игрок может управлять цифрами. Если игрок выиграл или проиграл функция выводит соответствующее сообщение.

Функции `open_record` и `save_record` считывает рекорд из файла и сохраняет рекорд в файл соответственно.

Функция `draw_field` отрисовывает игровое поле. Функция `draw_numbers` отрисовывает цифры по координатам из структуры `Cells`. Функция `new_number` создает цифру на случайной позиции игрового поля.

Функция `find_num` возвращает число которое определяет коэффицент цвета числа. Функция принимает значение числа.

Функция `save_field` запоминает значения текущего поля в переменные предыдущего хода. Функция `back_step` возвращает игровое поле в состояние предыдущего хода.

Функция `can_move` и `can_combine` проверяют можно ли сделать ход и можно ли сделать соединение цифр в выбранном направлении. Функции принимают значение нажатой клавиши и перебирают массив игрового поля в нужном направлении.

Функция `move` принимает значение нажатой клавиши и двигает цифры в нужном направлении.

Функция `do_animation` выполняет анимацию в выбранную сторону. Функция принимает значения скорости по координатам x и y и количество цифр которые участвуют в анимации. Функция `refresh_field` обновляет значения массива `Cells` в соответствии с массивом игрового поля `playing_field`.

Функция `check_lose` проверяет проиграна ли игра.

Функция `check_win` проверяет выиграна ли игра. Функция `draw_over`

отрисовывает окно победы или поражения. Функция принимает изображение которое необходимо вывести.

ЗАКЛЮЧЕНИЕ

Была разработана компьютерная программа реализующая игру «2048». Разработан алгоритм главного меню игры. Разработан алгоритм отрисовки игрового поля и генерации чисел и механизм управления с помощью многоальтернативного выбора. Алгоритм прорисовки игрового поля реализован с помощью функции, которая отрисовывает игровое поле в соответствии с матрицей структур. Таким образом, все поставленные задачи были выполнены. Цель курсовой работы достигнута.

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

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

using namespace std;

int main()
{
   initwindow(WIDTH, HEIGHT, "2048");
   load();
   start();
   menu();
   closegraph();
   return 0;
}
```

Файл menu.h

```
#ifndef MENU_H
#define MENU_H
#include "graphics.h"

const int WIDTH = 1280, HEIGHT = 800;

enum button_values { NONE = -1, GAME, SETTINGS,
                     ABOUT, EXIT, N_BUTTONS, FOUR,
                     FIVE, SIX, SEVEN, S_BUTTONS };

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

void load();
void start();
void menu();
void settings();
void about();
void close();

void create_button(int, int, int, const char*);
int  select_button(int);
```

```
#endif
```

Файл menu.cpp

```
#include "menu.h"
#include "game.h"
#include "graphics.h"

using namespace std;

Button buttons[S_BUTTONS];
IMAGE *start_image, *background_image,
            *menu_image, *settings_image, *about_image;
IMAGE *four, *five, *six, *seven;
int field_size = 4;

void load()
{
   start_image = loadBMP("images/start.bmp");
   background_image = loadBMP("images/background.bmp");
   menu_image = loadBMP("images/menu.bmp");
   about_image = loadBMP("images/about.bmp");
   settings_image = loadBMP("images/settings.bmp");

   create_button(GAME,  540, 250, "buttons/game_button.bmp");
   create_button(SETTINGS,  515, 350,
      "buttons/settings_button.bmp");
   create_button(ABOUT, 540, 450, "buttons/about_button.bmp");
   create_button(EXIT,  540, 550, "buttons/exit_button.bmp");
   create_button(FOUR, 320, 250, "buttons/4x4_button.bmp");
   create_button(FIVE, 760, 250, "buttons/5x5_button.bmp");
   create_button(SIX, 320, 500, "buttons/6x6_button.bmp");
   create_button(SEVEN, 760, 500, "buttons/7x7_button.bmp");
}

void start()
{
   putimage(0, 0, start_image, COPY_PUT);
   swapbuffers();
   getch();
}

void menu()
{
   while (true)
   {
      int state = NONE;
      setfillstyle(SOLID_FILL, WHITE);
      while (state == NONE)
      {
```

