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 «Пазлы»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ
по дисциплине «Языки программирования»
ЮУрГУ–020301.2023.036.ПЗ КР

Автор работы,
студент группы ЕТ-111
_____ Сычев А.Д.
«____» _____ 2023 г.

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

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

Челябинск 2023

АННОТАЦИЯ

Сычев А.Д. Разработка игры «Пазлы». – Челябинск: ЮУрГУ, ЕТ-111, 2023. – 38 с., 11 ил., библиогр. список – 5 наим., 3 прил.

Целью работы является разработка компьютерной игры «Пазлы». 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 перемещения деталей пазла и алгоритм прикрепления детали к ячейке. 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 «Пазлы». 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 пазла, механизм управления отдельными деталями пазла;

– 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 «Пазлы».

После запуска программы появляется заставка игры. После нажатия любой клавиши на клавиатуре открывается главное меню программы, которое содержит пункты «Начать игру», «Покинуть игру», «Информация» и «Настройки» (рисунок 1.1). Выборы пункта меню осуществляются с помощью левой кнопки мыши.

При выборе пункта «Настройки» открывается окно с параметрами, в котором пользователь может выбрать язык(в игре представлено 2 языка русский и английский), а также может выбрать сложность, которая влияет на количество деталей пазла. (рисунок 1.2).

При выборе пункта «Информация» появляется окно с информацией обо мне и моем продукте. При выборе пункта «Выход» программа завершает работу.(рисунок 1.3).

При выборе пользователем пункта «Начать игру» появляется окно выбора уровня В этом окне есть 10 кнопок, каждая из которых позволяет вам выбрать 1 из 10 уровней.(рисунок 1.4). При нажатии на клавиатуре клавиши ESC вы выйдете в главное меню

После выбора уровня, появляется окно игры(рисунок 1.5). Окно игры делится на 2 части. В левой части находится игровое поле, в котором будет собираться пазл. В правой части снизу лежат детали пазла, а вверху представлен пример изображения для сбора.

После сбора итоговой картинки, появляется финальное окно с этим изображением(рисунок 1.6).

