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

ОТЧЕТ

о выполнении лабораторной работы № 6 по дисциплине
«Математические основы компьютерной графики»

Автор работы,
студент группы ЕТ-212
_____ Мухутдинов Б.А.
«____» _____ 2022 г.

Руководитель работы,
старший преподаватель
_____ Шелудько А.С.
«____» _____ 2022 г.

Челябинск 2022

1 ЗАДАНИЕ

Написать программу для выполнения аффинных преобразований многоугольника на плоскости. Предварительно определить структуру данных (класс) и разработать соответствующие подпрограммы (методы). Число и координаты вершин многоугольника считать из файла. Интерфейс программы должен содержать следующие элементы управления:

- перемещение фигуры;
- поворот фигуры (относительно центра фигуры);
- растяжение/сжатие фигуры;
- сохранение результата в файл;
- выход из программы.

2 ОПИСАНИЕ КЛАССА FIGURE

Есть структура coords для хранения координат. В ней находятся следующие объекты:

- int x;
- int y.

В классе Figure находятся объекты:

- coords center;
- coords *external;
- int r;
- int n;
- int rot=0.

В center хранится координаты центра фигуры, в массиве структур external хранится координаты вершин фигуры, в r радиус фигуры, в n количество вершин и в rot хранится как повернута фигура.

Методы класса:

- Figure(int rad, int N);
- set_external(int, coord);
- calc_center();
- void remove();
- void draw();
- void move(int direction);
- void rotate(int direction, int type);
- void change_size(int direction);
- void help(int type);
- void error();
- bool in_field(int dx);
- void calc_external_coords().

Figure(int rad, int N) - конструктор. remove() - для удаления фигуры с экрана. draw() - для рисования фигуры. move(int direction) - движение фигуры в определенном направлении. rotate(int direction, int type) - поворот фигуры влево или вправо. change_size(int direction) - для изменения размера фигуры. help(int type) - для вывода инструкции. error() - для вывода ошибки в случае выхода за границы экрана. bool in_field(int dx) - для проверки нахождения фигуры внутри экрана. void calc_external_coords() - приватная функция для вычисления вершин фигуры(в случае изменения размера). Сеттер set_external для получения координат вершин из файла. calc_center() - для вычисления координат середины фигуры.

Реализация в task.cpp

Файл main.cpp

```
#include <iostream>
#include <graphics.h>
#include <task.hpp>
#include <math.h>
#include <control.h>
#include <fstream>

using namespace std;

int main(){
   int type = 0, radius, n;//type: 0- , 1-move, 2-rotate, 3-resize

   ifstream file;
   file.open("info_for_figure.txt");
   file >> n;
   coords temp[n];
   for(int i=0; i<n; i++){
      file >> temp[i].x >> temp[i].y;
   }
   Figure a = Figure(fabs(temp[0].x-temp[5].x)/2, n);
   for(int i=0; i<n; i++){
      a.set_external(i, temp[i]);
   }
   a.calc_center();
   initwindow(WIDTH, HEIGHT, "well...");
   create_bg("bg.jpg");
   create_button(MOVE, 657, 378);
   create_button(ROTATE, 657, 463);
   create_button(RESIZE, 657, 547);
   create_button(SAVE, 657, 632);
   create_button(EXIT, 657, 718);
   setbkcolor(COLOR(235, 245, 238));
   setcolor(COLOR(70, 35, 122));
   setfillstyle(SOLID_FILL, COLOR(61, 220, 151));
   a.draw();
   a.help(0);
   while(1){
      while(mousebuttons() != 1){
         switch(type){
            case 1:
               switch(getch(kbhit())){
                  case KEY_UP:
                     a.move(0);
                     break;
                  case KEY_DOWN:
                     a.move(1);
                     break;
                  case KEY_LEFT:
```

```
                    a.move(2);
                    break;
                case KEY_RIGHT:
                    a.move(3);
                    break;
            }
            break;
        case 2:
            switch(getch(kbhit())){
                case KEY_LEFT:
                    a.rotate(0, 0);
                    break;
                case KEY_RIGHT:
                    a.rotate(1, 0);
                    break;
            }
            break;
        case 3:
            switch(getch(kbhit())){
                case KEY_LEFT:
                    a.change_size(0);
                    break;
                case KEY_RIGHT:
                    a.change_size(1);
                    break;
            }
            break;
    }
  }
  switch(select_control()){
    case MOVE:
      a.help(1);
      type = 1;
      break;
    case ROTATE:
      a.help(2);
      type = 2;
      break;
    case RESIZE:
      a.help(3);
      type = 3;
      break;
    case SAVE:
      save();
      break;
    case EXIT:
      return 0;
  }
 }
}
```