```
            while (mousebuttons() != 1)
            {
               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, TRANSPARENT_PUT);
                  if (mousex() > buttons[i].left &&
                     mousex() < buttons[i].left + buttons[i].width &&
                     mousey() > buttons[i].top &&
                     mousey() < buttons[i].top + buttons[i].height)
                  {
                     setfillstyle(SOLID_FILL, WHITE);
                     bar (buttons[i].left,
                           buttons[i].top + 0.96 * buttons[i].height,
                           buttons[i].left + buttons[i].width,
                           buttons[i].top + buttons[i].height);
                  }
               }
               swapbuffers();

               if(kbhit())
                  getch();

               state = select_button(0);
            }
         }


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

   void settings()
   {
      int state = NONE;
      while (state == NONE)
      {
         while (mousebuttons() != 1)
         {
            putimage(0, 0, settings_image, COPY_PUT);
            for (int i = N_BUTTONS + 1; i < S_BUTTONS; i++)
            {
               putimage(buttons[i].left, buttons[i].top,
                              buttons[i].image, COPY_PUT);
               if (mousex() > buttons[i].left &&
```

```
                mousex() < buttons[i].left + buttons[i].width &&
                mousey() > buttons[i].top &&
                mousey() < buttons[i].top + buttons[i].height)
            {
                setfillstyle(SOLID_FILL, WHITE);
                bar (buttons[i].left,
                      buttons[i].top + 0.96 * buttons[i].height,
                      buttons[i].left + buttons[i].width,
                      buttons[i].top + buttons[i].height);
            }
        }
        swapbuffers();

        if (kbhit())
        {
            int key = getch();
            if (key == KEY_ESC)
                return;
        }

        state = select_button(1);
      }
   }
   field_size = state - N_BUTTONS + 3;
}

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

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

   for (int i = N_BUTTONS + 1; i < S_BUTTONS; i++)
      freeimage(buttons[i].image);

   freeimage(start_image);
   freeimage(background_image);
   freeimage(menu_image);
   freeimage(settings_image);
   freeimage(about_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 choice)
{
    int start, end;
    if (choice == 0)
    {
        start = 0;
        end = N_BUTTONS;
    }
    else if (choice == 1)
    {
        start = N_BUTTONS + 1;
        end = S_BUTTONS;
    }

    int x, y;

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

    for (int i = start; i != end; 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 "graphics.h"

const int X_OFFSET = 340, Y_OFFSET = 150;
const int FIELD_WALL_WIDTH = 10;

struct Cell
{
    long long data;
    int pos_x;
```

```
    int pos_y;
    int size;

    int red;
    int green;
    int blue;

    int next_x;
    int next_y;
};

void game();
void play();
void load_game();
void create_field();
void delete_field();

void open_record();
void save_record();

void draw_field();
void draw_numbers();
void new_number();
int find_num(int);

void save_field();
void back_step();

bool can_move(int);
bool can_combine(int);
void move(int);
void do_animation(int, int, int);
void refresh_field();

bool check_lose();
bool check_win();

void draw_over(IMAGE*);

#endif
```

Файл game.cpp

```
#define FONT_TEXT 3
#include <ctime>
#include <fstream>
#include "game.h"
#include "menu.h"
#include "graphics.h"

using namespace std;
```

```
extern int field_size;
extern IMAGE *background_image;

IMAGE *win_image, *lose_image;
Cell **Cells;

int **move_pos;

int speed;
int field_width;

long long **play_field;
int free_fields;
bool win;
int score, record;

long long **last_field;
int last_score, last_free_fields;
bool last_win;

void game()
{
   cleardevice();
   swapbuffers();

   load_game();
   create_field();
   open_record();
   play();
   delete_field();
}

void load_game()
{
   field_width = 600;
   while (field_width % field_size != 0)
      field_width++;

   free_fields = field_size * field_size;
   score = 0;
   last_score = 0;

   speed = field_size * 5;

   win = false;
   last_win = false;

   win_image = loadBMP("images/win.bmp");
   lose_image = loadBMP("images/over.bmp");
}
```

```
void play()
{
   srand(time(NULL));

   while(true)
   {
      int key;
      draw_field();
      draw_numbers();
      swapbuffers();

      key = getch();
      if (key == 0)
         continue;

      if (key == KEY_ESC)
         break;
      else if (key == KEY_BACKSPACE)
      {
         back_step();
         continue;
      }

      if (can_move(key) == true ||
         can_combine(key) == true)
      {
         save_field();
         move(key);
      }
      else
         continue;

      if (free_fields > 0)
         new_number();

      if (check_lose() == true)
      {
         draw_over(lose_image);

         int choice = getch();
         while (choice != KEY_ESC && choice != KEY_BACKSPACE)
            choice = getch();

         if (choice == KEY_BACKSPACE)
         {
            back_step();
            continue;
         }
         break;
      }
      else if (check_win() == true)
      {
```