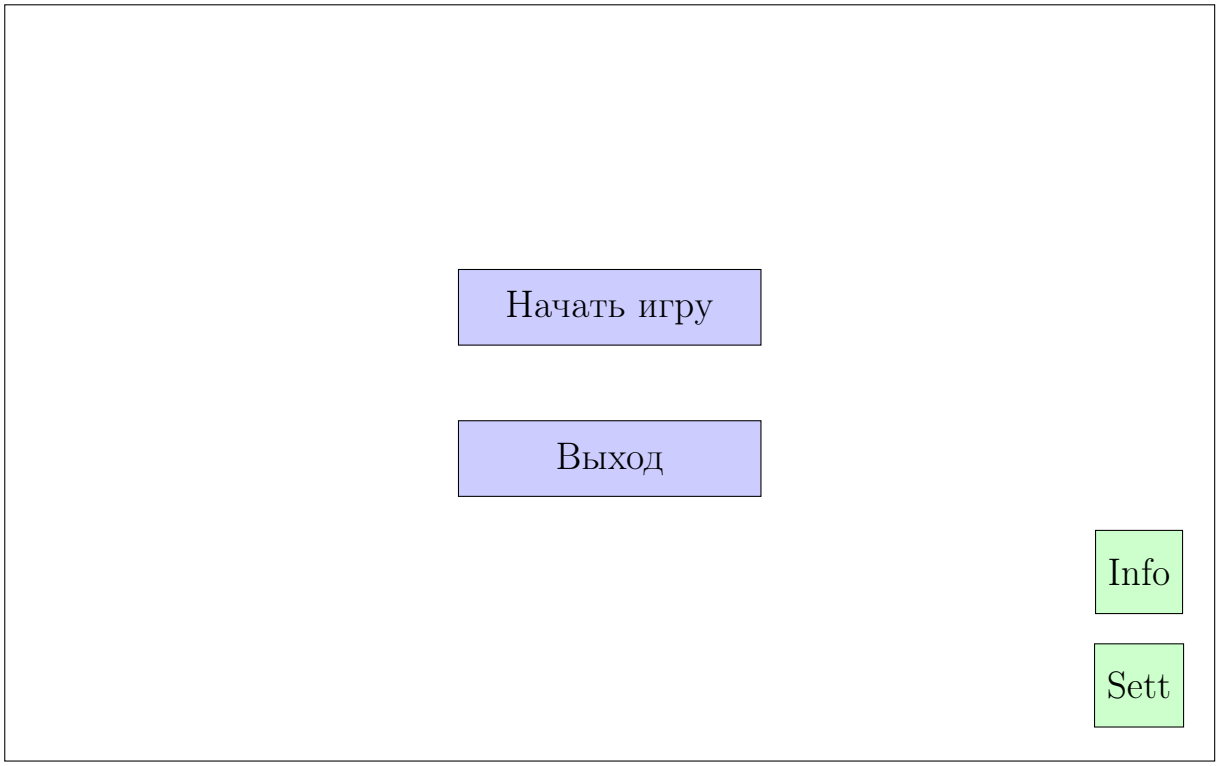

Рисунок 1.1 – Схема главного меню

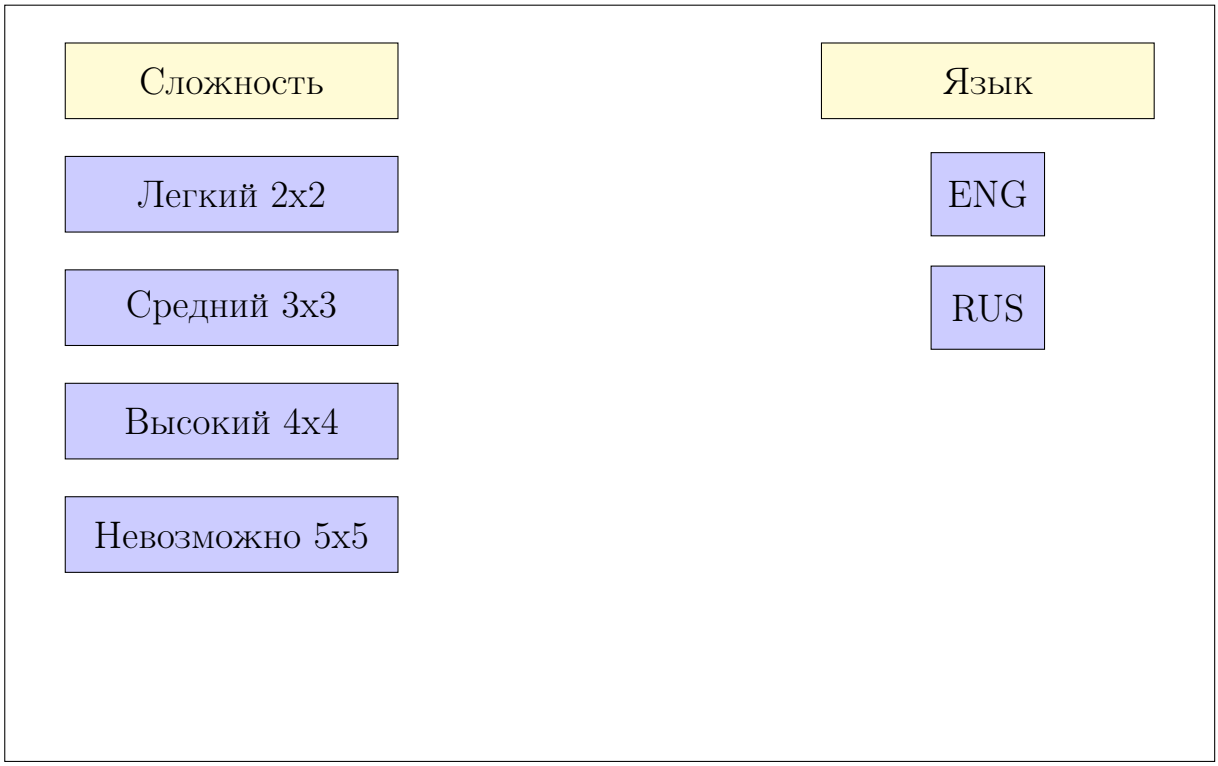

Рисунок 1.2 – Схема окна с настройками

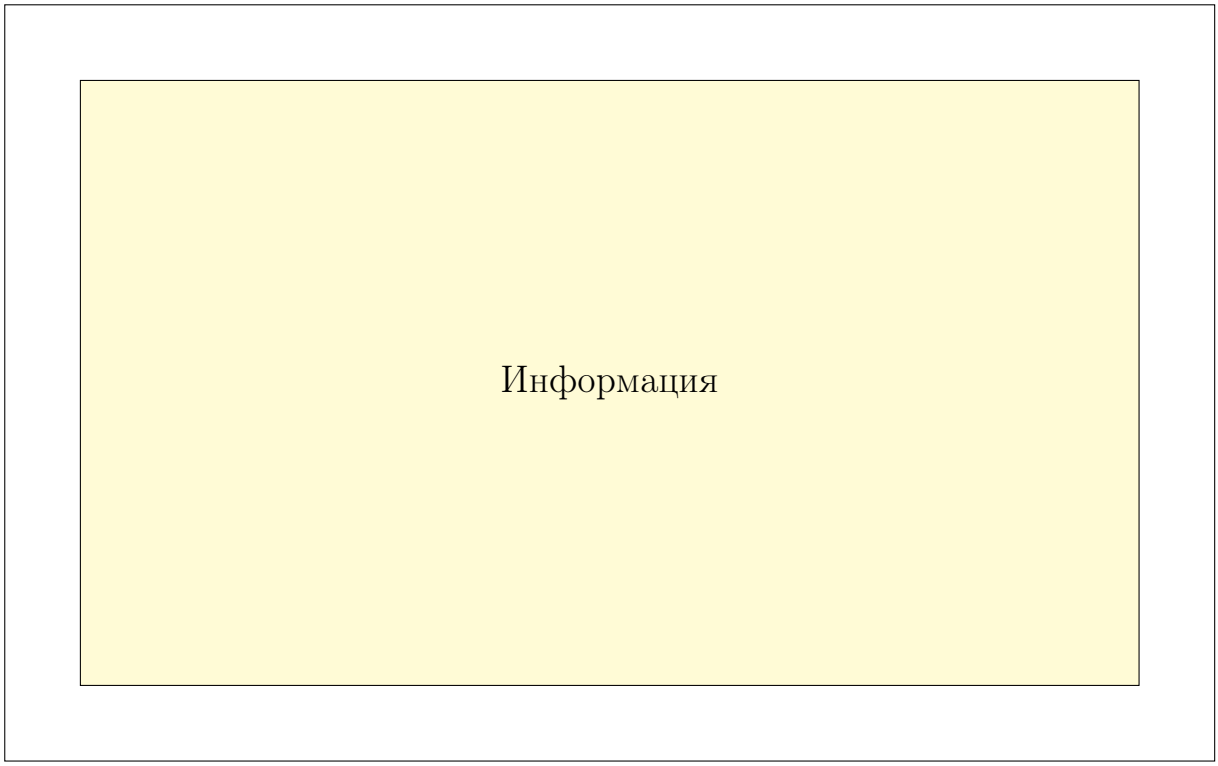

Рисунок 1.3 – Схема информационного окна

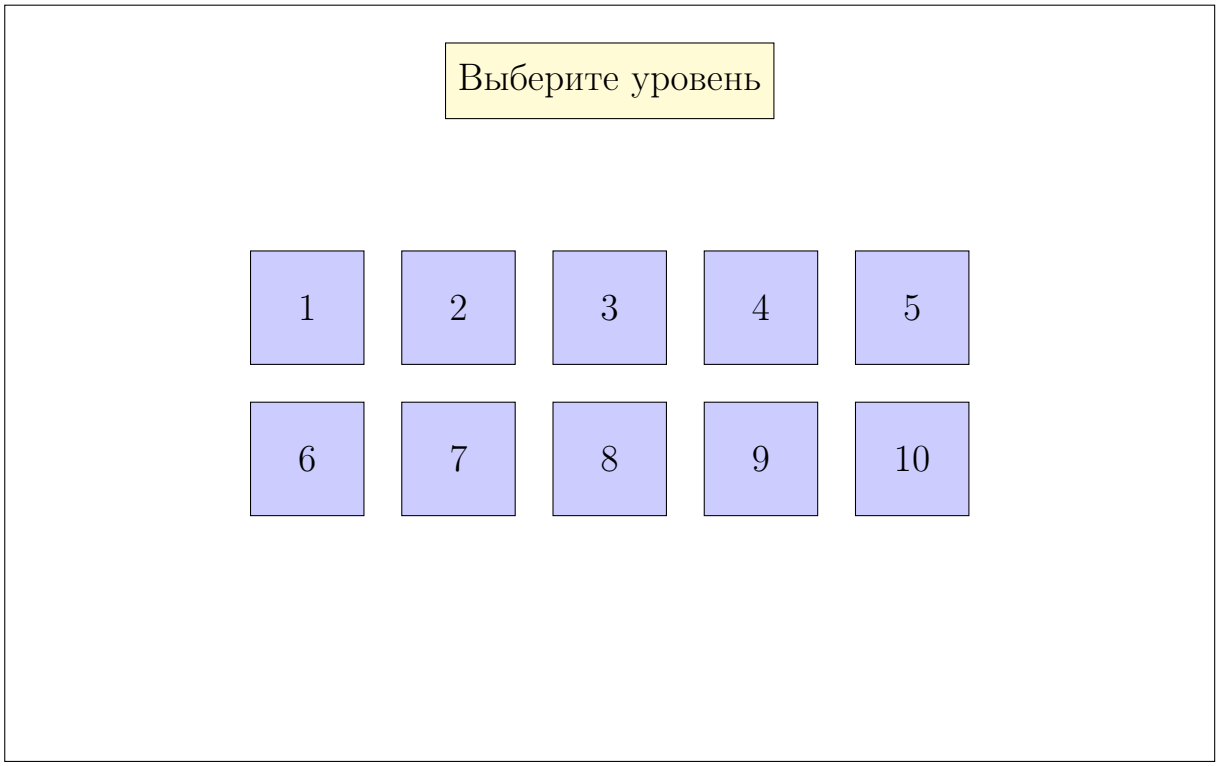

Рисунок 1.4 – Схема окна с выбором уровня

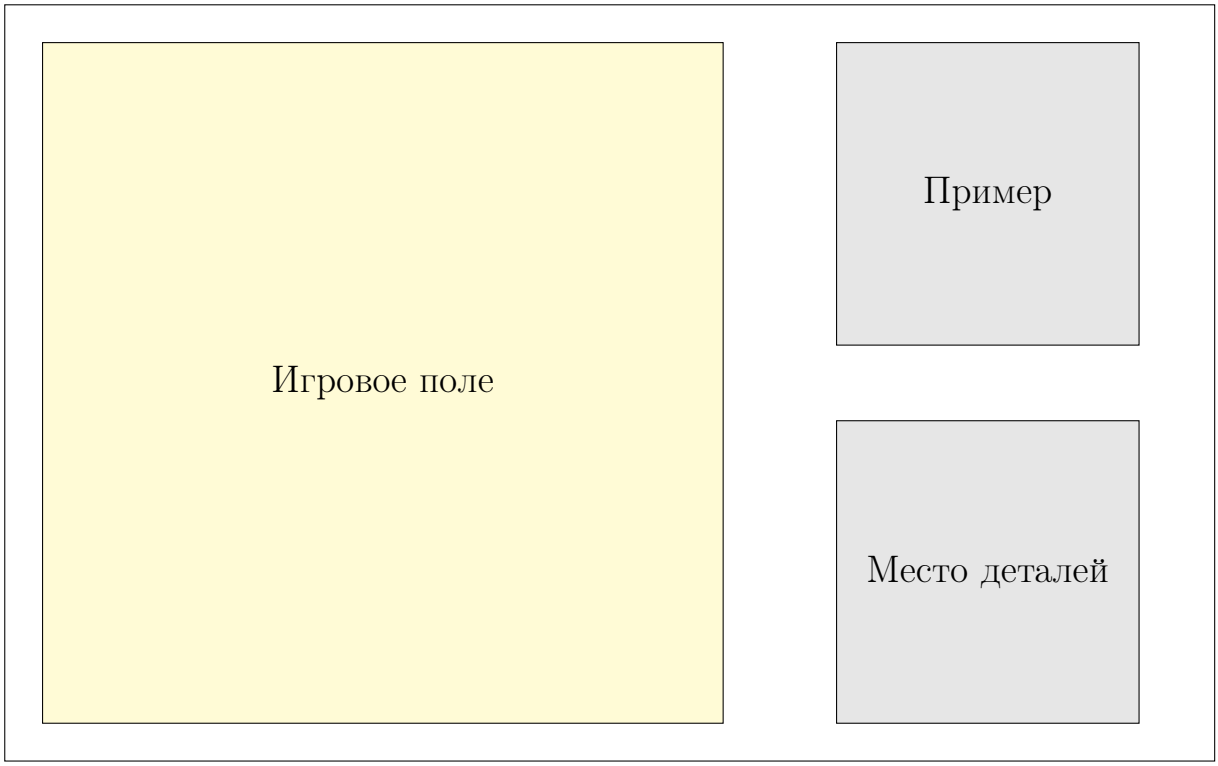

Рисунок 1.5 – Схема игрового окна

Рисунок 1.6 – Схема финального окна

2 ФОРМАЛИЗАЦИЯ ЗАДАЧИ

Игровое поле представляет собой область ограниченую координатами и разбитую на ячейки. Каждая ячейка имеет свой определенный номер, необходимый для сбора пазла в правильном порядке. Игровое поле:

$$F = \begin{pmatrix} 1 & 2 & \dots & n \\ n+1 & n+2 & \dots & n+n \\ \vdots & \vdots & \ddots & \vdots \\ n \times (n-1) & n \times (n-1)+1 & \dots & n \times n \end{pmatrix}$$

Пазл представляет собой квадратичную матрицу размерности n (по умолчанию равным 2). Однако, по желанию, пользователь может его задавать сам, в окне «Настройки».