Файл task.hpp

```
#ifndef TASK_HPP
#define TASK_HPP
#define HEIGHT 800
#define WIDTH 1000

struct coords{
   int x;
   int y;
};

class Figure{
   private:
      coords center;
      coords *external;
      int r;
      int n;
      int rot=0;
      void calc_external_coords();
   public:
      Figure(int rad, int N);
      void set_external(int, coords);
      void calc_center();
      void remove();
      void draw();
      void move(int direction);
      void rotate(int direction, int type);
      void change_size(int direction);
      void help(int type);
      void error();
      bool in_field(int dx);
};

void save();

#endif
```

Файл task.cpp

```
#define _USE_MATH_DEFINES
#include <graphics.h>
#include <task.hpp>
#include <math.h>

Figure::Figure(int rad, int N){
   r = rad;
   n = N;
   external = new coords[n];
}
```

```
void Figure::set_external(int i, coords temp){
   external[i].x = temp.x;
   external[i].y = temp.y;
}

void Figure::calc_external_coords(){
   double theta = 0.0;
   for(int i = 0; i<n; i++){
      external[i].x = center.x + r*cos(theta);
      external[i].y = center.y + r*sin(theta);
      theta+=M_PI/3;
   }
}

void Figure::calc_center(){
   int temp_x = 0, temp_y = 0;
   for(int i=0; i<n; i++){
      temp_x += external[i].x;
      temp_y += external[i].y;
   }
   center.x = temp_x/n;
   center.y = temp_y/n;
}

void Figure::draw(){
   for(int i = 0; i<n; i++){
      line(external[i].x, external[i].y, external[(i+1)%n].x, externa
   }
}

void Figure::remove(){
   setcolor(COLOR(235, 245, 238));
   for(int i = 0; i<n; i++){
      line(external[i].x, external[i].y, external[(i+1)%n].x, externa
   }
   setcolor(COLOR(70, 35, 122));
}

void Figure::move(int direction){//direction: 0-up, 1-down equals for
   int d[2] = {-2, 2};
   if(in_field(d[direction%2])){
      remove();
      direction<2?center.y+=d[direction%2]:center.x+=d[direction%2];
      for(int i = 0; i<n; i++){
         direction<2?external[i].y+=d[direction%2]:external[i].x+=d[d
      }
      draw();
   }else{
      error();
   }
}
```

```
void Figure::rotate(int direction, int type){//direction: 0 - rotate
   int d[2] = {1, -1};
   remove();
   coords temp;
   int reduce = 0;
   rot>0?reduce = -1:reduce=1;
   if(type == 0){
      rot += d[direction];
      for(int i=0; i < n; i++){
         temp.x = center.x+(-center.x+external[i].x)*cos(d[direction]
         temp.y = center.y+(-center.x+external[i].x)*sin(d[direction]
         external[i] = temp;
      }
   }else{
      int temp_rot = rot;
      while(temp_rot != 0){
         for(int i=0; i < n; i++){
            temp.x = center.x+(-center.x+external[i].x)*cos(-reduce*M
            temp.y = center.y+(-center.x+external[i].x)*sin(-reduce*M
            external[i] = temp;
         }
         temp_rot += reduce;
      }
   }
   draw();
}

void Figure::change_size(int direction){
   int d[2] = {-2, 2};
   remove();
   if(in_field(d[direction])){
      r+=d[direction];
      calc_external_coords();
      rotate(0, 1);
   }else{
      error();
   }
}

void Figure::help(int type){//type: 0-about program, 1-move, 2-rotate
   setbkcolor(COLOR(61, 220, 151));
   bar(670, 45, 1000, 370);
   switch(type){
      case 0:
         outtextxy(675, 50, " !");
         outtextxy(675, 50+textheight("), "   ");
         outtextxy(675, 50+textheight(")*2, "");
         outtextxy(675, 50+textheight(")*3, ");
         break;
      case 1:
         outtextxy(675, 50, "  );
```

```
            outtextxy(675, 50+textheight("), "    ");
            outtextxy(675, 50+textheight(")*2, ");
            break;
        case 2:
            outtextxy(675, 50, "  );
            outtextxy(675, 50+textheight("), "    ;
            outtextxy(675, 50+textheight(")*2, "<-   ->) ");
            break;
        case 3:
            outtextxy(675, 50, "  );
            outtextxy(675, 50+textheight("), "    ;
            outtextxy(675, 50+textheight(")*2, "<-   ->) ");
            break;
    }
    setbkcolor(COLOR(235, 245, 238));
}

void Figure::error(){
    setcolor(RED);
    outtextxy(675, 5, "ERROR!!");
    delay(150);
    bar(674, 4, 676+textwidth("ERROR!!"), 6+textheight("ERROR!!"));
    setcolor(COLOR(70, 35, 122));
}

bool Figure::in_field(int dl){
    for(int i=0; i<n; i++){
        if(external[i].x + dl*10 >= 638) return false;
        if(external[i].x - dl*10 < 0) return false;
    }
    return true;
}

void save(){
    int width, height;
    IMAGE *output;

    width  = getmaxx() + 1;
    height = getmaxy() + 1;
    output = createimage(width, height);

    getimage(0, 0, width - 1, height - 1, output);
    saveBMP("output.jpg", output);
    freeimage(output);
}
```