```
            draw_over(win_image);

            win = true;
            int choice = getch();
            while (choice != KEY_ENTER &&
                     choice != KEY_BACKSPACE &&
                     choice != KEY_ESC)
               choice = getch();

            if (choice == KEY_BACKSPACE)
               back_step();
            else if (choice == KEY_ESC)
               break;
         }
      }
      save_record();
   }

   void create_field()
   {
      Cells = new Cell*[field_size];
      play_field = new long long*[field_size];

      move_pos = new int*[field_size * field_size];
      last_field = new long long*[field_size];

      for (int i = 0; i < field_size; i++)
      {
         Cells[i] = new Cell[field_size];
         last_field[i] = new long long[field_size];
         play_field[i] = new long long[field_size];


         int offset = field_width / field_size / 2 +
                          (FIELD_WALL_WIDTH / 2);
         for (int j = 0; j < field_size; j++)
         {
            play_field[i][j] = 0;
            last_field[i][j] = play_field[i][j];
            Cells[i][j] =
            {play_field[i][j],
               X_OFFSET + j * (field_width / field_size) + offset,
               Y_OFFSET + i * (field_width / field_size) + offset,
               field_width / field_size / 2, 0, 0, 0,
               X_OFFSET + j * (field_width / field_size) + offset,
               Y_OFFSET + i * (field_width / field_size) + offset};
            }
      }

      for (int i = 0; i < field_size * field_size; i++)
      {
         move_pos[i] = new int[2];
```

```
      for (int j = 0; j < 2; j++)
         move_pos[i][j] = 0;
   }

   new_number();
   new_number();
}

void open_record()
{
   string path = "records/record_" +
      to_string(field_size) + ".txt";
   ifstream fin;
   fin.open(path);
   record = 0;
   if (!fin.eof())
      fin >> record;
   fin.close();
}

void save_record()
{
   string path = "records/record_" +
      to_string(field_size) + ".txt";
   ofstream fout;
   fout.open(path);
   fout << record;
   fout.close();
}

void delete_field()
{
   freeimage(win_image);
   freeimage(lose_image);

   for (int i = 0; i < field_size; i++)
   {
      delete[] last_field[i];
      delete[] Cells[i];
      delete[] play_field[i];
      delete[] move_pos[i];
   }
   delete[] move_pos;
   delete[] Cells;
   delete[] last_field;
   delete[] play_field;
}

void new_number()
{
   int i, j;
   int num;
```

```
    int chance = 10;

    i = rand() % field_size;
    j = rand() % field_size;

    while (play_field[i][j] != 0)
    {
        i = rand() % field_size;
        j = rand() % field_size;
    }

    num = rand() % chance;

    if (num == 0)
        play_field[i][j] = 4;
    else
        play_field[i][j] = 2;

    refresh_field();
    free_fields--;
}

void draw_field()
{
    putimage(0, 0, background_image, COPY_PUT);

    string score_txt = "Score: " + to_string(score);
    string record_txt = "Record: " + to_string(record);

    setusercharsize(6, 5, 4, 3);
    settextstyle(FONT_TEXT, HORIZ_DIR, USER_CHAR_SIZE);
    setbkcolor(imagegetpixel(background_image, 0, 0));
    setcolor(WHITE);

    int score_y = 80;
    int record_y = 120;

    settextjustify(LEFT_TEXT, BOTTOM_TEXT);
    outtextxy(X_OFFSET, score_y, score_txt.c_str());
    outtextxy(X_OFFSET, record_y, record_txt.c_str());

    setfillstyle(SOLID_FILL, COLOR(100, 100, 100));
    bar (X_OFFSET, Y_OFFSET,
        X_OFFSET + field_width + FIELD_WALL_WIDTH,
        Y_OFFSET + field_width + FIELD_WALL_WIDTH);

    setfillstyle(SOLID_FILL, COLOR(70, 70, 70));
    setcolor(WHITE);


    int field_down = 150 + field_width;
    for (int i = 0; i <= field_size; i++)
```