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

Каждый элемент матрицы P определен структурой $Detail$. Она включает в себя, координаты детали, ее изображение, порядковый номер и логическую переменную положения.

Во время запуска уровня, создается массив P для нужного изображения и заданного размера n, который нам нужен для определения $cell\_size$ (размер одной ячейки). Он вычисляется по формуле:

$$cell\_size = \frac{FIELD\_SIZE}{n}$$

где $FIELD\_SIZE$ это размер игрового поля. После этого, все детали перемешиваются и выкладываются в правую часть окна.

Перемещение каждой детали пазла происходит путем подбирания ее ЛКМ. При отпускании ЛКМ, деталь остается в тех координатах, в которых ее отпустили. В случае, если координаты детали попадают в координаты ячейки, деталь автоматически прикрепляется к этой ячейке.

Если номер изображения и номер ячейки совпадут, то логическая переменная положения становится 1, а в противном случае она остается 0.

После всех активных действий происходит проверка на сбор итоговой картинки. Если окажется, что все логические переменные равны 1, то все детали пазла находятся в правильных ячейках, следовательно, картинка собрана правильно.

В таком случае, вызывается финальное, победное окно, а после нажатия на любую клавишу клавиатуры, нас переносит в главное меню.

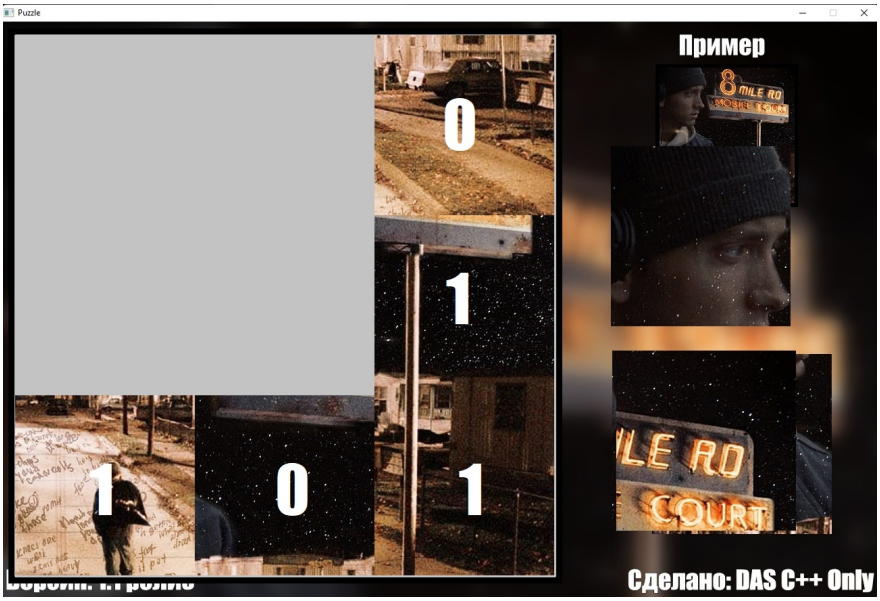

Рисунок 2.1 – Пример работы логической переменной

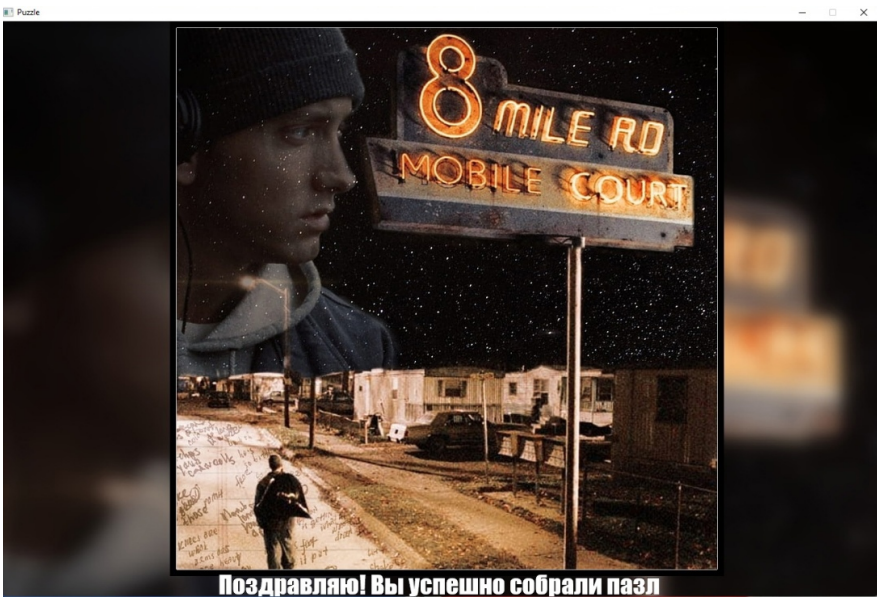

Рисунок 2.2 – Пример победного окна

3 РАЗРАБОТКА АЛГОРИТМА

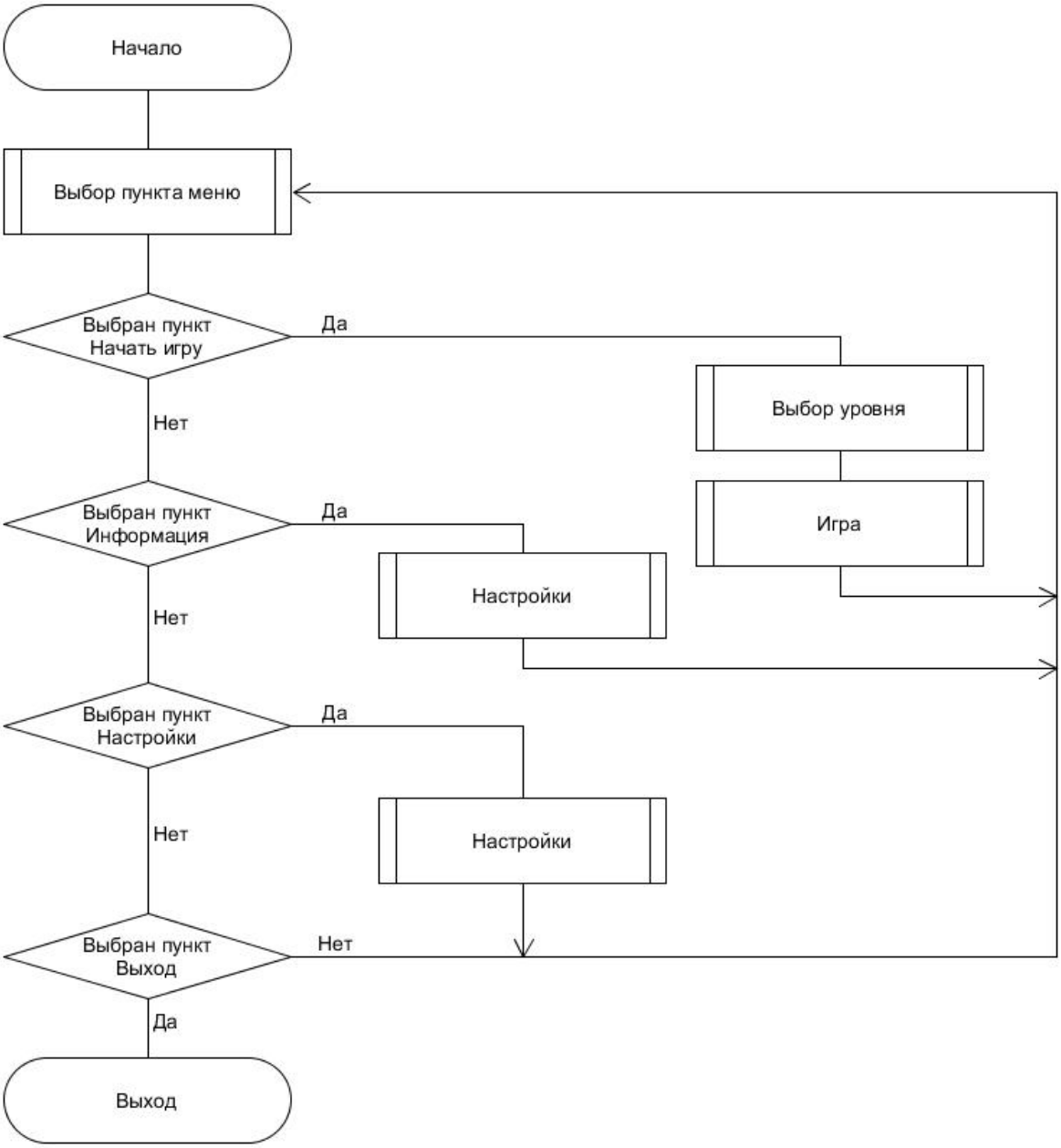

Рисунок 3.1 – Алгоритм работы программного меню

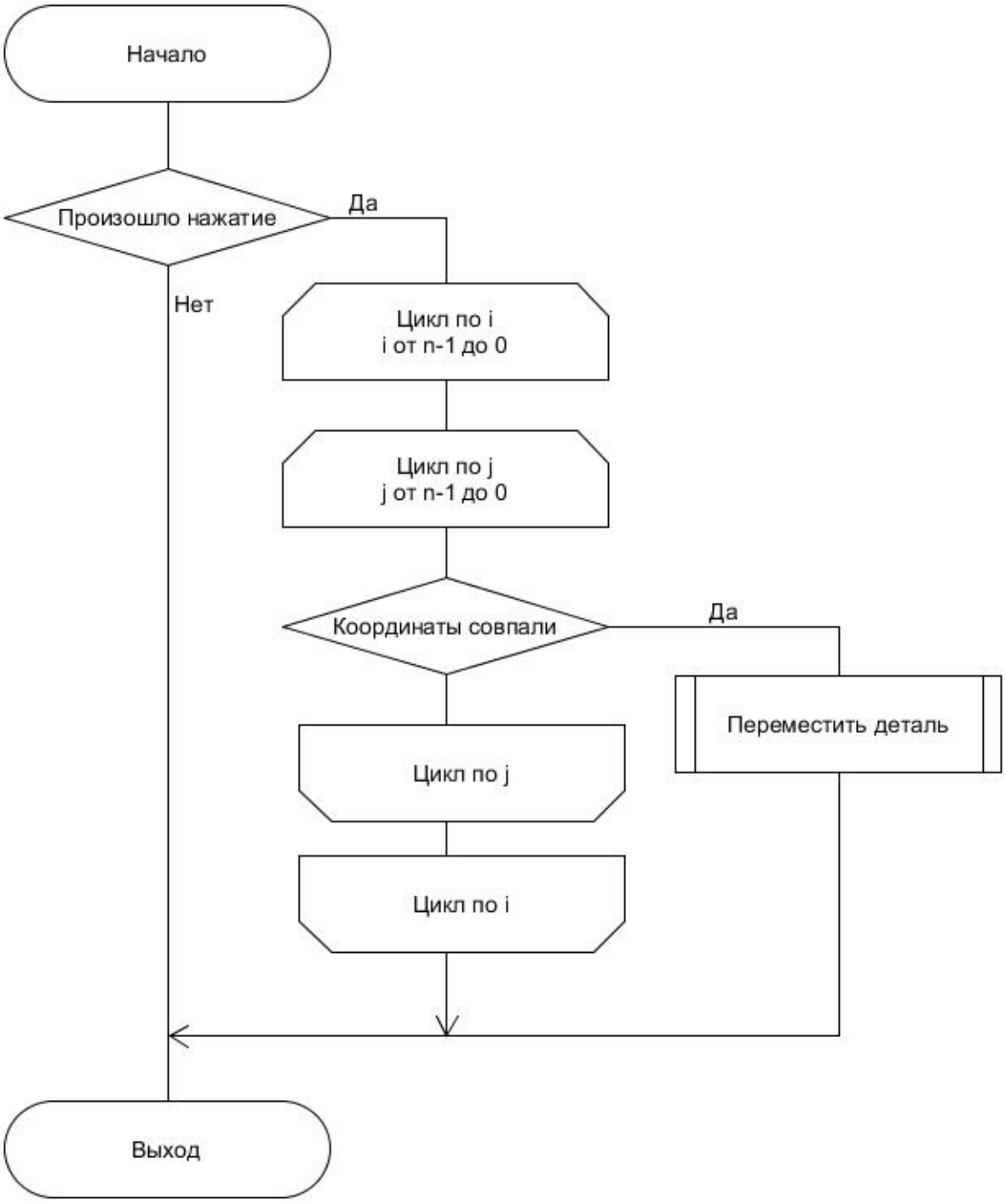

Рисунок 3.2 – Алгоритм перемещения деталей

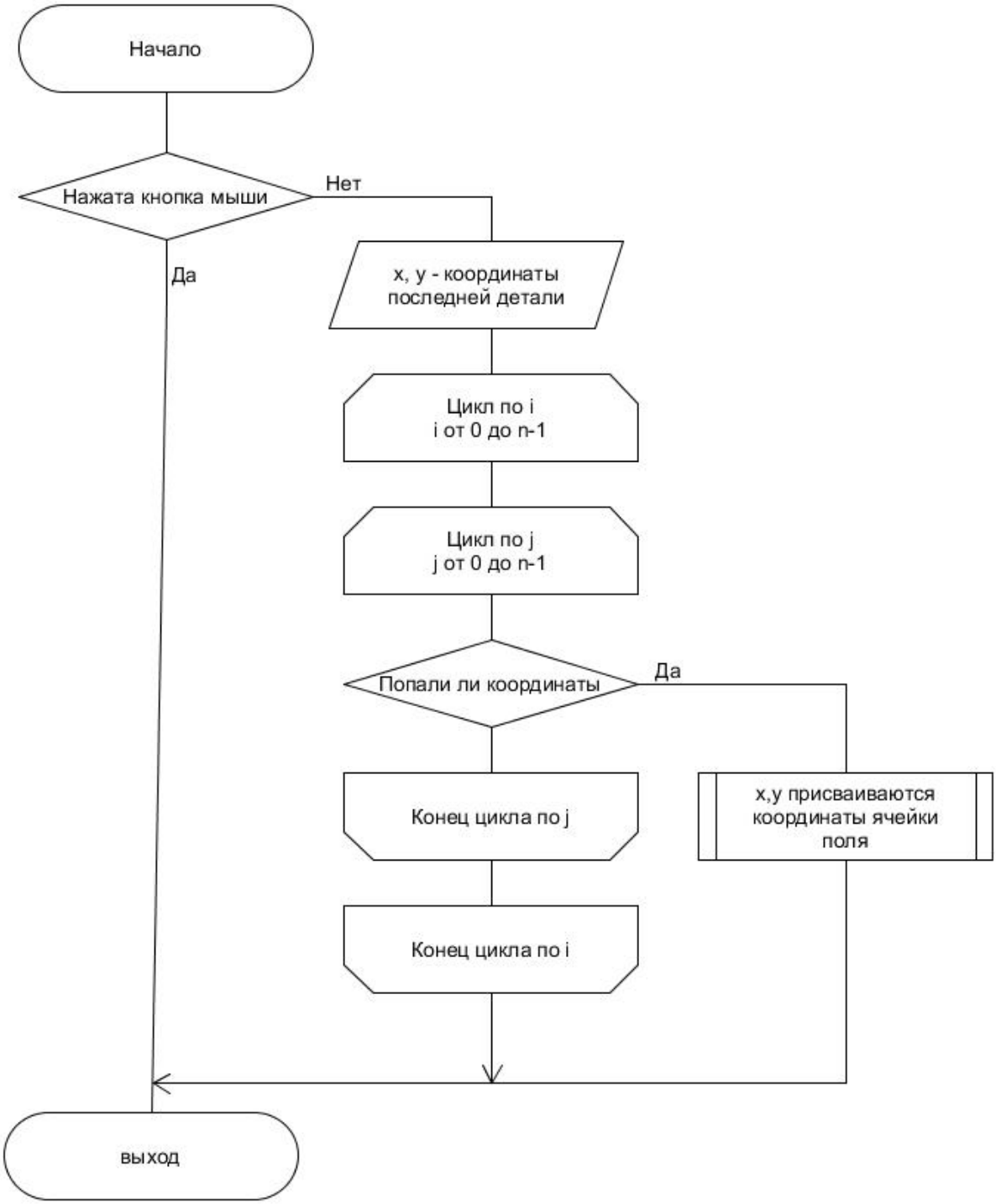

Рисунок 3.3 – Алгоритм примагничивания деталей

4 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

В реализации, программа разбита на 5 основных модулей `main.cpp`, `menu.cpp`, `settings.cpp`, `lvlchoose.cpp` и `game.cpp`, каждый из которых отвечает за определенную часть в программе.

Модуль `main.cpp` является начальным модулем в программе, именно в нем и происходит создание игрового окна, затем вызов функций для загрузки программы, а в конце, отчистка памяти от мусора.