Файл control.h

```
#ifndef CONTROL_H
#define CONTROL_H
#define dx 327
```

```
#define dy 62

enum CONTROLS {MOVE, ROTATE, RESIZE, SAVE, EXIT, N_CONTROLS};

struct button{
   int x0;
   int y0;
};

void create_button(int, int, int);
void create_bg(const char*);
int select_control();

#endif
```

Файл control.cpp

```
#include <graphics.h>
#include <control.h>

button Buttons[N_CONTROLS];

void create_button(int i, int x, int y){
   Buttons[i].x0 = x;
   Buttons[i].y0 = y;
}

void create_bg(const char *file_name){
   IMAGE *image;
   image = loadBMP(file_name);
   putimage(0, 0, image, COPY_PUT);
   freeimage(image);
}

int select_control(){
   int x, y;

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

   for (int i = 0; i < N_CONTROLS; i++)
   {
      if (x > Buttons[i].x0 && x < Buttons[i].x0 + dx &&
          y > Buttons[i].y0 && y < Buttons[i].y0 + dy)
      {
         return i;
      }
   }

   return -1;
}
```

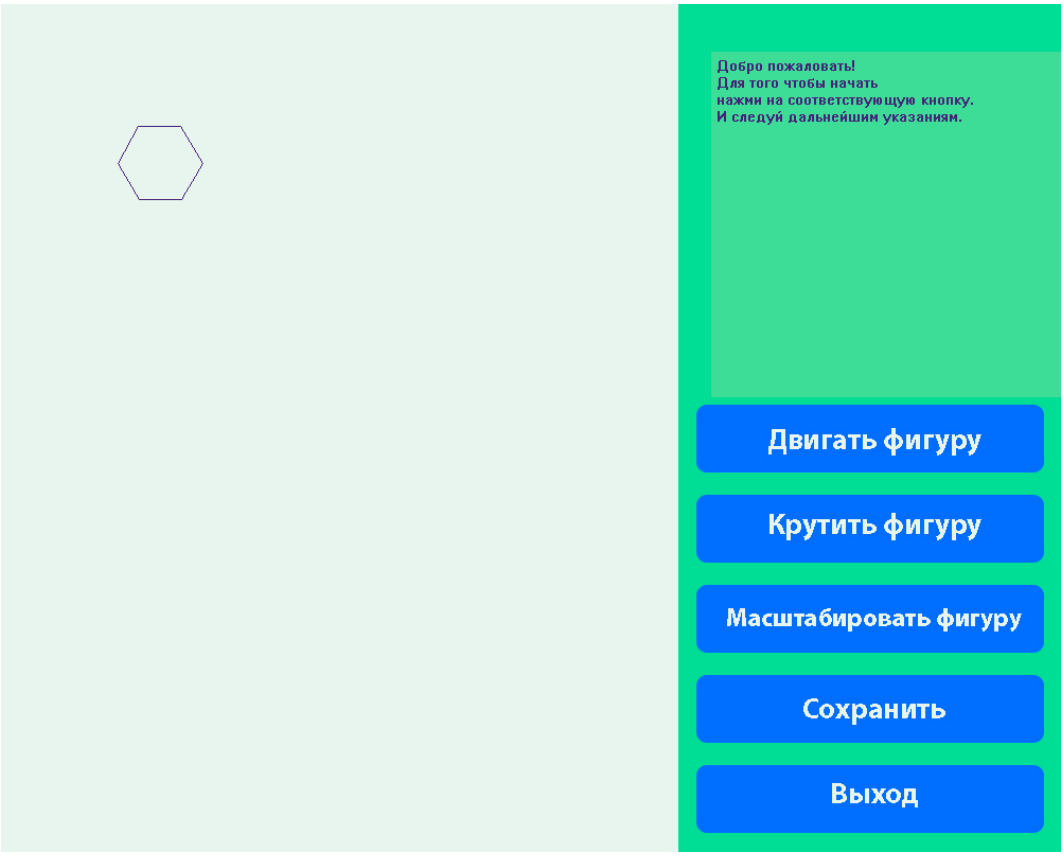

Рисунок 4.1 – Результат выполнения программы, после включения программы

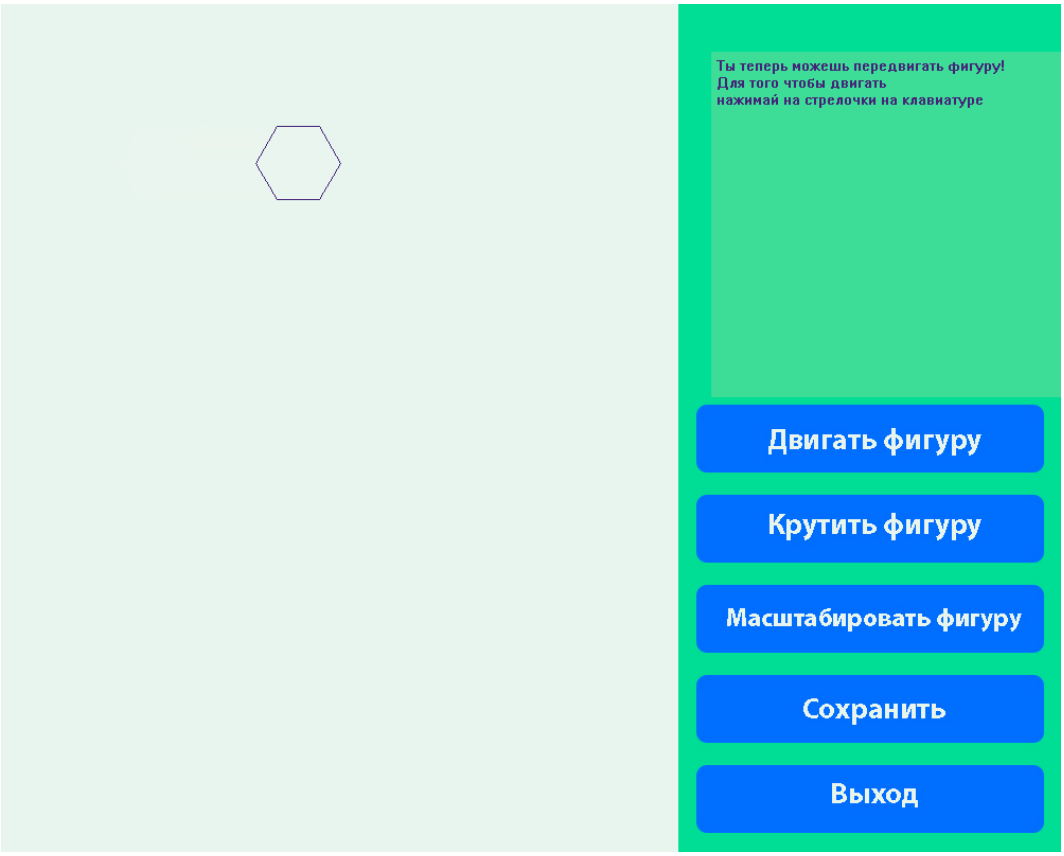

Рисунок 4.2 – Результат выполнения программы, после нажатия кнопки двигать фигуру