```
    {
        int offset = i * field_width / field_size;

        bar(X_OFFSET + offset, Y_OFFSET,
            X_OFFSET + offset + FIELD_WALL_WIDTH,
            field_down + FIELD_WALL_WIDTH);

        bar(X_OFFSET, Y_OFFSET + offset,
        WIDTH - X_OFFSET + FIELD_WALL_WIDTH,
        Y_OFFSET + offset + FIELD_WALL_WIDTH);
    }
}

void draw_numbers()
{
    settextjustify(CENTER_TEXT, BOTTOM_TEXT);
    for (int i = 0; i < field_size; i++)
        for (int j = 0; j < field_size; j++)
            if (Cells[i][j].data > 0)
            {
                string text = to_string(Cells[i][j].data);
                int Cell_background_color =
                COLOR(Cells[i][j].red,
                          Cells[i][j].green,
                          Cells[i][j].blue);

                setbkcolor(Cell_background_color);
                setfillstyle(SOLID_FILL, Cell_background_color);

                int coefficient = (text.length() + 1) * field_size;
                setusercharsize(31, coefficient + 2, 31, coefficient);
                settextstyle(FONT_TEXT, HORIZ_DIR, USER_CHAR_SIZE);

                int offset = Cells[i][j].size - FIELD_WALL_WIDTH / 2;

                bar (Cells[i][j].pos_x - offset,
                      Cells[i][j].pos_y - offset,
                      Cells[i][j].pos_x + offset,
                      Cells[i][j].pos_y + offset);

                outtextxy(Cells[i][j].pos_x,
                   Cells[i][j].pos_y + 0.5 * textheight(text.c_str()),
                   text.c_str());
            }
}

int find_num(int n)
{
    if (n < 1)
        return 0;

    int res = 0;
```

```
    while(n != 1)
    {
        n /= 2;
        res++;
    }
    return res;
}

void save_field()
{
    for (int i = 0; i < field_size; i++)
        for (int j = 0; j < field_size; j++)
            last_field[i][j] = play_field[i][j];

    last_score = score;
    last_free_fields = free_fields;
}

void back_step()
{
    for (int i = 0; i < field_size; i++)
        for (int j = 0; j < field_size; j++)
            play_field[i][j] = last_field[i][j];

    win = last_win;
    score = last_score;
    free_fields = last_free_fields;
    refresh_field();
}

bool can_move(int key)
{
    int j_start, step, end;

    if (key == KEY_UP || key == KEY_LEFT)
    {
        j_start = 0;
        step = 1;
        end = field_size;
    }
    else if (key == KEY_DOWN || key == KEY_RIGHT)
    {
        j_start = field_size - 1;
        step = -1;
        end = -1;
    }

    if (key == KEY_UP || key == KEY_DOWN)
    {
        for (int i = 0; i < field_size; i++)
            for (int j = j_start; j != end; j += step)
                if (play_field[j][i] == 0)
```

```
            {
               int k_start = j + step;
               for (int k = k_start; k != end; k += step)
                  if (play_field[k][i] != 0)
                     return true;
            }
   }
   else if (key == KEY_RIGHT || key == KEY_LEFT)
   {
      for (int i = 0; i < field_size; i++)
         for (int j = j_start; j != end; j += step)
            if (play_field[i][j] == 0)
            {
               int k_start = j + step;
               for (int k = k_start; k != end; k += step)
                  if (play_field[i][k] != 0)
                     return true;
            }
   }
   return false;
}

bool can_combine(int key)
{
   if (key == KEY_UP || key == KEY_DOWN)
   {
      for (int i = 0; i < field_size; i++)
         for (int j = 0; j < field_size - 1; j++)
            if (play_field[j][i] != 0 &&
            play_field[j][i] == play_field[j + 1][i])
               return true;
   }
   else if (key == KEY_RIGHT || key == KEY_LEFT)
   {
      for (int i = 0; i < field_size; i++)
         for (int j = 0; j < field_size - 1; j++)
            if (play_field[i][j] != 0 &&
            play_field[i][j] == play_field[i][j + 1])
               return true;
   }
   return false;
}

void move(int key)
{
   int j_start, end, step;
   int n_moved_Cells = 0;
   int x_speed = 0, y_speed = 0;

   switch (key)
   {
      case KEY_UP:
```

```
            j_start = 0;
            step = 1;
            end = field_size;
            y_speed = -speed;
            break;
        case KEY_DOWN:
            j_start = field_size - 1;
            step = -1;
            end = -1;
            y_speed = speed;
            break;
        case KEY_LEFT:
            j_start = 0;
            step = 1;
            end = field_size;
            x_speed = -speed;
            break;
        case KEY_RIGHT:
            j_start = field_size - 1;
            step = -1;
            end = -1;
            x_speed = speed;
            break;
    }

    if (key == KEY_RIGHT || key == KEY_LEFT)
    {
        for (int i = 0; i < field_size; i++)
            for (int j = j_start; j != end; j += step)
                for (int k = j + step; k != end; k += step)
                {