Модуль `menu.cpp` это модуль, в котором находятся константы и функции для создания меню и обработки действий в нем. Для определения номера кнопок введено перечисление `GAME, EXIT, INFO, SETTINGS, N_BUTTONS`, которые нужны для описания состояние нажатия клавиш. Константа `COUNT_L` нужна для определения глобального `extern` массива `lvl_change`, который хранит в себе все текстовые изображения. Массив `button` хранит в себе кнопки, описанные структурой `Button`, Структура `Button` нужна для создания кнопок. Она хранит в себе `pos_x, pos_y` - координаты кнопки, `size_x, size_y` - размер кнопки, а `image` - изображение кнопки.

```
Button
{
    int pos_x;
    int pos_y;
    int size_x;
    int size_y;
    IMAGE *image;
};
```

Также, в модуле `menu.cpp` есть глобальные `extern` переменные, `*backg, *text, *quote`, которые используются во всех модулях, переменная `flag`, которая отвечает за выбранный язык и переменные `r_game, r_exit`, которые нужны для правильного отображения русских кнопок.

Функция `start()` рисует начальное изображение на экране. Функция `menu()` главная функция в модуле `menu.cpp`, она выполняет все нужные функции. Функция `load()` загружает все изображения, включая `extern`

переменные, которые описаны ранее. Функция `draw_backg()` рисует задний фон включая текст. Функция `define_array()` принимает на вход параметры координат, изображения и номера кнопки, для создания объекта структуры `Button`. Фактически, эта функция создает все кнопки с помощью констант. Функция `draw_menu()` рисует все кнопки в меню. Функция `choose_button()` принимает на вход количество кнопок и картинку. Она нужна для проверки нажатия на кнопку. В конце возвращает номер нажатой кнопки. Функция `draw_info()` нужна, что бы отрисовать информацию при выборе пункта «Информация». Функция `clear_image()` очищает все изображения включая `extern`.

В модуле `settings.cpp` описаны переменные, функции, которые нужны для отрисовки пункта «Настройки». Глобальные `extern` переменные `diff, size` влияют на переменные из модуля `game.cpp` и используются для установления размера пазла. Для определения нумерации кнопок введено перечисление `EASY, MEDIUM, HARD, IMPOSSIBLE, ENG, RUS, C_BUTTONS`, которые нужны для кнопок в модуле «Настройки». Глобальные переменные `sett, diffic, language, set_bag, pointer` типа IMAGE используются в модуле, для отрисовки изображений. Массив `difficult` размерности `C_BUTTONS` хранит в себе кнопки, описанные структурой `Button` из модуля `menu.cpp`

Функция `settings()` это основная функция из модуля `settings.cpp`, которая нужна для отрисовки настроек и всех кнопок. Функция `define_field()` нужна для определения всех кнопок в разделе «Настройки». Функция у `put_field()` зарисовывает меню «Настройки», включая кнопки. Функция `clear_image()` отчищает все изображения, которые используются в блоке «Настройки». Функция `choose_diff()` выбирает сложность игры, которая зависит от нажатой кнопки. Возвращает число, которое затем присваивается к переменной `diff`. Функция `choose_language()` выбирает язык игры, который влияет на логическую переменную `f` из модуля `menu.cpp`. Соответственно, 0 это английский язык, а 1 это русский язык.

Модуль `lvl_choose.cpp`, является промежуточным модулем программы и нужен для того, чтобы выбрать уровень игры. Определена кон-

станта MAX_LEVEL, которая определяет максимальный уровень и количество уровней. Также есть глобальная extern переменная, которая сохраняет в себя номер выбранного уровня и используется в модуле game.cpp. Каждая кнопка определена с помощью структуры Button из модуля menu.cpp, а потом помещена в массив levels, размерности MAX_LEVEL.

Функция lvl_choose() является основной функцией, которая нужна для отрисовки фона и кнопок для выбора уровня. Функция draw_levels() рисует кнопки для выбора уровня. Функция define_levels() использует переменную lvl, чтобы выбрать уровень. Функция clear_levels() отчищает все изображения, которые используются в модуле lvl_choose.cpp.

Модуль game.cpp это модуль, в котором создается сама игра. Константы FIELD_X, FIELD_Y это координаты начала игрового холста. Константа FIELD_SIZE это размер холста. Константы WORKPLACE_X, WORKPLACE_Y это координаты поля, куда выкладываются детали пазла при запуске программы. Константа MAX_SIZE максимальный уровень сложности, а также используется для создания глобальных, двумерных массивов array, в нем потом хранятся детали пазла и mask, где хранятся порядковые номера холста. Константа MAGIC_NUM коэффициент смещения, нужен для равномерного распределения деталей в поле, куда выкладываются детали. Глобальная extern переменнаяcomplete показывает, что игра закончена. Нужна для правильного построения игрового цикла.

В модуле game.cpp описана структура Detail, которая хранит в себе атрибуты pos_x, pos_y это координаты детали пазла, cur_num это номер текущей детали относительно собранной картинки, image это сама картинка пазла и логическая переменная move, которая нужна для проверки правильного положения детали.

```
Detail
{
   int pos_x;
   int pos_y;
   int cur_num;
   IMAGE *image;
   bool move;
```

```
};
```

Функция `game()` - основная функция, в которой происходит рисование полей, деталей и сама игра. Функция `load_images()` загружает изображения, которые нужны для конкретного уровня. Функция `put_secondary()` зарисовывает второстепенные изображения, например задний фон и текст. Функция `create_puzzle()` берет исходное изображение и разбивает его на детальки, размер которых зависит от переменной `diff` из модуля `settings.cpp`. Функция `mixing()` перемешивает детали пазла из массива `array`. Функция `put_details()` рисует детали пазла на экране. Функция `draw_field()` отрисовывает само игровое поле, размер которого также зависит от переменной `diff` из модуля `settings.cpp`. Функция `drag_detail()` отвечает за перемещение деталей по экрану программы. Функция `is_field()` проверяет, что при отпускании детали, она попадает в координаты ячейки игрового поля. Если это происходит, то проверяется номер пазла и номер ячейки, в случае их совпадения, логическая переменная пазла становится 1. Функция `is_end()` проверяет логические переменные `move` и если они все равны 1, то игра закончена. Функция `clear_all()` отчищает программу от картинок, в конце игры. Функция `draw_end()` рисует финальное, победное окно.

ЗАКЛЮЧЕНИЕ

Была разработана игра «Пазлы». Для этого был разработан алгоритм работы главного меню, для перехода по модулям программы. Так же, был разработан алгоритм разбиения картинки на детали фиксированного размера. Был разработан алгоритм перемещения деталей с помощью ЛКМ. Игровое поле и пазлы реализованы с помощью двух массивов, в которых каждый элемент имеет свой порядковый номер. Алгоритм проверки на сбор реализован путем сравнения этих порядковых номеров. Таким образом, все поставленные задачи были выполнены и цели курсовой работы достигнуты.

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 "settings.h"
#include "lvlchoose.h"
#include "graphics.h"

int main(){
   initwindow(1300, 850, "Puzzle");
   load();
   define_field();
   menu();
   clear_image();
   clear_levels();
}
```

Файл menu.h

```
#ifndef MENU_H
#define MENU_H

#define COUNT_L 8

#include "graphics.h"

struct Button{
   int pos_x;
   int pos_y;
   int size_x;
   int size_y;
   IMAGE *image;
   IMAGE *image2;
};

extern bool flag;


enum butt {GAME = 0, EXIT, INFO, SETTINGS, N_BUTTONS};

extern IMAGE *lvl_change[COUNT_L][2];
extern IMAGE *backg, *text, *quote;
extern Button button[N_BUTTONS];

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

```
void define_array(int pos, int x, int y, IMAGE *img, Button*);
void draw_menu();
int choose_button(const int, Button*);
void draw_info();
void clear_menu();

#endif
```