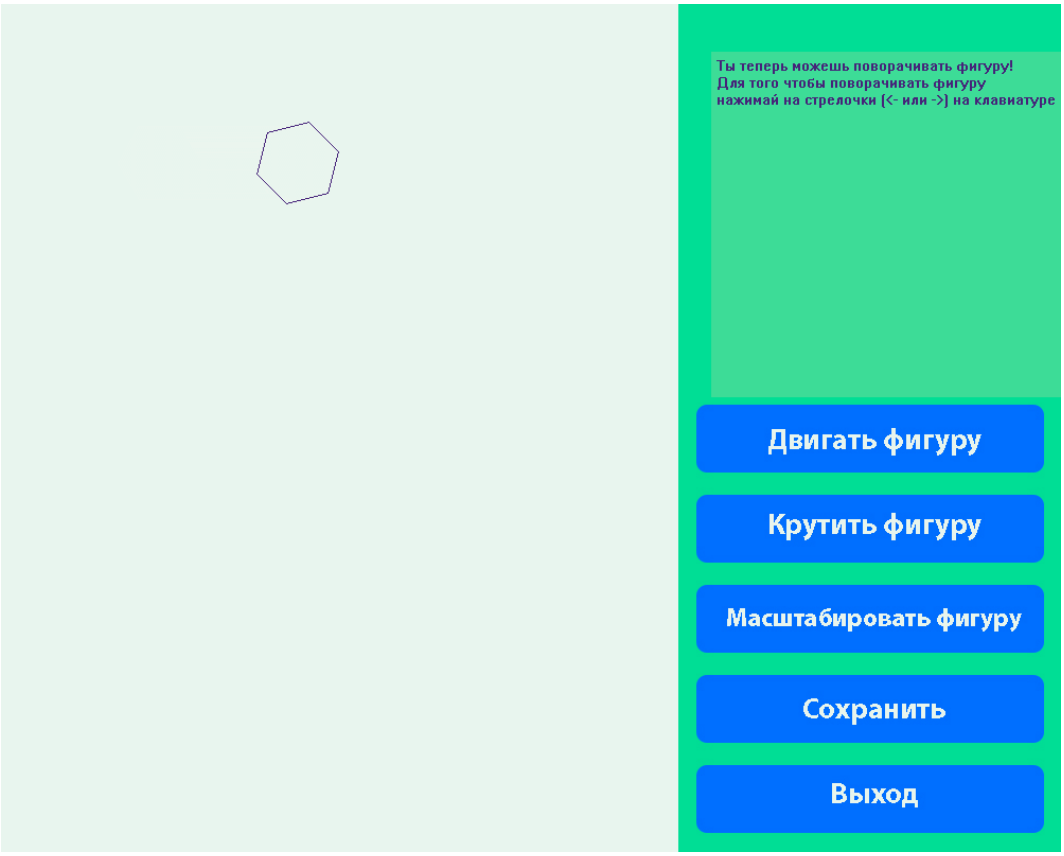

Рисунок 4.3 – Результат выполнения программы, после нажатия кнопки крутить фигуру