                    if ((play_field[i][j] == 0 &&
                            play_field[i][k] != 0) ||
                        (play_field[i][j] == play_field[i][k] &&
                            play_field[i][j] > 0))
                    {
                        move_pos[n_moved_Cells][0] = i;
                        move_pos[n_moved_Cells][1] = k;
                        n_moved_Cells++;
                    }

                    if (play_field[i][j] == 0 && play_field[i][k] != 0)
                    {
                        Cells[i][k].next_x = Cells[i][j].pos_x;
                        play_field[i][j] = play_field[i][k];
                        play_field[i][k] = 0;
                    }
                    else if (play_field[i][j] != 0 &&
                    play_field[i][j] == play_field[i][k])
                    {
                        Cells[i][k].next_x = Cells[i][j].next_x;
```

```
                play_field[i][j] *= 2;
                play_field[i][k] = 0;
                free_fields++;
                score += play_field[i][j];
                break;
             }
             else if (play_field[i][j] != 0 &&
             play_field[i][j] != play_field[i][k] &&
             play_field[i][k] != 0)
                break;
          }
   }
   else if (key == KEY_UP || key == KEY_DOWN)
   {
      for (int i = 0; i < field_size; i++)
         for (int j = j_start; j != end; j += step)
            for (int k = j + step; k != end; k += step)
            {
               if ((play_field[j][i] == 0 &&
                     play_field[k][i] != 0) ||
                  (play_field[j][i] == play_field[k][i] &&
                     play_field[j][i] > 0))
               {
                  move_pos[n_moved_Cells][0] = k;
                  move_pos[n_moved_Cells][1] = i;
                  n_moved_Cells++;
               }

               if (play_field[j][i] == 0 && play_field[k][i] != 0)
               {
                  Cells[k][i].next_y = Cells[j][i].pos_y;
                  play_field[j][i] = play_field[k][i];
                  play_field[k][i] = 0;
               }
               else if (play_field[j][i] != 0 &&
                        play_field[j][i] == play_field[k][i])
               {
                  Cells[k][i].next_y = Cells[j][i].pos_y;
                  play_field[j][i] *= 2;
                  play_field[k][i] = 0;
                  free_fields++;
                  score += play_field[j][i];
                  break;
               }
               else if (play_field[j][i] != 0 &&
                        play_field[j][i] != play_field[k][i] &&
                        play_field[k][i] != 0)
                  break;
            }
   }

   if (score > record)
```

```
      record = score;

   do_animation(x_speed, y_speed, n_moved_Cells);
}

void do_animation(int x_speed, int y_speed, int n_moved_Cells)
{
   bool moved;
   do
   {
      moved = false;
      for (int i = 0; i < n_moved_Cells; i++)
      {
         if (((x_speed >= 0 && y_speed >= 0) &&
            (Cells[move_pos[i][0]][move_pos[i][1]].next_x >
            Cells[move_pos[i][0]][move_pos[i][1]].pos_x ||
            Cells[move_pos[i][0]][move_pos[i][1]].next_y >
            Cells[move_pos[i][0]][move_pos[i][1]].pos_y))
            ||
            ((x_speed <= 0 && y_speed <= 0) &&
            (Cells[move_pos[i][0]][move_pos[i][1]].next_x <
            Cells[move_pos[i][0]][move_pos[i][1]].pos_x ||
            Cells[move_pos[i][0]][move_pos[i][1]].next_y <
            Cells[move_pos[i][0]][move_pos[i][1]].pos_y)))
         {
            Cells[move_pos[i][0]][move_pos[i][1]].pos_x += x_speed;
            Cells[move_pos[i][0]][move_pos[i][1]].pos_y += y_speed;
            moved = true;
         }
      }
      draw_field();
      draw_numbers();
      swapbuffers();
   }
   while (moved == true);

   refresh_field();
}

void refresh_field()
{
   int cell_width = field_width / field_size;
   for (int i = 0; i < field_size; i++)
      for (int j = 0; j < field_size; j++)
      {
         Cells[i][j].data = play_field[i][j];

         Cells[i][j].pos_x = X_OFFSET + j * cell_width +
               Cells[i][j].size + FIELD_WALL_WIDTH / 2;
         Cells[i][j].pos_y = Y_OFFSET + i * cell_width +
               Cells[i][j].size + FIELD_WALL_WIDTH / 2;
```

```
            Cells[i][j].next_x = Cells[i][j].pos_x;
            Cells[i][j].next_y = Cells[i][j].pos_y;