Файл menu.cpp

```
#include "settings.h"
#include "menu.h"
#include "lvlchoose.h"
#include "game.h"

#include "graphics.h"

IMAGE *backg, *quote, *info, *r_game, *r_exit;
IMAGE *lvl_change[COUNT_L][2];
Button button[N_BUTTONS];
bool f = 1;

void menu() {
   draw_backg();
   getch();
   while (true) {
      cleardevice();
      draw_menu();
      swapbuffers();
      if ((mousebuttons() == 1)) {
         int statement = choose_button(N_BUTTONS, button);
         switch (statement) {
         case GAME:
            lvl_choose();
            break;
         case EXIT:
            return;
         case INFO:
            draw_info();
            break;
         case SETTINGS:
            settings();
         }
      }
   }
}

void load() {

   backg = loadBMP("levels/backg.jpg");
   quote = loadBMP("levels/quote.bmp");
```

```
    info = loadBMP("levels/info.jpg");
    r_game = loadBMP("buttons/r_game.bmp");
    r_exit = loadBMP("buttons/r_exit.bmp");

    const char* languages[2] = {"eng", "rus"};
    const char* lvl_names[8] = {"settings", "difficulty", "language",
       "game", "exit", "example", "end"};
    for (int i = 0; i < 2; i++) {
       for (int j = 0; j < 8; j++) {
          char file_name[100];
          sprintf(file_name, "language/%s/%s.bmp",
          languages[i], lvl_names[j]);
          lvl_change[j][i] = loadBMP(file_name);
       }
    }

    define_array(GAME, 525, 300,
    loadBMP("buttons/game.bmp"), button);
    define_array(EXIT, 525, 500,
    loadBMP("buttons/exit.bmp"), button);
    define_array(INFO, 1200, 700,
    loadBMP("buttons/info.bmp"), button);
    define_array(SETTINGS, 1200, 600,
    loadBMP("buttons/settings.bmp"), button);
}

void define_array(int pos, int x, int y, IMAGE *img,
Button *array) {

    Button temp{x, y, imagewidth(img), imageheight(img), img};
    array[pos] = temp;
}

void draw_backg() {
    putimage(0, 0, backg, COPY_PUT);
    putimage(0, 800, lvl_change[3][f], TRANSPARENT_PUT);
    putimage(0, 0, quote, TRANSPARENT_PUT);
}

void draw_menu() {
    draw_backg();

    for (int i =0; i < N_BUTTONS; i++) {
       putimage(button[i].pos_x, button[i].pos_y, button[i].image,
       COPY_PUT);
    }
    if(f == 1){
        putimage(button[0].pos_x, button[0].pos_y, r_game,
       COPY_PUT);
       putimage(button[1].pos_x, button[1].pos_y, r_exit,
       COPY_PUT);
    }
```

```
}

void draw_info() {
   putimage(0, 0, info, COPY_PUT);
   putimage(0, 800, lvl_change[3][f], TRANSPARENT_PUT);
   putimage(0, 0, quote, TRANSPARENT_PUT);
   swapbuffers();
   getch();
}

int choose_button(const int N, Button *array) {
   int state = -1;
   if (mousebuttons() == 1) {
      int x = mousex();
      int y = mousey();


      for (int i = 0; i < N; i++) {
         if (x > array[i].pos_x &&
               x < array[i].pos_x + array[i].size_x &&
               y > array[i].pos_y &&
               y < array[i].pos_y + array[i].size_y)
         {
            state = i;
            break;
         }
      }
   }
   return state;
}

void clear_menu(){
   freeimage(r_exit);
   freeimage(r_game);
   freeimage(backg);
   freeimage(quote);
   freeimage(info);
   for(int i = 0; i < N_BUTTONS;i++)
      freeimage(button[i].image);
}
```

Файл settings.h

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

extern int diff, size;
enum difficult {EASY =0, MEDIUM, HARD,
 IMPOSSIBLE, ENG, RUS, C_BUTTONS};
```

```
void settings();
void define_field();
void put_field();
void clear_image();
int choose_diff();
void choose_language();

#endif
```

Файл settings.cpp

```
#include <iostream>
#include "settings.h"
#include "game.h"
#include "menu.h"
#include "graphics.h"

IMAGE *sett, *diffic, *language, *set_bag, *pointer;
int diff = 2;
int size = FIELD_SIZE / diff;
Button difficult[C_BUTTONS];

void settings() {
   while (true) {
      cleardevice();
      put_field();
      swapbuffers();
      diff = choose_diff();
      choose_language();
      size = FIELD_SIZE / diff;
      curr_part =
      imageresize(loadBMP("levels/curr_part.bmp"),
      size, size);
      if (kbhit() == 1)
         break;
   }
   getch();
}

void define_field() {
   set_bag = loadBMP("levels/settings.jpg");
   pointer = loadBMP("buttons/pointer.bmp");

   define_array(EASY, 185, 255,
   loadBMP("buttons/easy.bmp"),
   difficult);
   define_array(MEDIUM, 185, 345,
   loadBMP("buttons/medium.bmp"),
   difficult);
   define_array(HARD, 185, 435,
```

```
    loadBMP("buttons/hard.bmp"),
    difficult);
    define_array(IMPOSSIBLE, 185, 525,
    loadBMP("buttons/impossible.bmp"),
    difficult);
    define_array(ENG, 950, 260,
    loadBMP("language/eng/eng.bmp"),
    difficult);
    define_array(RUS, 950, 360,
    loadBMP("language/rus/rus.bmp"),
    difficult);
}

void put_field() {
    setfillstyle(SOLID_FILL, WHITE);
    setcolor(WHITE);

    putimage(0, 0, set_bag, COPY_PUT);
    putimage(0, 800, lvl_change[3][flag],
    TRANSPARENT_PUT);
    putimage(0, 0, quote,
    TRANSPARENT_PUT);
    putimage(425, 100, lvl_change[0][flag],
    TRANSPARENT_PUT);
    putimage(150, 200, lvl_change[1][flag],
    TRANSPARENT_PUT);
    putimage(850, 200, lvl_change[2][flag],
    TRANSPARENT_PUT);

    putimage(difficult[diff - 2].pos_x - 20,
    difficult[diff-2].pos_y, pointer,
    TRANSPARENT_PUT);

    if (flag == 0) {
        setfillstyle(SOLID_FILL, LIGHTGREEN);
        setcolor(LIGHTGREEN);
        bar(difficult[4].pos_x - 3, difficult[4].pos_y - 3,
            difficult[4].pos_x + difficult[4].size_x + 3,
        difficult[4].pos_y +  difficult[4].size_y + 3);

        setfillstyle(SOLID_FILL, WHITE);
        setcolor(WHITE);
        bar(difficult[5].pos_x - 3, difficult[5].pos_y - 3,
            difficult[5].pos_x + difficult[5].size_x + 3,
        difficult[5].pos_y +  difficult[5].size_y + 3);
    }
    else {
        setfillstyle(SOLID_FILL, LIGHTGREEN);
        setcolor(LIGHTGREEN);
        bar(difficult[5].pos_x - 3, difficult[5].pos_y - 3,
            difficult[5].pos_x + difficult[5].size_x + 3,
        difficult[5].pos_y +  difficult[5].size_y + 3);
```

```
      setfillstyle(SOLID_FILL, WHITE);
      setcolor(WHITE);
      bar(difficult[4].pos_x - 3, difficult[4].pos_y - 3,
          difficult[4].pos_x + difficult[4].size_x + 3,
      difficult[4].pos_y +  difficult[4].size_y + 3);
   }

   for (int i = 0; i < C_BUTTONS; i++) {
      putimage(difficult[i].pos_x, difficult[i].pos_y,
      difficult[i].image, COPY_PUT);
   }
}

void clear_image() {
   freeimage(sett);
   freeimage(diffic);
   freeimage(language);
   freeimage(set_bag);
}

int choose_diff() {
   int state = diff;
   if (mousebuttons() == 1) {
      int x = mousex();
      int y = mousey();

      for (int i = 0; i < C_BUTTONS - 2; i++) {
         if (x > difficult[i].pos_x &&
               x < difficult[i].pos_x +
         difficult[i].size_x &&
               y > difficult[i].pos_y &&
               y < difficult[i].pos_y +
         difficult[i].size_y)
         {
            state = i + 2;
            break;
         }
      }
   }
   return state;
}

void choose_language() {
   if (mousebuttons() == 1) {
      int x = mousex();
      int y = mousey();
      if (x > difficult[5].pos_x &&
           x < difficult[5].pos_x + difficult[5].size_x &&
           y > difficult[5].pos_y &&
           y < difficult[5].pos_y + difficult[5].size_y)
      {
```

```
            flag = 1;
        }
        else if (x > difficult[4].pos_x &&
             x < difficult[4].pos_x + difficult[4].size_x &&
             y > difficult[4].pos_y &&
             y < difficult[4].pos_y + difficult[4].size_y)
        {
            flag = 0;
        }
    }
}
```

Файл lvlchoose.h

```
#ifndef LVLCHOOSE_H
#define LVLCHOOSE_H

#include "menu.h"
#define MAX_LEVEL 10

extern int lvl;

void lvl_choose();
void draw_levels();
void define_levels();
void clear_levels();

#endif
```

Файл lvlchoose.cpp

```
#include <iostream>
#include "menu.h"
#include "game.h"
#include "lvlchoose.h"
#include "graphics.h"


Button levels[MAX_LEVEL];
int lvl = -1;

void lvl_choose(){
    delay(100);
    define_levels();
    while(lvl == -1){
        draw_levels();
        swapbuffers();
        lvl = choose_button(MAX_LEVEL, levels);
        if(kbhit() && getch() == KEY_ESC)
            return;
```