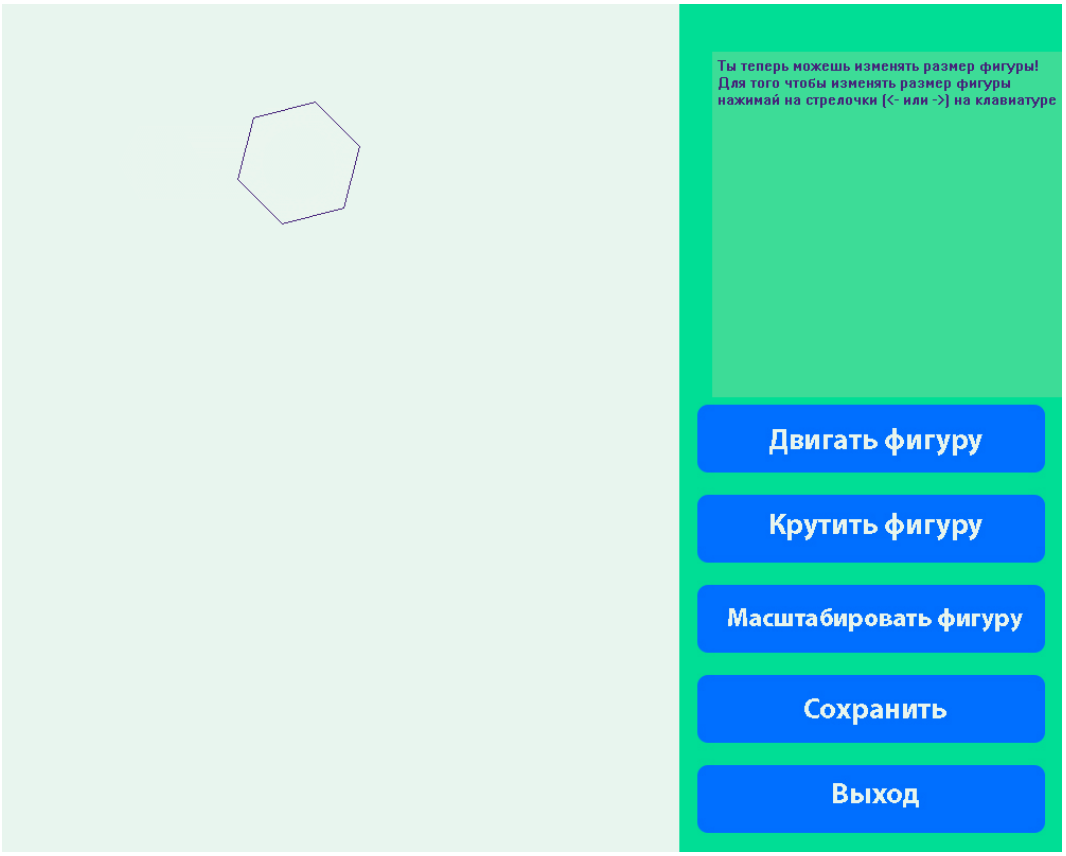

Рисунок 4.4 – Результат выполнения программы, после нажатия кнопки масштабировать фигуру

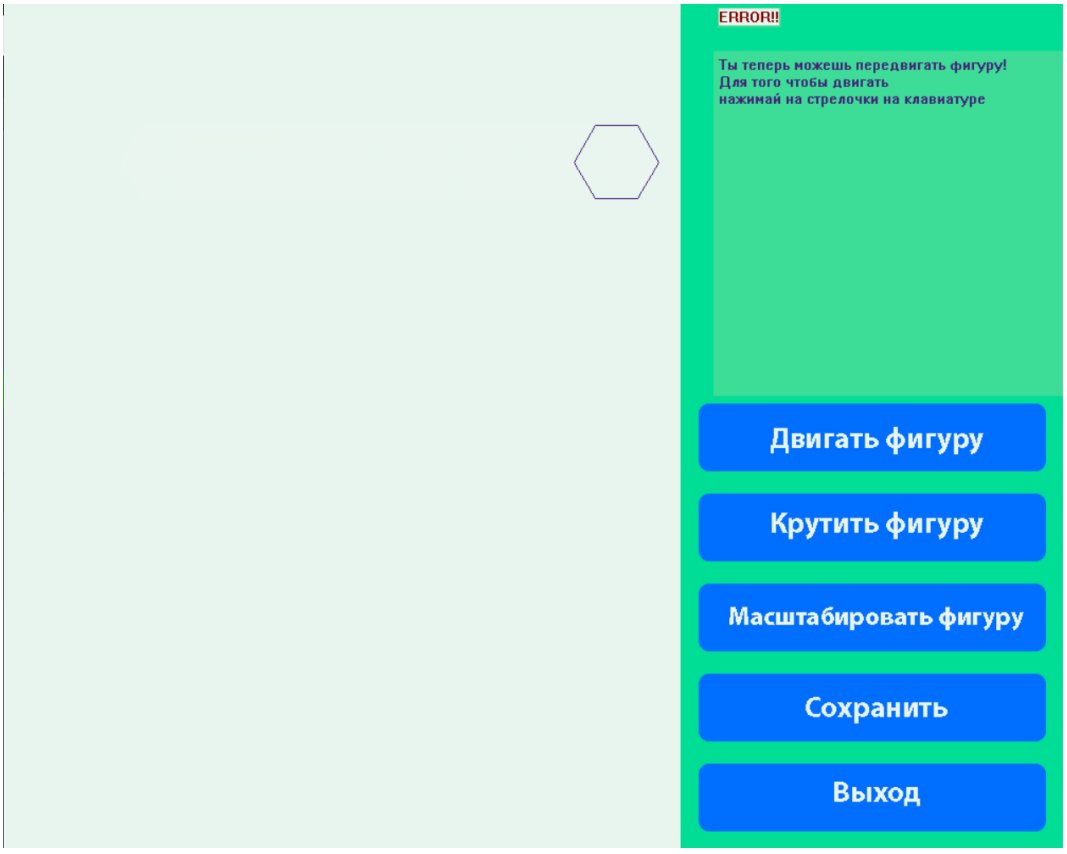

Рисунок 4.5 – Результат выполнения программы, случай попытки выхода за рабочий экран