            Cells[i][j].red = 255 -
               (find_num(play_field[i][j]) * 15 + 50) % 255;
            Cells[i][j].green = 255 -
               (find_num(play_field[i][j]) * 10 + 50) % 255;
            Cells[i][j].blue = 255;
         }
}

bool check_win()
{
   if (win == true)
      return false;

   for (int i = 0; i < field_size; i++)
      for (int j = 0; j < field_size; j++)
         if (play_field[i][j] >= 2048)
            return true;
   return false;
}

bool check_lose()
{
   if (free_fields > 0)
      return false;

   for (int i = 0; i < field_size; i++)
      for (int j = 0; j < field_size - 1; j++)
         if (play_field[i][j] == play_field[i][j + 1])
            return false;

   for (int i = 0; i < field_size - 1; i++)
      for (int j = 0; j < field_size; j++)
         if (play_field[i][j] == play_field[i + 1][j])
            return false;

   return true;
}

void draw_over(IMAGE* image)
{
   int x_pos = 240, y_pos = 200;

   draw_field();
   draw_numbers();
   putimage(x_pos, y_pos, image, COPY_PUT);
   swapbuffers();
}
```

ПРИЛОЖЕНИЕ 2

Руководство пользователя

При запуске программы пользователь может пропустить заставку при нажатии любой клавиши клавиатуры. В меню игры пользователь может выбрать 4 пункта меню. При выборе пункта «Настройки» откроется окно выбором размерности игрового поля, по умолчанию установлена размерность 4. При выборе пункта «Об игре» откроется окно с краткой информацией о программе и информации о разработчике. При выборе пункта «Выход» программа завершит работу. Начать игру можно, выбрав пункт «Играть».