```
        if(lvl != -1){
            game();
            if(complete == true){
                complete = false;
                return;
            }
        }
    }
}

void draw_levels(){
    draw_backg();
    for(int i = 0;i < MAX_LEVEL;i++){
        putimage(levels[i].pos_x, levels[i].pos_y,
        levels[i].image, COPY_PUT);
    }
}


void define_levels(){
    for(int i = 0; i < MAX_LEVEL;i++){
        std::string path = "buttons/lvls/lvl" +
        std::to_string(i + 1) + ".bmp";
        define_array(i, 200 * ((i % 5) + 1), 200 * ((i / 5) + 1),
        imageresize(loadBMP(path.c_str()), 150, 150), levels);
    }
}

void clear_levels(){
    for(int i =0; i < MAX_LEVEL;i++){
        freeimage(levels[i].image);
    }
}
```

Файл game.h

```
#ifndef GAME_H
#define GAME_H

#define FIELD_X 20
#define FIELD_Y 20
#define FIELD_SIZE 800
#define WORKPLACE_X 890
#define WORKPLACE_Y 420
#define MAX_SIZE 5
#define MAGIC_NUM 25


#include "graphics.h"

struct Detail {
```

```
    int pos_x;
    int pos_y;
    int cur_num;
    IMAGE *image;
    bool move;
};


extern IMAGE *example, *end, *curr_part;
extern bool complete;

void game();
void load_images();
void put_secondary();
void create_puzzle();
void create_field();
void mixing();
void put_details();
void draw_field();
void drag_detail();
void is_field();
bool is_end();
void clear_all();
void draw_end();



#endif
```

Файл game.cpp

```
#include <iostream>
#include <ctime>
#include "game.h"
#include "menu.h"
#include "lvlchoose.h"
#include "settings.h"
#include "graphics.h"



IMAGE *background, *puzzle, *field,*curr_part;

Detail array[MAX_SIZE][MAX_SIZE];
int mask[MAX_SIZE][MAX_SIZE];
bool complete = false;


void game() {
    load_images();
    create_puzzle();
```

```
    mixing();
    while (true) {
        cleardevice();
        put_secondary();
        draw_field();
        put_details();
        if (mousebuttons() == 0) {
            is_field();
        }
        else if (mousebuttons() == 1) {
            drag_detail();
        }
        swapbuffers();
        if (is_end()) {
            draw_end();
            swapbuffers();
            complete = true;
            getch();
            break;
        }
        else if(kbhit() && getch() == KEY_ESC)
            break;
    }
    clear_all();
    lvl = -1;
}

void load_images() {
    background = loadBMP(("levels/lvl" +
    std::to_string(lvl+1) + "/backg.jpg").c_str());

    field = loadBMP("levels/field.jpg");

    puzzle = loadBMP(("levels/lvl" +
    std::to_string(lvl+1) +  "/puzzle.jpg").c_str());

    curr_part = imageresize(
    loadBMP("levels/curr_part.bmp"), size, size);
}

void put_secondary() {
    putimage(0, 0, background, COPY_PUT);
    setfillstyle(SOLID_FILL, BLACK);

    bar(968, 63, 1178, 273);
    putimage(974, 68, puzzle, COPY_PUT, 200, 200);

    putimage(968, 10, lvl_change[6][flag],
    TRANSPARENT_PUT);

    putimage(0, 800, lvl_change[3][flag],
    TRANSPARENT_PUT);
```

```
}


void create_puzzle() {
   srand(time(NULL));
   int cur_iter = 0;
   for (int i = 0; i < diff; i++)
      for (int j = 0; j < diff; j++) {
         int x = (rand() % (MAGIC_NUM * diff))
         +WORKPLACE_X;

         int y = (rand() % (MAGIC_NUM * diff))
         + WORKPLACE_Y;

         Detail temp{x, y, cur_iter,
            imagecopy(puzzle, i * size, j * size,
            size, size), 1};

         array[i][j] = temp;
         mask[i][j] = cur_iter++;
      }
}


void mixing() {
   for (int i = 0; i < diff; i++)
      for (int j = 0; j < diff; j++) {
         int rand_i = rand() % diff,rand_j = rand() % diff;
         Detail temp = array[i][j];
         array[i][j] = array[rand_i][rand_j];
         array[rand_i][rand_j] = temp;
      }
}


void put_details() {
   for (int i = 0; i < diff; i++)
      for (int j = 0; j < diff; j++)
         putimage(array[i][j].pos_x, array[i][j].pos_y,
   array[i][j].image, COPY_PUT);
}


void draw_field() {
   putimage(FIELD_X - 10, FIELD_Y - 10, field, COPY_PUT);
   for (int i = 0; i < diff; i++)
      for (int j = 0; j < diff; j++) {
         int m_x = mousex(), m_y = mousey();
         if (m_x > i * size && m_x < size * (i + 1)
            && m_y > j * size && m_y < size * (j + 1))
         {
            putimage(i * size + FIELD_X,j * size + FIELD_Y,
```

```
            curr_part, COPY_PUT);
         }
      }
}


void drag_detail() {
   for (int i = diff - 1; i >= 0; i--) {
      for (int j = diff - 1; j >= 0; j--) {
         int m_x = mousex();
         int m_y = mousey();
         if (m_x > array[i][j].pos_x && m_x
            < array[i][j].pos_x + size
               && m_y > array[i][j].pos_y && m_y
         < array[i][j].pos_y + size) {

            array[i][j].move = 1;
            Detail temp = array[i][j];
            array[i][j] = array[diff-1][diff-1];
            array[diff-1][diff-1] = temp;
            array[diff-1][diff-1].pos_x = m_x - size / 2;
            array[diff-1][diff-1].pos_y = m_y - size / 2;
            return;
         }
      }
   }
}


void is_field() {
   for (int i = 0; i < diff; i++) {
      for (int j = 0; j < diff; j++) {
         int temp_x = array[diff-1][diff-1].pos_x - size / 2;
         int temp_y = array[diff-1][diff-1].pos_y - size / 2;
         if (i * size + FIELD_X  > temp_x && i * size +
            FIELD_X <temp_x + size &&
            j * size + FIELD_Y > temp_y && j * size
         + FIELD_Y < temp_y + size) {

            array[diff-1][diff-1].pos_x = i * size + FIELD_X;
            array[diff-1][diff-1].pos_y =  j * size + FIELD_X;
            if (mask[i][j] == array[diff-1][diff-1].cur_num) {
               array[diff-1][diff-1].move = 0;
            }
            return;
         }
      }
   }
}

bool is_end() {
   for (int i = 0; i < diff; i++)
```

```
      for (int j = 0; j < diff; j++)
         if (array[i][j].move == 1)
            return 0;
   return 1;
}

void clear_all() {
   freeimage(background);
   freeimage(puzzle);
   freeimage(field);
   freeimage(curr_part);
   for (int i = 0; i < diff; i++)
      for (int j = 0; j < diff; j++)
         freeimage(array[i][j].image);
}