При старте игры в над игровым полем отображается текущее количество очков и рекорд по очкам набранный в выбранном режиме игры. Рекорд загружается и сохраняется в файл. Движения цифр происходит с помощью стрелок на клавиатуре. После каждого хода на случайной позиции поля появляется цифра. Для победы необходимо набрать 2048 на любой позиции игрового поля. В случае выигрыша можно продолжить игру.

При нажатии клавиши ESC можно вернуться из любого окна обратно в меню игры.

ПРИЛОЖЕНИЕ 3

Примеры выполнения программы

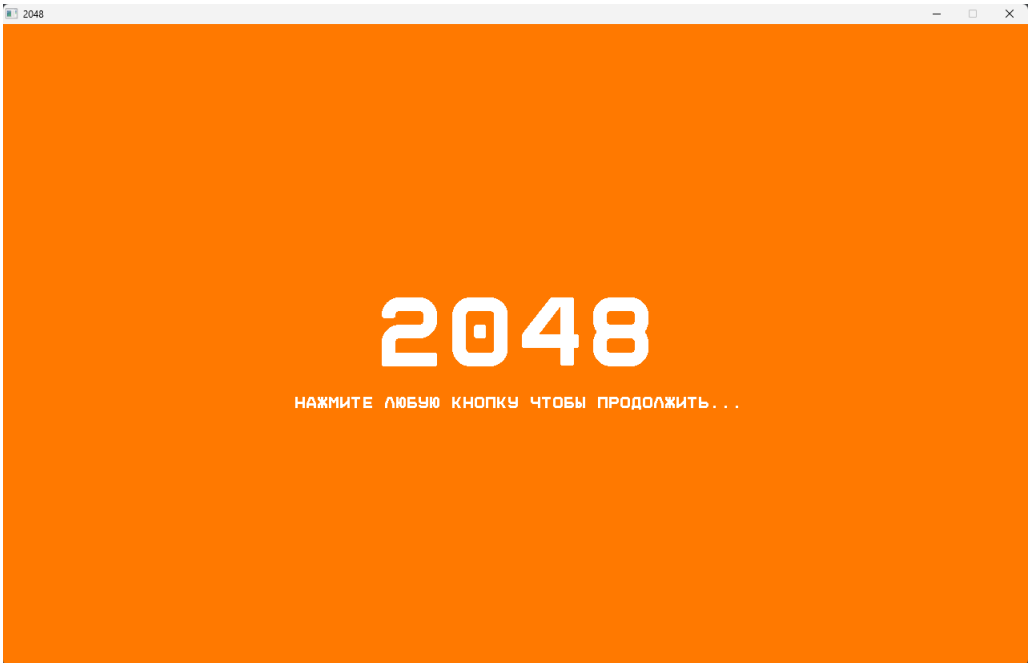

Рисунок П3.1 - Заставка игры

Рисунок П3.2 – Меню игры

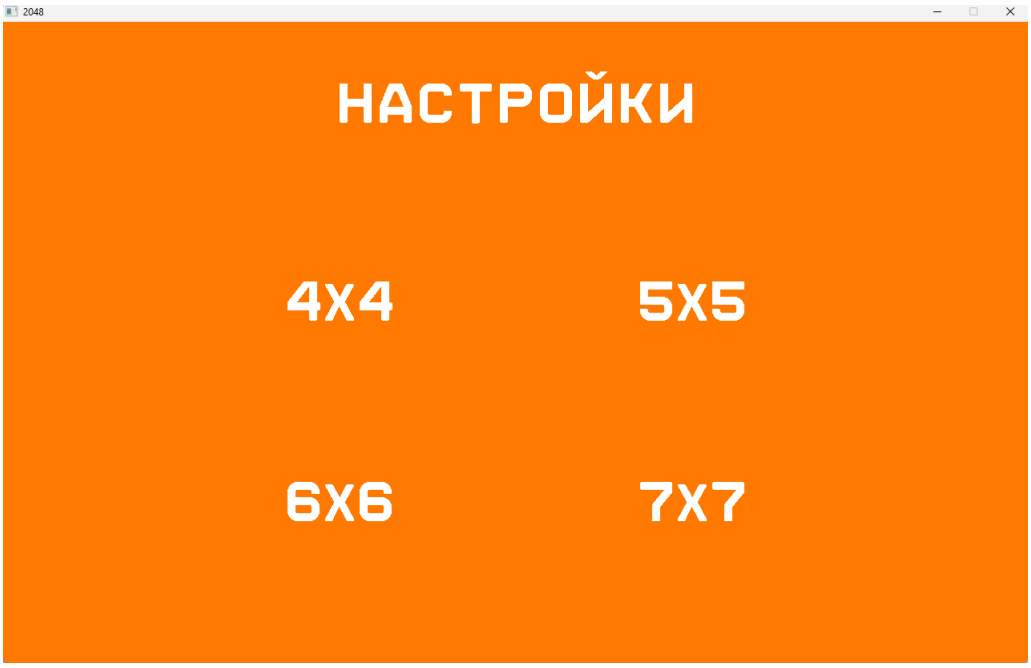

Рисунок П3.3 – Пункт меню «Настройки»

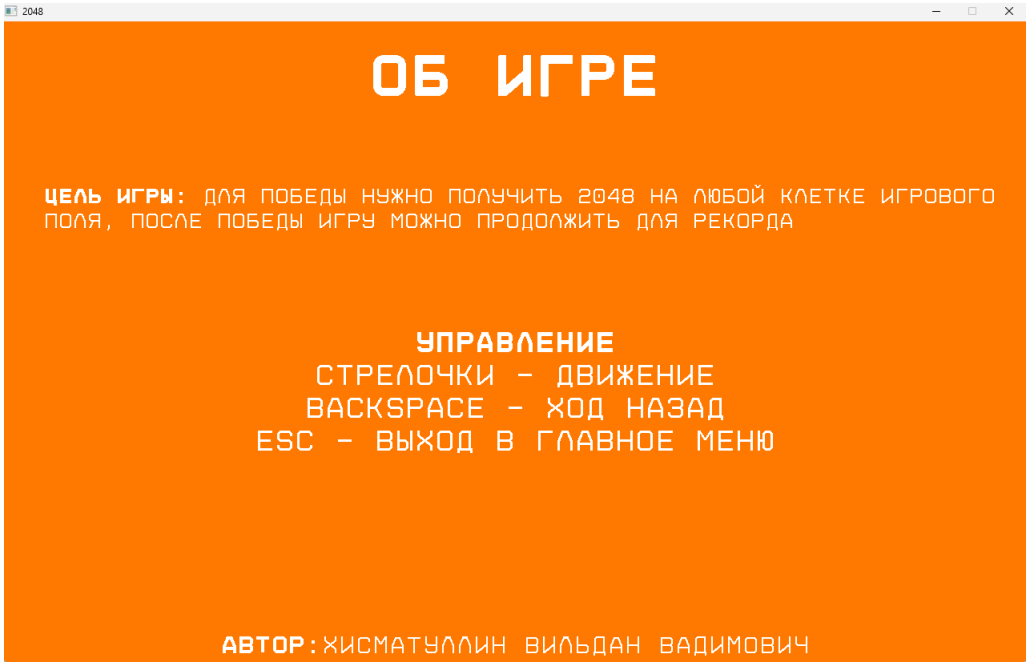

Рисунок П3.4 – Пункт меню «Об игре»

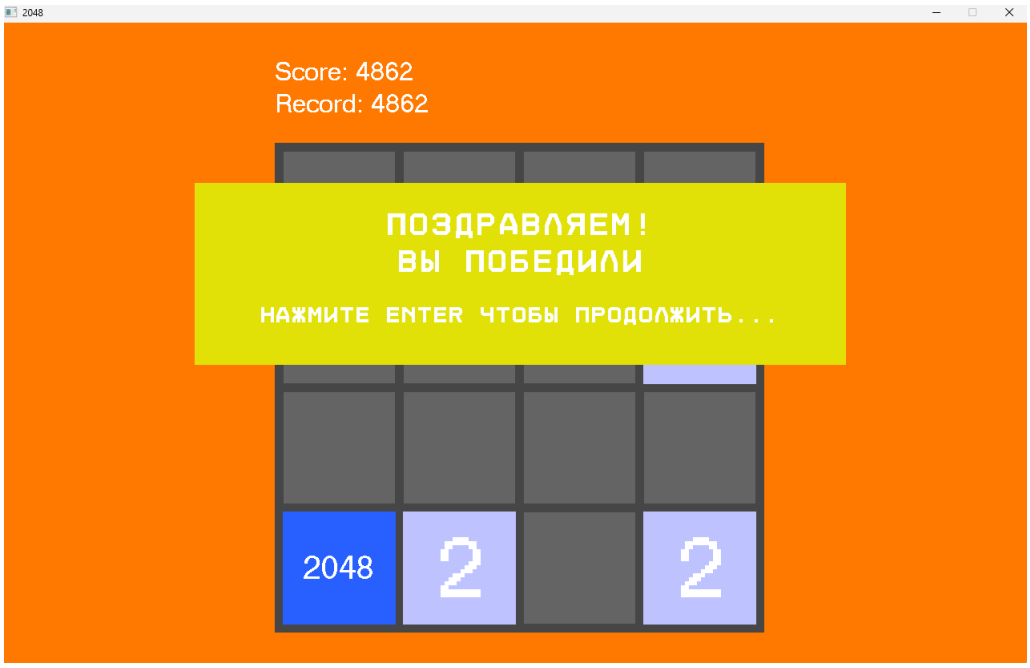

Рисунок П3.5 – Состояние игры в случае выигрыша

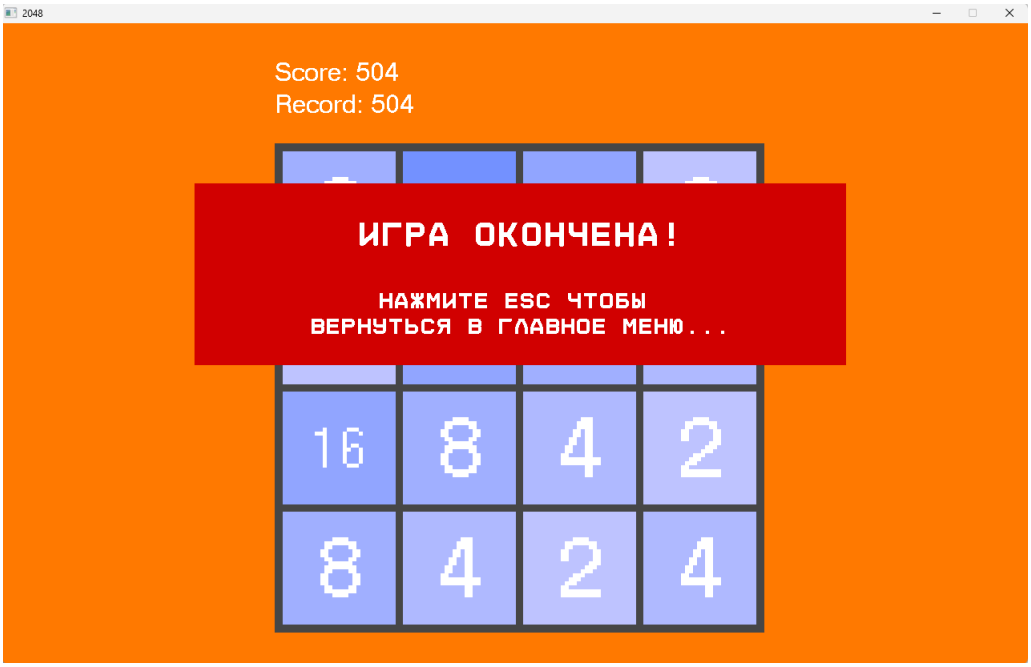

Рисунок П3.6 – Состояние игры в случае проигрыша