void draw_end() {
   putimage(0, 0, background, COPY_PUT);
   putimage(250, 0, field, COPY_PUT);
   putimage(260, 10, puzzle, COPY_PUT);
   putimage(250, 810,  lvl_change[7][flag],
   TRANSPARENT_PUT);
}
```

ПРИЛОЖЕНИЕ 2

Руководство пользователя

После запуска программы открывается заставка, которую можно пропустить нажатием на любую кнопку клавиатуры. После заставки появляется само меню программы, в котором можно выбрать 4 пункта. В пункте «Настройки», можно выбрать сложность пазла и язык программы(доступно 2 языка, русский и английский. По умолчанию русский). В пункте «Информация», вы можете прочитать информацию обо мне и моем продукте, а так же информацию об авторских правах на программу. При нажатии пункта «Выход» осуществляется выход из программы.

При выборе пункта «Старт», открывается новое окно, в котором необходимо выбрать уровень. Всего в программе представлено 10 уровней, каждый из которых отличается только изображением.

При запуске уровня, в левой части экрана появится холст для сбора пазла, в правой части будет пример сбора и детали пазла. Конечной целью игры является сбор пазла. Для этого необходимо переместить все детали пазла в правильные позиции на холсте. Перемещение происходит путем зажатия ЛКМ на детали и ее последующем ведении в нужную позицию. В том месте холста, где загорается яркий квадрат, можно отпустить деталь и она автоматически встанет в эту позицию.

Если был выбран не тот уровень, то всегда можно выйти из пазла с помощью клавиши ESC, в таком случае заново появится возможность выбрать уровень.

В случае правильного сбора пазла, откроется финальная картинка и поздравление. После нажатия на любую клавишу на клавиатуре, откроется главное меню.

ПРИЛОЖЕНИЕ 3

Примеры выполнения программы

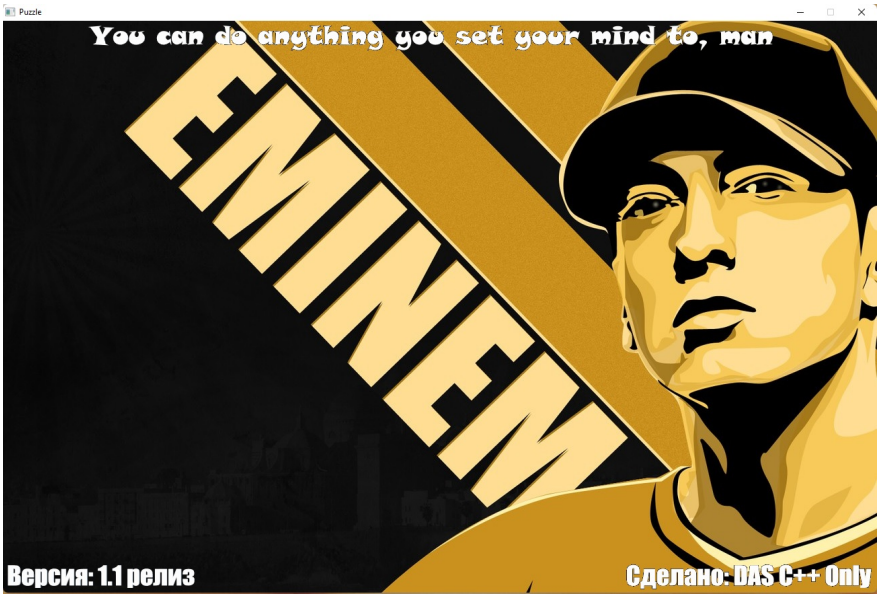

Рисунок П3.1 - Экран загрузки

Рисунок П3.2 - Главное меню

Рисунок П3.3 - Пункт «Информация»

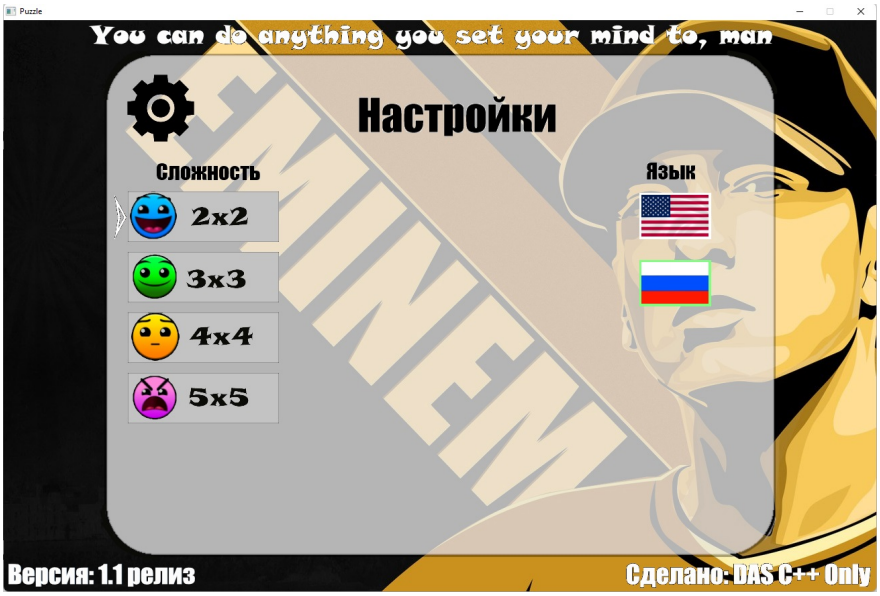

Рисунок П3.4 - Пункт «Настройки»

Рисунок П3.5 - Пункт «Выбор уровня»

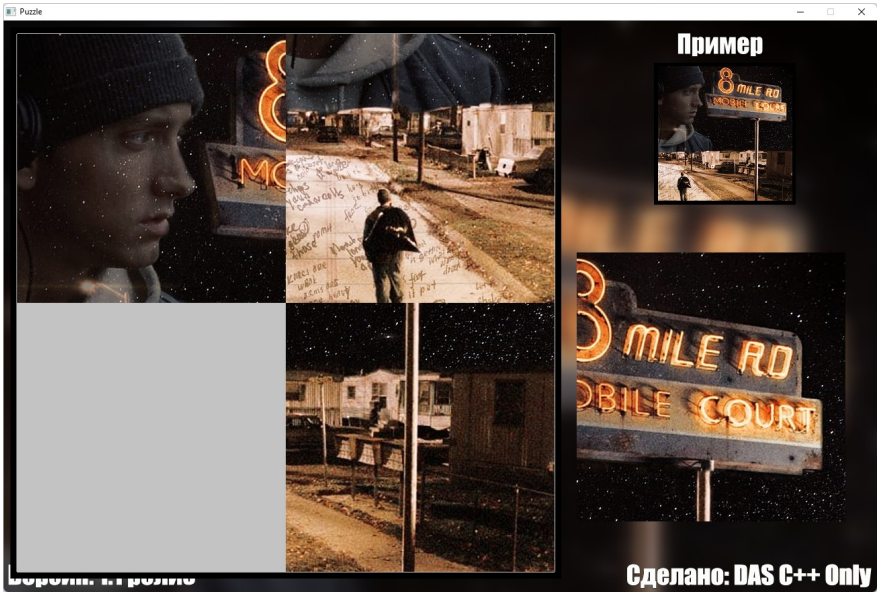

Рисунок П3.6 - Прохождение уровня

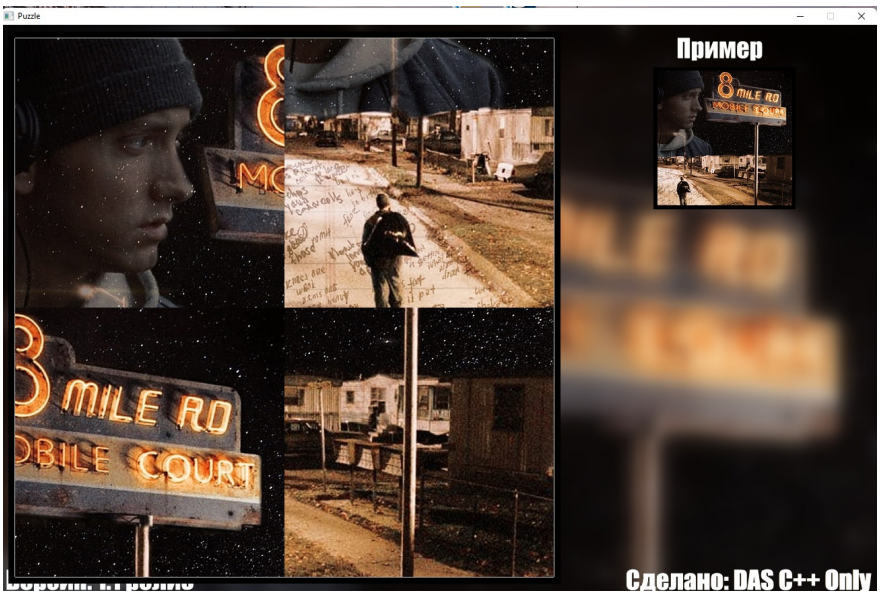

Рисунок П3.7 - Пример неправильного сбора

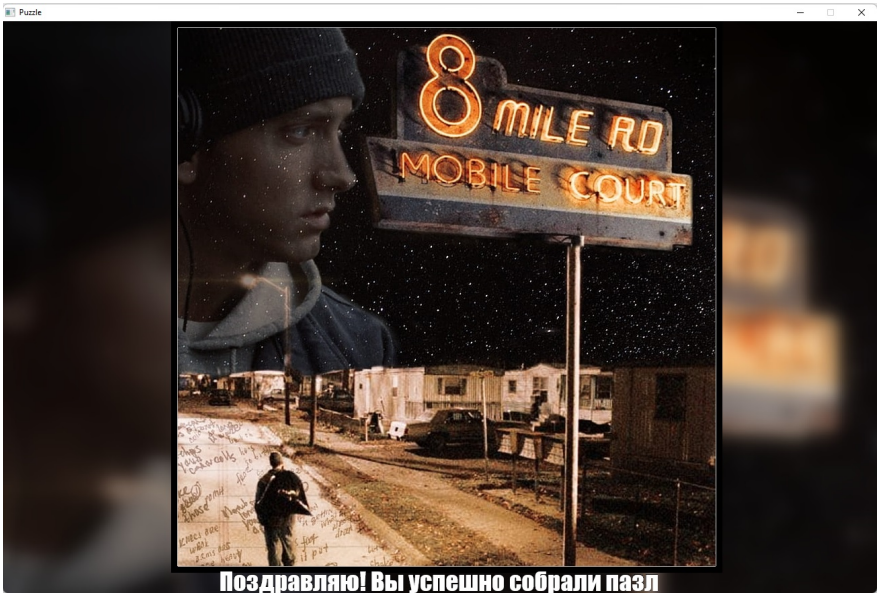

Рисунок П3.8 - Уровень пройден